Lecture Notes

Institute of Bioinformatics, Johannes Kepler University Linz

Bioinformatics II

Theoretical Bioinformatics and Machine Learning

Summer Semester 2009

by Sepp Hochreiter

Institute of Bioinformatics
Johannes Kepler University Linz
A-4040 Linz, Austria

Tel. +43 732 2468 8880
Fax +43 732 2468 9308
http://www.bioinf.jku.at

Literature

- Duda, Hart, Stork; Pattern Classification; Wiley & Sons, 2001
- C. M. Bishop; Neural Networks for Pattern Recognition, Oxford University Press, 1995
- Schölkopf, Smola; Learning with kernels, MIT Press, 2002
- V. N. Vapnik; Statistical Learning Theory, Wiley & Sons, 1998
- S. M. Kay; Fundamentals of Statistical Signal Processing, Prentice Hall, 1993
- M. I. Jordan (ed.); Learning in Graphical Models, MIT Press, 1998 (Original by Kluwer Academic Pub.)
- T. M. Mitchell; Machine Learning, Mc Graw Hill, 1997
- R. M. Neal, Bayesian Learning for Neural Networks, Springer, (Lecture Notes in Statistics), 1996
- Guyon, Gunn, Nikravesh, Zadeh (eds.); Feature Extraction - Foundations and Applications, Springer, 2006
- Schölkopf, Tsuda, Vert (eds.); Kernel Methods in Computational Biology, MIT Press, 2003

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This course is part of the curriculum of the master of science in bioinformatics at the Johannes Kepler University Linz. Machine learning has a major application in biology and medicine and many fields of research in bioinformatics are based on machine learning. For example one of the most prominent bioinformatics textbooks "Bioinformatics: The Machine Learning Approach" by P. Baldi and S. Brunak (MIT Press, ISBN 0-262-02506-X) sees the foundation of bioinformatics in machine learning.

Machine learning methods, for example neural networks used for the secondary and 3D structure prediction of proteins, have proven their value as essential bioinformatics tools. Modern measurement techniques in both biology and medicine create a huge demand for new machine learning approaches. One such technique is the measurement of mRNA concentrations with microarrays, where the data is first preprocessed, then genes of interest are identified, and finally predictions made. In other examples DNA data is integrated with other complementary measurements in order to detect alternative splicing, nucleosome positions, gene regulation, etc. All of these tasks are performed by machine learning algorithms. Alongside neural networks the most prominent machine learning techniques relate to support vector machines, kernel approaches, projection method and belief networks. These methods provide noise reduction, feature selection, structure extraction, classification / regression, and assist modeling. In the biomedical context, machine learning algorithms predict cancer treatment outcomes based on gene expression profiles, they classify novel protein sequences into structural or functional classes and extract new dependencies between DNA markers (SNP - single nucleotide polymorphisms) and diseases (schizophrenia or alcohol dependence).

In this course the most prominent machine learning techniques are introduced and their mathematical foundations are shown. However, because of the restricted space neither mathematical or practical details are presented. Only few selected applications of machine learning in biology and medicine are given as the focus is on the understanding of the machine learning techniques. If the techniques are well understood then new applications will arise, old ones can be improved, and the methods which best fit to the problem can be selected.

Students should learn how to chose appropriate methods from a given pool of approaches for solving a specific problem. Therefore they must understand and evaluate the different approaches, know their advantages and disadvantages as well as where to obtain and how to use them. In a step further, the students should be able to adapt standard algorithms for their own purposes or to modify those algorithms for specific applications with certain prior knowledge or special constraints.

Chapter 2

Basics of Machine Learning

The conventional approach to solve problems with the help of computers is to write programs which solve the problem. In this approach the programmer must understand the problem, find a solution appropriate for the computer, and implement this solution on the computer. We call this approach *deductive* because the human deduces the solution from the problem formulation. However in biology, chemistry, biophysics, medicine, and other life science fields a huge amount of data is produced which is hard to understand and to interpret by humans. A solution to a problem may also be found by a machine which learns. Such a machine processes the data and automatically finds structures in the data, i.e. learns. The knowledge about the extracted structure can be used to solve the problem at hand. We call this approach *inductive*, Machine learning is about inductively solving problems by machines, i.e. computers.

Researchers in machine learning construct algorithms that automatically improve a solution a problem with more data. In general the quality of the solution increases with the amount of problem-relevant data which is available.

Problems solved by machine learning methods range from classifying observations, predicting values, structuring data (e.g. clustering), compressing data, visualizing data, filtering data, selecting relevant components from data, extracting dependencies between data components, modeling the data generating systems, constructing noise models for the observed data, integrating data from different sensors,

Using classification a diagnosis based on the medical measurements can be made or proteins can be categorized according to their structure or function. Prediction support the current action through the knowledge of the future. A prominent example is stock market prediction but also prediction the outcome of therapy helps to choose the right therapy or to adjust the doses of the drugs. In genomics identifying the relevant genes for a certain investigation (gene selection) is important for understanding the molecular-biological dynamics in the cell. Especially in medicine the identification of genes related to cancer draw the attention of the researchers.

2.1 Machine Learning in Bioinformatics

Many problems in bioinformatics are solved using machine learning techniques.

Machine learning approaches to bioinformatics include:

- Protein secondary structure prediction (neural networks, support vector machines)

- Gene recognition (hidden Markov models)
- Multiple alignment (hidden Markov models, clustering)
- Splice site recognition (neural networks)
- Microarray data: normalization (factor analysis)
- Microarray data: gene selection (feature selection)
- Microarray data: prediction of therapy outcome (neural networks, support vector machines)
- Microarray data: dependencies between genes (independent component analysis, clustering)
- Protein structure and function classification (support vector machines, recurrent networks)
- Alternative splice site recognition (SVMs, recurrent nets)
- Prediction of nucleosome positions
- Single nucleotide polymorphism (SNP)
- Peptide and protein arrays
- Systems biology and modeling

For the last tasks like SNP data analysis, peptide or protein arrays and systems biology new approaches are developed currently.

For protein 3D structure prediction machine learning methods outperformed "threadingt't' methods in template identification (Cheng and Baldi, 2006).

Threading was the golden standard for protein 3D structure recognition if the structure is known (almost all structures are known).

Also for alternative splice site recognition machine learning methods are superior to other methods (Gunnar Rätsch).

2.2 Introductory Example

In the following we will consider a classification problem taken from "Pattern Classification", Duda, Hart, and Stork, 2001, John Wiley & Sons, Inc. In this classification problem salmons must be distinguished from sea bass given pictures of the fishes. Goal is that an automated system is able to separate the fishes in a fish-packing company, where salmons and sea bass are sold. We are given a set of pictures where experts told whether the fish on the picture is salmon or sea bass. This set, called *training set*, can be used to construct the automated system. The objective is that future pictures of fishes can be used to automatically separate salmon from sea bass, i.e. to classify the fishes. Therefore, the goal is to correctly classify the fishes in the future on unseen data. The performance on future novel data is called *generalization*. Thus, our goal is to maximize the generalization performance.

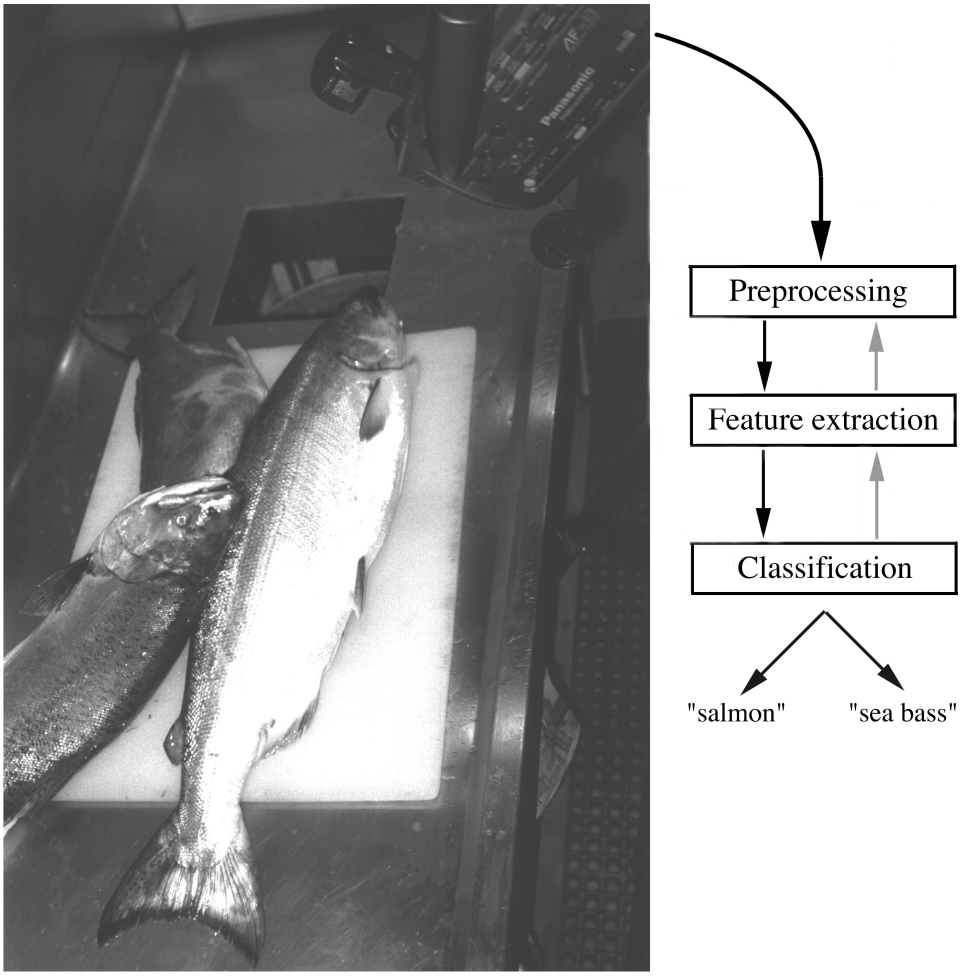


Figure 2.1: Salmons must be distinguished from sea bass. A camera takes pictures of the fishes and these pictures have to be classified as showing either a salmon or a sea bass. The pictures must be preprocessed and features extracted whereafter classification can be performed. Copyright © 2001 John Wiley & Sons, Inc.

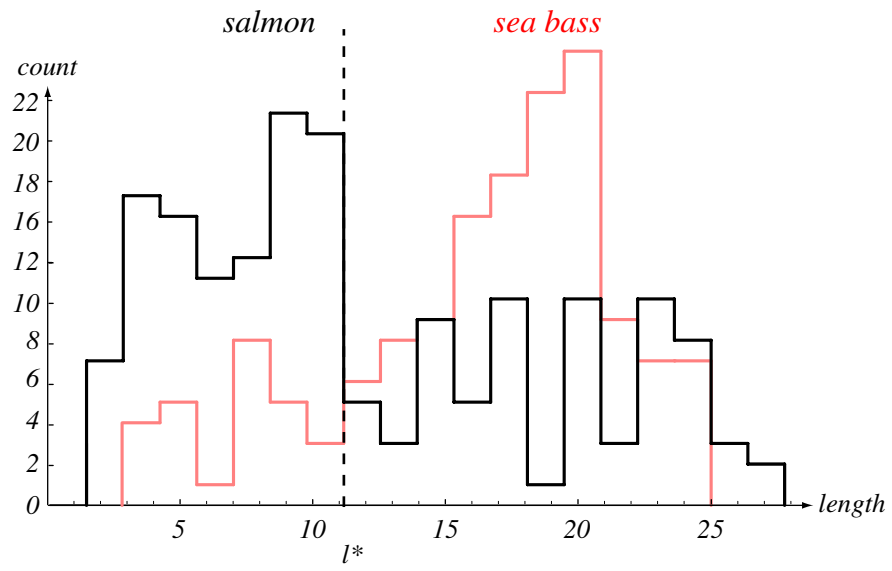


Figure 2.2: Salmon and see bass are separated by their length. Each vertical line gives a decision boundary l, where fish with length smaller than l are assumed to be salmon and others as sea bass. l^* gives the vertical line which will lead to the minimal number of misclassifications. Copyright © 2001 John Wiley & Sons, Inc.

Before the classification can be done the pictures must be preprocessed and features extracted. Classification is performed by the extracted features. See Fig. 2.1.

The preprocessing might involve contrast and brightness adjustment, correction of a brightness gradient in the picture, and segmentation to separate the fish from other fishes and from the background. Thereafter the single fish is aligned, i.e. brought in a predefined position. Now features of the single fish can be extracted. Features may be the length of the fish and its lightness.

First we consider the length in Fig. 2.2. We chose a decision boundary l, where fish with length smaller than l are assumed to be salmon and others as sea bass. The optimal decision boundary l^* is the one which will lead to the minimal number of misclassifications.

The second feature is the lightness of the fish. A histogram if using only this feature to decide about the kind of fish is given in Fig. 2.3.

For the optimal boundary we assumed that each misclassification is equally serious. However it might be that selling sea bass as salmon by accident is more serious than selling salmon as sea bass. Taking this into account we would chose an decision boundary which is on the left hand side of x^* in Fig. 2.3. Thus the cost function governs the optimal decision boundary.

As third feature we use the width of the fishes. This feature alone may not be a good choice to separate the kind of fishes, however we may have observed that the optimal separating lightness value depends on the width of the fishes. Perhaps the width is correlated with the age of the fish and the lightness of the fishes change with age. It might be a good idea to combine both features. The result is depicted in Fig. 2.4, where for each width an optimal lightness value is given. The optimal lightness value is a linear function of the width.

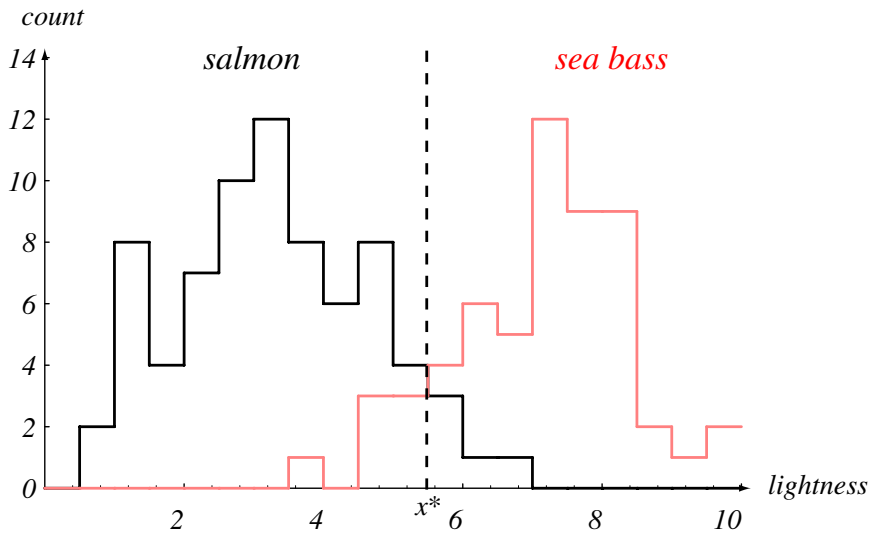


Figure 2.3: Salmon and see bass are separated by their lightness. x^* gives the vertical line which will lead to the minimal number of misclassifications. Copyright © 2001 John Wiley & Sons, Inc.

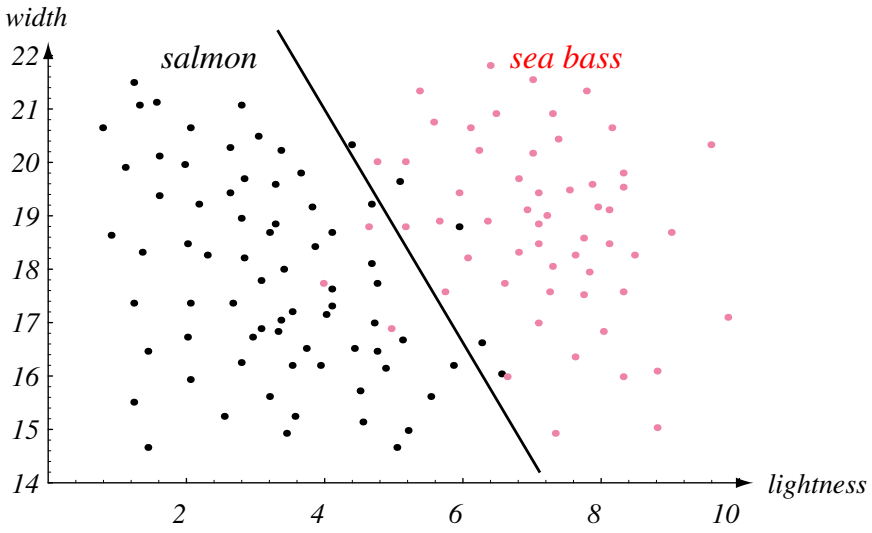


Figure 2.4: Salmon and see bass are separated by their lightness and their width. For each width there is an optimal separating lightness value given by the line. Here the optimal lightness is a linear function of the width. Copyright © 2001 John Wiley & Sons, Inc.

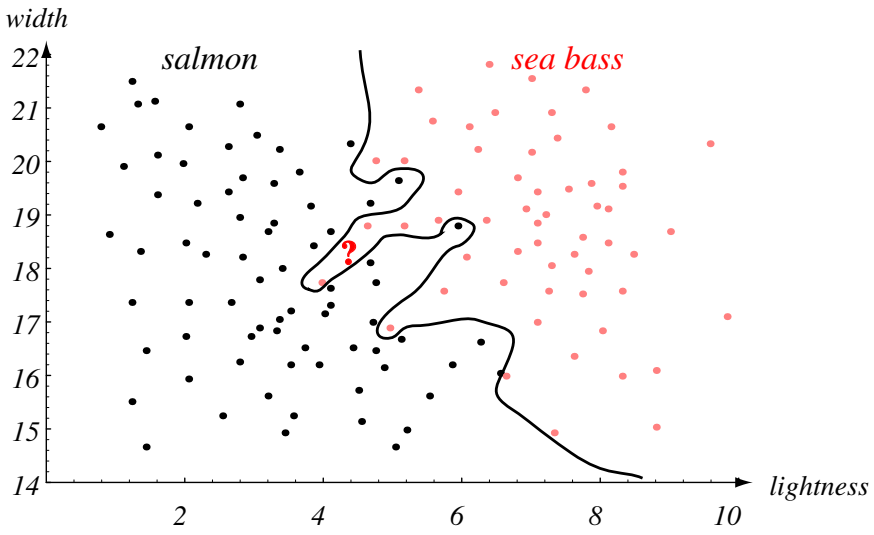


Figure 2.5: Salmon and see bass are separated by a nonlinear curve in the two-dimensional space spanned by the lightness and the width of the fishes. The training set is separated perfectly. A new fish with lightness and width given at the position of the question mark "?" would be assumed to be sea bass even if most fishes with similar lightness and width were previously salmon. Copyright © 2001 John Wiley & Sons, Inc.

Can we do better? The optimal lightness value may be a nonlinear function of the width or the optimal boundary may be a nonlinear curve in the two-dimensional space spanned by the lightness and the width of the fishes. The later is depicted in Fig. 2.5, where the boundary is chosen that every fish is classified correctly on the training set. A new fish with lightness and width given at the position of the question mark "?" would be assumed to be sea bass. However most fishes with similar lightness and width were previously classified as salmon by the human expert. At this position we assume that the generalization performance is low. One sea bass, an outlier, has lightness and width which are typically for salmon. The complex boundary curve also catches this outlier however must assign space without fish examples in the region of salmons to sea bass. We assume that future examples in this region will be wrongly classified as sea bass. This case will later be treated under the terms *overfitting*, *high variance*, *high model complexity*, and *high structural risk*.

A decision boundary, which may represent the boundary with highest generalization, is shown in Fig. 2.6.

In this classification task we selected the features which are best suited for the classification. However in many bioinformatics applications the number of features is large and selecting the best feature by visual inspections is impossible. For example if the most indicative genes for a certain cancer type must be chosen from 30,000 human genes. In such cases with many features describing an object *feature selection* is important. Here a machine and not a human selects the features used for the final classification.

Another issue is to construct new features from given features, i.e. *feature construction*. In above example we used the width in combination with the lightness, where we assumed that

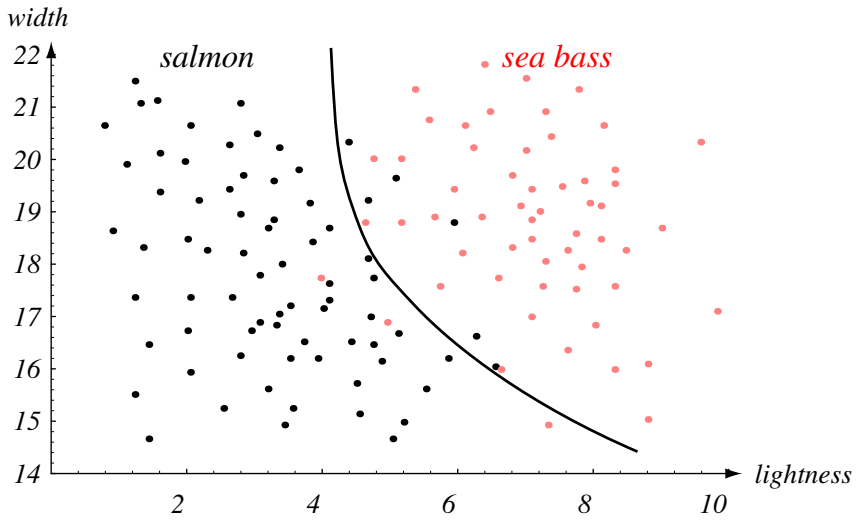


Figure 2.6: Salmon and see bass are separated by a nonlinear curve in the two-dimensional space spanned by the lightness and the width of the fishes. The curve may represent the decision boundary leading to the best generalization. Copyright © 2001 John Wiley & Sons, Inc.

the width indicates the age. However, first combining the width with the length may give a better estimate of the age which thereafter can be combined with the lightness. In this approach averaging over width and length may be more robust to certain outliers or to errors in processing the original picture. In general redundant features can be used in order to reduce the noise from single features.

Both feature construction and feature selection can be combined by randomly generate new features and thereafter select appropriate features from this set of generated features.

We already addressed the question of cost. That is how expensive is a certain error. A related issue is the kind of noise on the measurements and on the class labels produced in our example by humans. Perhaps the fishes on the wrong side of the boundary in Fig. 2.6 are just error of the human experts. Another possibility is that the picture did not allow to extract the correct lightness value. Finally, outliers in lightness or width as in Fig. 2.6 may be typically for salmons and sea bass.

2.3 Supervised and Unsupervised Learning

In previous example a human expert characterized the data, i.e. supplied the label (the class). Tasks, where the desired output for each object is given, are called *supervised* and the desired outputs are called *targets*. This term stems from the fact that during learning a model can obtain the correct value from the teacher, the supervisor.

If data has to be processed by machine learning methods, where the desired output is not given, then the learning task is called *unsupervised*. In supervised task one can immediately measure how good the model performs on the training data, because the optimal outputs, the targets, are

given. Further the measurement is done for each single object. That is the model supplies an error value on each object. In contrast to supervised problems, the quality of models on unsupervised problems is mostly measured on the cumulative output on all objects. Typically measurements for unsupervised methods include the information contents, the orthogonality of the constructed components, the statistical independence, the variation explained by the model, the probability that the observed data can be produced by the model (later introduced as *likelihood*), distances between and within clusters, etc.

Typical fields of supervised learning are classification, regression (assigning a real value to the data), or time series analysis (predicting the future). An examples for regression is to predict the age of the fish from above examples based on length, width and lightness. In contrast to classification the age is a continuous value. In a time series prediction task future values have to be predicted based on present and past values. For example a prediction task would be if we monitor the length, width and lightness of the fish every day (or every week) from its birth and want to predict its size, its weight or its health status as a grown out fish. If such predictions are successful appropriate fish can be selected early.

Typical fields of unsupervised learning are projection methods ("principal component analysis", "independent component analysis", "factor analysis", "projection pursuit"), clustering methods ("k-means", "hierarchical clustering", "mixture models", "self-organizing maps"), density estimation ("kernel density estimation", "orthonormal polynomials", "Gaussian mixtures") or generative models ("hidden Markov models", "belief networks"). Unsupervised methods try to extract structure in the data, represent the data in a more compact or more useful way, or build a model of the data generating process or parts thereof.

Projection methods generate a new representation of objects given a representation of them as a feature vector. In most cases, they down-project feature vectors of objects into a lower-dimensional space in order to remove redundancies and components which are not relevant. "Principal Component Analysis" (PCA) represents the object through feature vectors which components give the extension of the data in certain orthogonal directions. The directions are ordered so that the first direction give the direction of maximal data variance, the second the maximal data variance orthogonal to the first component, and so on. "Independent Component Analysis" (ICA) goes a step further than PCA and represents the objects through feature components which are statistically mutual independent. "Factor Analysis" extends PCA by introducing a Gaussian noise at each original component and assumes Gaussian distribution of the components. "Projection Pursuit" searches for components which are non-Gaussian, therefore, may contain interesting information. Clustering methods are looking for data cluster and, therefore, finding structure in the data. "Self-Organizing Maps" (SOMs) are a special kind of clustering methods which also perform a down-projection in order to visualize the data. The down-projection keeps the neighborhood of clusters. Density estimation methods attempt at producing the density from which the data was drawn. In contrast to density estimation methods generative models try to build a model which represents the density of the observed data. Goal is to obtain a world model if the density of the data points produced by the model matches the observed data density.

The clustering or (down-)projection methods may be viewed as feature construction methods because the object can now be described via the new components. For clustering the description of the object may contain the cluster to which it is closest or a whole vector describing the distances to the different clusters.

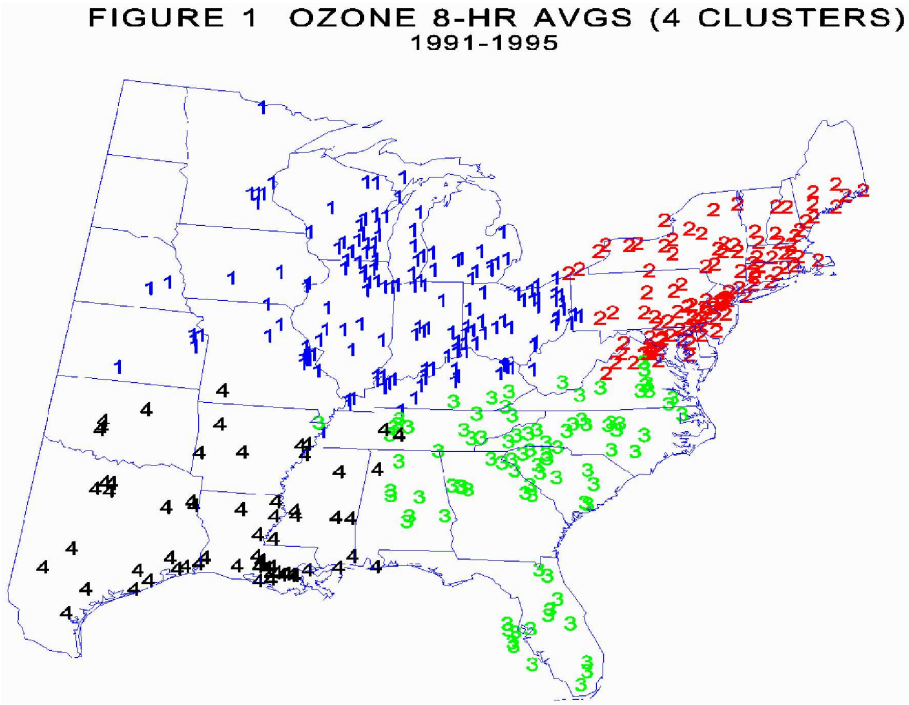


Figure 2.7: Example of a clustering algorithm. Ozone was measured and four clusters with similar ozone were found.

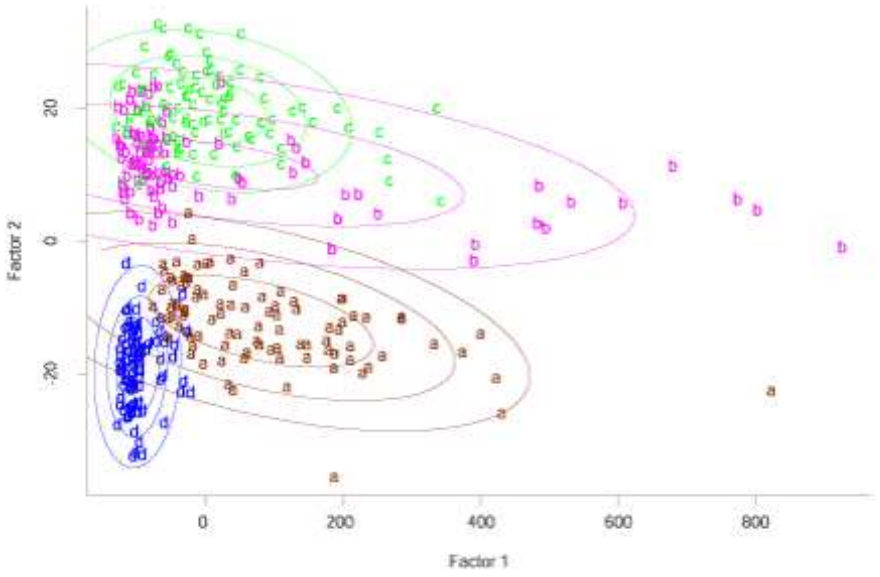


Figure 2.8: Example of a clustering algorithm where the clusters have different shape.

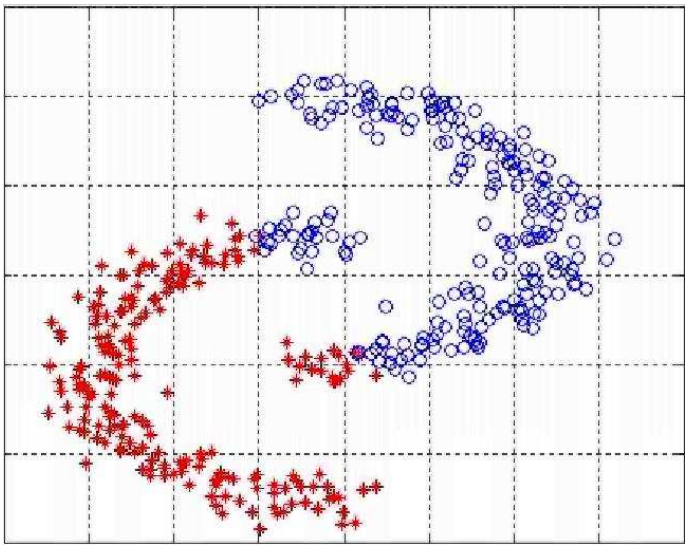


Figure 2.9: Example of a clustering where the clusters have a non-elliptical shape and clustering methods fail to extract the clusters.

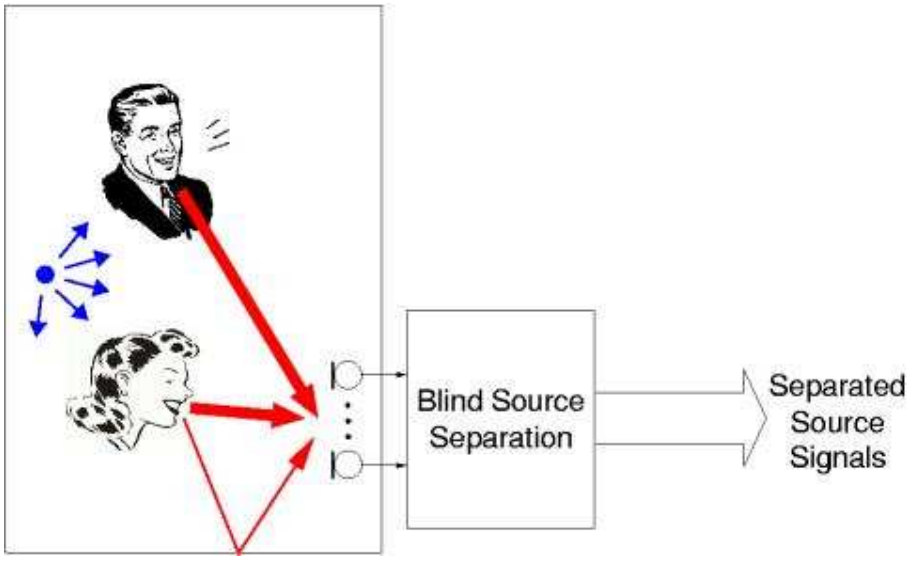


Figure 2.10: Two speakers recorded by two microphones. The speaker produce independent acoustic signals which can be separated by ICA (here called Blind Source Separation) algorithms.

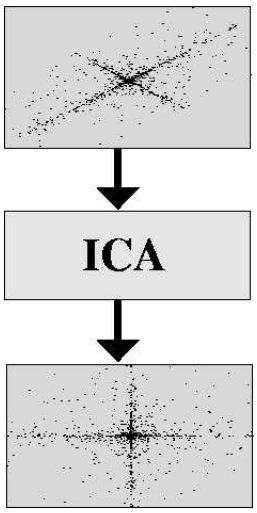


Figure 2.11: On top the data points where the components are correlated: knowing the x-coordinate helps to guess were the y-coordinate is located. The components are statistically dependent. After ICA the components are statistically independent.

2.4 Reinforcement Learning

There are machine learning methods which do not fit into the unsupervised/supervised classification.

For example, with *reinforcement learning* the model has to produce a sequence of outputs based on inputs but only receives a signal, a reward or a penalty, at sequence end or during the sequence. Each output influences the world in which the model, *the actor*, is located. These outputs also influence the current or future reward/penalties. The learning machine receives information about success or failure through the reward and penalties but does not know what would have been the best output in a certain situation. Thus, neither supervised nor unsupervised learning describes reinforcement learning. The situation is determined by the past and the current input.

In most scenarios the goal is to maximize the reward over a certain time period. Therefore, it may not be the best *policy*, that is the model in reinforcement learning, to maximize the immediate reward but to maximize the reward on a longer time scale. Many reinforcement algorithms build a *world model* which in then used to predict the future reward which in turn can be used to produce the optimal current output. In most cases the world model is a *value function* which estimates the expected current and future reward based on the current situation and the current output.

Most reinforcement algorithms can be divided into *direct policy optimization* and *policy / value iteration*. The former does not need a world model and in the later the world model is optimized for the current policy (the current model), then the policy is improved using the current world model, then the world model is improved based on the new policy, etc. The world model can only be build based on the current policy because the actor is part of the world.

Another problem in reinforcement learning is the *exploitation / exploration trade-off*. This addresses the question: is it better to optimize the reward based on the current knowledge or is it better to gain more knowledge in order to obtain more reward in the future.

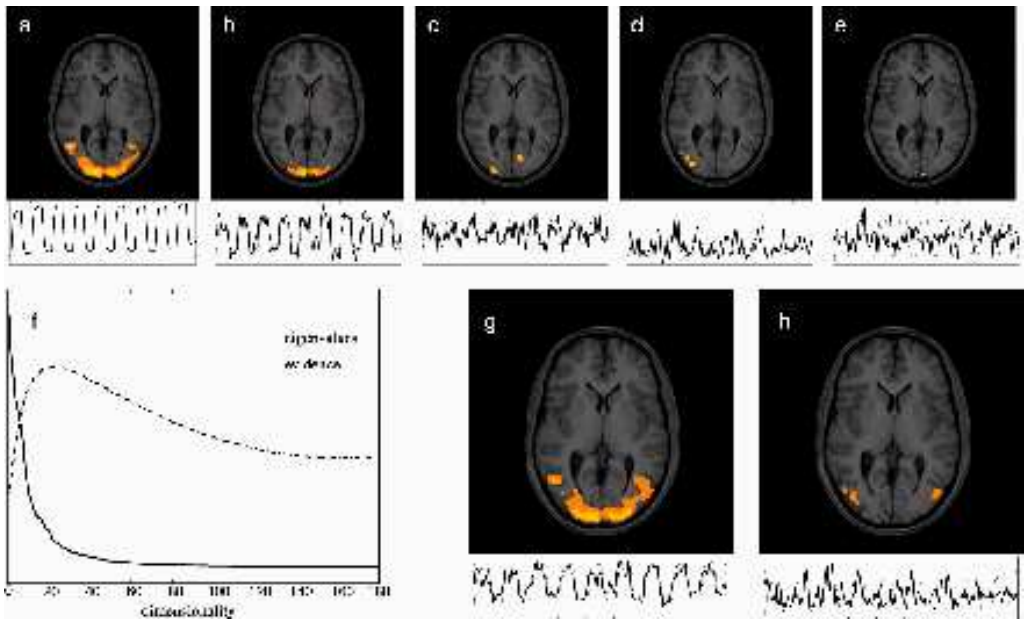


Figure 2.12: Images of fMRI brain data together with EEG data. Certain active brain regions are marked.

The most popular reinforcement algorithms are Q-learning, SARSA, Temporal Difference (TD), and monte carlo estimation.

Reinforcement learning will not be considered in this course because it has no application in bioinformatics until yet.

2.5 Feature Extraction, Selection, and Construction

As already mentioned in our example with the salmon and sea bass, features must be extracted from the original data. To generate features from the raw data is called *feature extraction*.

In our example features were extracted from images. Another example is given in Fig. 2.12 and Fig. 2.13 where brain patterns have to be extracted from fMRI brain images. In these figures also temporal patterns are given as EEG measurements from which also features can be extracted. Features from EEG patterns would be certain frequencies with their amplitudes whereas features from the fMRI data may be the activation level of certain brain areas which must be selected.

In many applications features are directly measured such features are length, weight, etc. In our fish example the length may not be extracted from images but is measured directly.

However there are task for which a huge number of features is available. In the bioinformatics contents examples are the microarray technique where 30,000 genes are measured simultaneously with cDNA arrays, peptide arrays, protein arrays, data from mass spectrometry, “single nucleotide” (SNP) data, etc. In such cases many measurements are not related to the task to be solved. For example only a few genes are important for the task (e.g. detecting cancer or prediction the outcome of a therapy) and all other genes are not. An example is given in Fig. 2.14, where one variable is related to the classification task an the other is not.

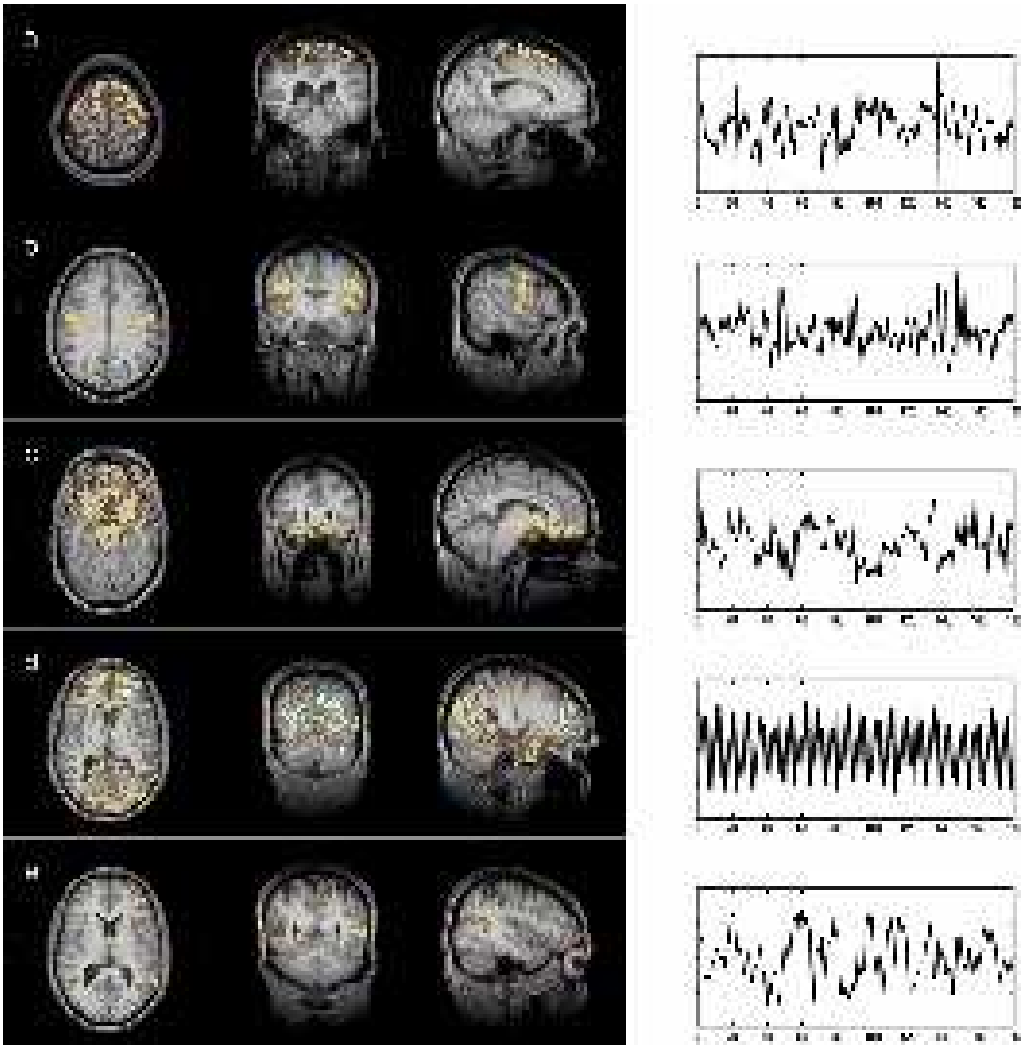


Figure 2.13: Another image of fMRI brain data together with EEG data. Again, active brain regions are marked.

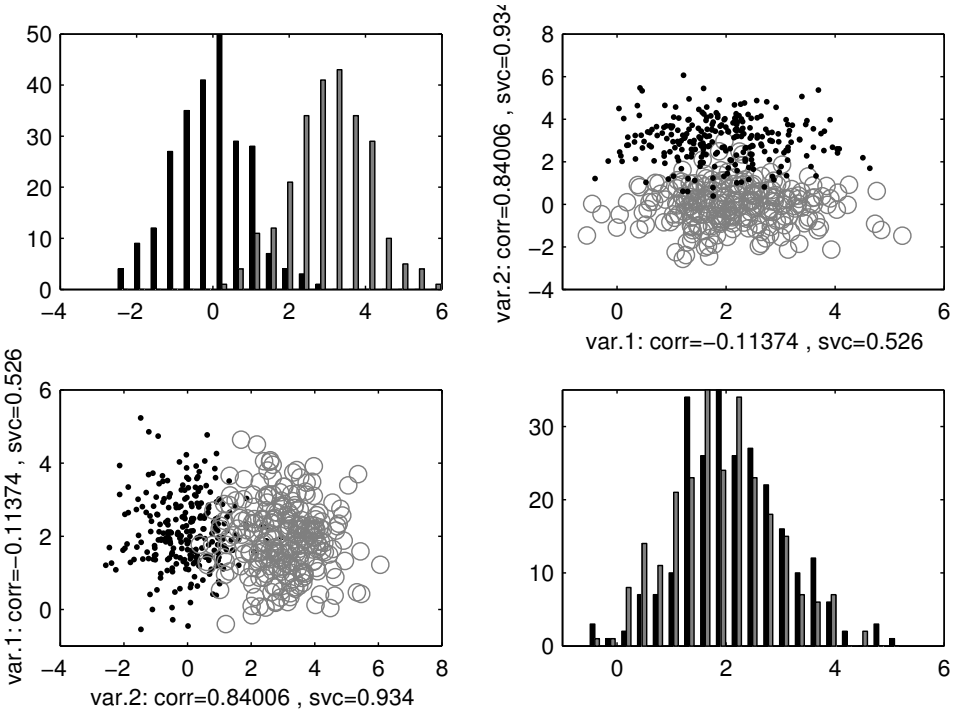


Figure 2.14: Simple two feature classification problem, where feature 1 (var. 1) is noise and feature 2 (var. 2) is correlated to the classes. In the upper right figure and lower left figure only the axis are exchanged. The upper left figure gives the class histogram along feature 2 whereas the lower right figure gives the histogram along feature 1. The correlation to the class (corr) and the performance of the single variable classifier (svc) is given.

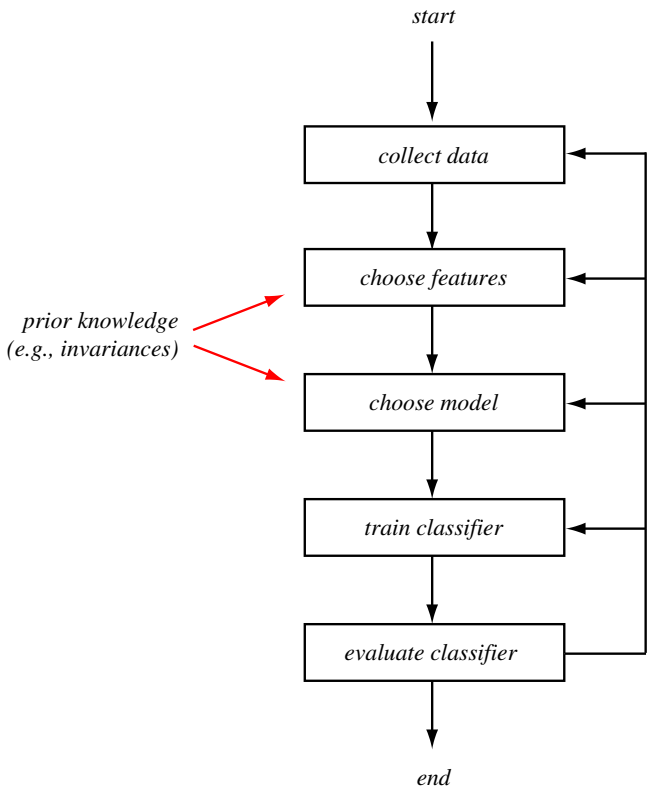


Figure 2.15: The design cycle for machine learning in order to solve a certain task. Copyright © 2001 John Wiley & Sons, Inc.

The first step of a machine learning approach would be to select the relevant features or chose a model which can deal with features not related to the task. Fig. 2.15 shows the design cycle for generating a model with machine learning methods. After collecting the data (or extracting the features) the features which are used must be chosen.

The problem of selecting the right variables can be difficult. Fig. 2.16 shows an example where single features cannot improve the classification performance but both features simultaneously help to classify correctly. Fig. 2.17 shows an example where in the left and right subfigure the features mean values and variances are equal for each class. However, the direction of the variance differs in the subfigures leading to different performance in classification.

There exist cases where the features which have no correlation with the target should be selected and cases where the feature with the largest correlation with the target should not be selected. For example, given the values of the left hand side in Tab. 2.1, the target t is computed from two features f_1 and f_2 as $t = f_1 + f_2$. All values have mean zero and the correlation coefficient between t and f_1 is zero. In this case f_1 should be selected because it has negative correlation with f_2. The top ranked feature may not be correlated to the target, e.g. if it contains target-independent information which can be removed from other features. The right hand side of Tab. 2.1 depicts another situation, where $t = f_2 + f_3$. f_1, the feature which has highest correlation coefficient with the target (0.9 compared to 0.71 of the other features) should not be

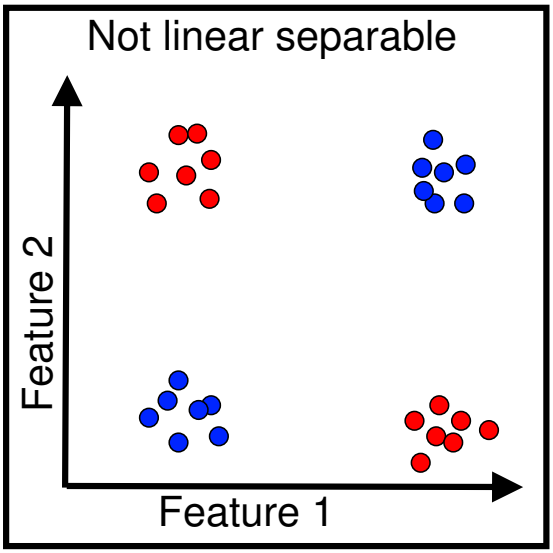


Figure 2.16: An XOR problem of two features, where each single feature is neither correlated to the problem nor helpful for classification. Only both features together help.

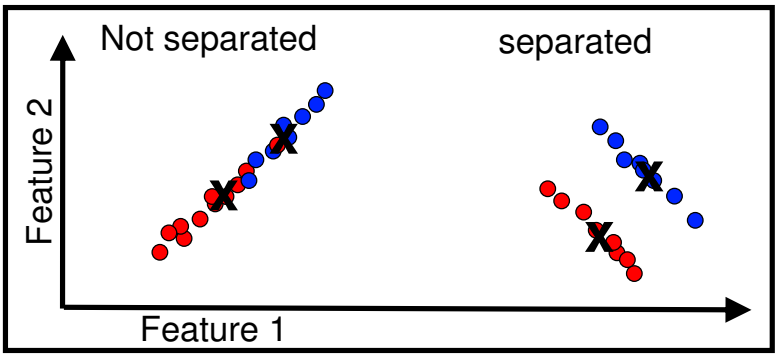


Figure 2.17: The left and right subfigure shows each two classes where the features mean value and variance for each class is equal. However, the direction of the variance differs in the subfigures leading to different performance in classification.

f_1	f_2	t	f_1	f_2	f_3	t
-2	3	1	0	-1	0	-1
2	-3	-1	1	1	0	1
-2	1	-1	-1	0	-1	-1
2	-1	1	1	0	1	1

Table 2.1: Left hand side: the target t is computed from two features f_1 and f_2 as $t = f_1 + f_2$. No correlation between t and f_1.

selected because it is correlated to all other features.

In some tasks it is helpful to combine some features to a new features, that is to construct features. In gene expression examples sometimes combining gene expression values to a meta-gene value gives more robust results because the noise is "averaged out". The standard way to combine linearly dependent feature components is to perform PCA or ICA as a first step. Thereafter the relevant PCA or ICA components are used for the machine learning task. Disadvantage is that often PCA or ICA components are no longer interpretable.

Using kernel methods the original features can be mapped into another space where implicitly new features are used. In this new space PCA can be performed (kernel-PCA). For constructing non-linear features out of the original one, prior knowledge on the problem to solve is very helpful. For example a sequence of nucleotides or amino acids may be presented by the occurrence vector of certain motifs or through their similarity to other sequences. For a sequence the vector of similarities to other sequences will be its feature vector. In this case features are constructed through alignment with other features.

Issues like missing values for some features or varying noise or non-stationary measurements have to be considered in selecting the features. Here features can be completed or modified.

2.6 Parametric vs. Non-Parametric Models

An important step in machine learning is to select the methods which will be used. This addresses the third step in Fig. 2.15. To "choose a model" is not correct as a model class must be chosen. Training and evaluation then selects an appropriate model from the model class. Model selection is based on the data which is available and on prior or domain knowledge.

A very common model class are *parametric models*, where each parameter vector represents a certain model. Parametric models are neural networks, where the parameter are the synaptic weights between the neurons, or support vector machines, where the parameters are the support vector weights. For parametric models in many cases it is possible to compute derivatives of the models with respect to the parameters. Here gradient directions can be used to change the parameter vector and, therefore, the model. If the gradient gives the direction of improvement then learning can be realized by paths through the parameter space.

Disadvantages of parametric models are: (1) one model may have two different parameterizations and (2) defining the model complexity and therefore choosing a model class must be done via the parameters. Case (1) can easily be seen at neural networks where the dynamics of one neuron

can be replaced by two neurons with the same dynamics each and both having outgoing synaptic connections which are half of the connections of the original neuron. Disadvantage is that not all neighboring models can be found because the model as more than one location in parameter space. Case (2) can also be seen at neural networks where model properties like smoothness or bounded output variance are hard to define through the parameters.

The counterpart of parametric models are *nonparametric models*. Using nonparametric models the assumption is that the model is locally constant or and superimpositions of constant models. Only by selecting the locations and the number of the constant models according to the data the models differ. Examples for nonparametric models are "k-nearest-neighbor", "learning vector quantization", or "kernel density estimator". These are local models and the behavior of the model to new data is determined by the training data which was close to this location. "k-nearest-neighbor" classifies the new data point according to the majority class of the k nearest neighbor training data points. "learning vector quantization" classifies a new data point according to the class assigned to the nearest cluster (nearest prototype). "kernel density estimator" computes the density at a new location proportional to the number and distance of training data points.

Another non-parametric model is the "decision tree". Here the locality principle is that each feature, i.e. each direction in the feature space, can split but both half-spaces obtain a constant value. In such a way the feature space can be partitioned into pieces (maybe with infinite edges) with constant function value.

However the constant models or the splitting rules must be a priori selected carefully using the training data, prior knowledge or knowledge about the complexity of the problem. For k-nearest-neighbor the parameter k and the distance measure must be chosen, for learning vector quantization the distance measure and the number of prototypes must be chosen, and for kernel density estimator the kernel (the local density function) must be adjusted where especially the width and smoothness of the kernel is an important property. For decision trees the splitting rules must be chosen a priori and also when to stop further portioning the space.

2.7 Generative vs. descriptive Models

In the previous section we mentioned the nonparametric approach of the kernel density estimator, where the model produces for a location the estimated density. And also for a training data point the density of its location is estimated, i.e. this data point has a new characteristic through the density at its location. We call this a *descriptive* model. Descriptive models supply an additional description of the data point or another representation. Therefore projection methods (PCA, ICA) are descriptive models as the data points are described by certain features (components).

Another machine learning approach to model selection is to model the data generating process. Such models are called *generative models*. Models are selected which produce the distribution observed for the real world data, therefore these models are describing or representing the data generation process. The data generation process may have also input components or random components which drive the process. Such input or random components may be included into the model. Important for the generative approach is to include as much prior knowledge about the world or desired model properties into the model as possible in order to restrict the number of models which can explain the observed data.

A generative model can be used predict the data generation process for unobserved inputs, to predict the behavior of the data generation process if its parameters are externally changed, to generate artificial training data, or to predict unlikely events. Especially the modeling approaches can give new insights into the working of complex systems of the world like the brain or the cell.

2.8 Prior and Domain Knowledge

In previous section we already mentioned to include as much prior and domain knowledge as possible into the model. Such knowledge helps in general. For example it is important to define reasonable distance measures for k-nearest-neighbor or clustering methods, to construct problem-relevant features, to extract appropriate features from the raw data, etc.

For kernel-based approaches prior knowledge in the field of bioinformatics include alignment methods, i.e. kernels are based on alignment methods like the string-kernel, the Smith-Waterman-kernel, the local alignment kernel, the motif kernel, etc. Or for secondary structure prediction with recurrent networks the 3.7 amino acid period of a helix can be taken into account by selecting as inputs the sequence elements of the amino acid sequence.

In the context of microarray data processing prior knowledge about the noise distribution can be used to build an appropriate model. For example it is known that the the log-values are more Gaussian distributed than the original expression values, therefore, mostly models for the log-values are constructed.

Different prior knowledge sources can be used in 3D structure prediction of proteins. The knowledge reaches from physical and chemical laws to empirical knowledge.

2.9 Model Selection and Training

Using the prior knowledge a model class can be chosen appropriate for the problem to be solved. In the next step a model from the model class must selected. The model with highest generalization performance, i.e. with the best performance on future data should be selected. The *model selection* is based on the training set, therefore, it is often called *training* or *learning*. In most cases a model is selected which best explains or approximates the training set.

However, as already shown in Fig. 2.5 of our salmon vs. sea bass classification task, if the model class is too large and a model is chosen which perfectly explains the training data, then the generalization performance (the performance on future data) may be low. This case is called *"overfitting"*. Reason is that the model is fitted or adapted to special characteristics of the training data, where these characteristics include noisy measurements, outliers, or labeling errors. Therefore before model selection based on the best training data fitting model, the model class must be chosen.

On the other hand, if a low complex model class is chosen, then it may be possible that the training data cannot be fitted well enough. The generalization performance may be low because the general structure in the data was not extracted because the model complexity did not allow to representing this structure. This case is called *"underfitting"*. Thus, the optimal generalization is a trade-off between underfitting and overfitting. See Fig. 2.18 for the trade-off between over- and underfitting error.

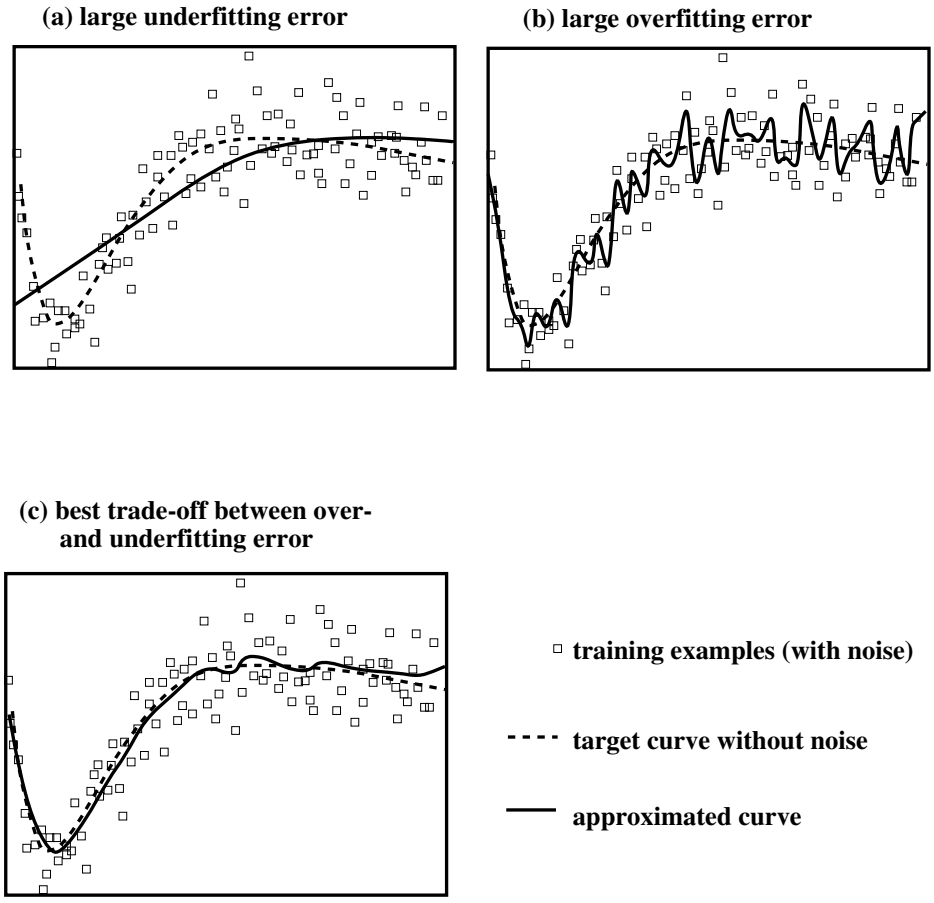


Figure 2.18: The trade-off between underfitting and overfitting is shown. The left upper subfigure shown underfitting, the right upper subfigure overfitting error, and the right lower subfigure shows the best compromise between both leading to the highest generalization (best performance on future data).

The model class can be chosen by the parameter k for k-nearest-neighbor, by the number of hidden neurons, their activation function and the maximal weight values for neural networks, by the value C penalizing misclassification and kernel (smoothness) parameters for support vector machines.

In most cases the model class must be chosen a priori to the training phase. However, for some methods, e.g. neural networks, the model class can be adjusted during training, where smoother decision boundaries correspond to lower model complexity.

In the context of "structural risk minimization" (see Section 3.6.4) the model complexity issue will be discussed in more detail.

Other choices before performing model selection concern the selection parameters, e.g. the learning rate for neural networks, the stopping criterion, precision values, number of iterations, etc.

Also the model selection parameters may influence the model complexity, e.g. if the model complexity is increased stepwise as for neural networks where the nonlinearity is increased during training. But also precision values may determine how exact the training data can be approximated and therefore implicitly influence the complexity of the model which is selected. That means also with a given model class the selection procedure may not be able to select all possible models.

The parameters controlling the model complexity and the parameters for the model selection procedure are called *"hyperparameters"* in contrast to the model parameters for parameterized models.

2.10 Model Evaluation, Hyperparameter Selection, and Final Model

In previous section we mentioned that before training/learning/model selection the hyperparameters must be chosen – but how? The same questions holds for choosing the best number of feature if feature selection was performed and a ranking of the features is provided.

For special cases the hyperparameters can be chosen with some assumptions and global training data characteristics. For example kernel density estimation (KDE) has as hyperparameter the width of the kernels which can be chosen using an assumption about the smoothness (peakiness) of the target density and the number of training examples.

However in general the hyperparameters must be estimated from the training data. In most cases they are estimated by *n-fold cross-validation*. The procedure of n-fold cross-validation first divides the training set into n equal parts. Then one part is removed and the remaining $(n-1)$ parts are used for training/model selection whereafter the selected model is evaluated on the removed part. This can be done n times because there are n parts which can be removed. The error or empirical risk (see definition eq. (3.161) in Section 3.6.2) on this n times evaluation is the n-fold cross-validation error.

The cross-validation error is supposed to approximate the generalization error by withholding a part of the training set and observing how well the model would have performed on the withhold data. However, the estimation is not correct from the statistical point of view because the values which are used to estimate the generalization error are dependent. The dependencies came from two facts. First the evaluation of different folds are correlated because the cross validation

training sets are overlapping (an outlier would influence more than one cross validation training set). Secondly the results on data points of the removed fold on which the model is evaluated are correlated because they use the same model (if the model selection is bad then all points in the fold are affected).

A special case of n-fold cross-validation is *leave-one-out cross validation*, where n is equal to the number of data points, therefore, only one data point is removed and the model evaluated on this data point.

Coming back to the problem of selecting the best hyperparameters. A set of specific hyperparameters can be evaluated by cross-validation on the training set. Thereafter the best performing hyperparameters are chosen to train the final model on all available data.

We evaluated the hyperparameters on a training set of size $\frac{n-1}{n}$ of the final training set. Therefore, methods which are sensitive to the size of the training set must be treated carefully.

In many cases the user wants to know how well a method will perform in future or wants to compare different methods. Can we use the performance of our method on the best hyperparameters as an estimate of the performance of our method in the future? No! We have chosen the best performing hyperparameters on the n-fold cross validation based on the training set which do in general not match the performance on future data.

To estimate the performance of a method we can use cross-validation, but for each fold we have to do a separate cross-validation run for hyperparameter selection.

Also for selection of the number of features we have to proceed as for the hyperparameters. And hyperparameter selection becomes a hyperparameter-feature selection, i.e. each combination of hyperparameters and number of features must be tested. That is reasonable as hyperparameters may depend on the input dimension.

A well know error in estimating the performance of a method is to select features on the whole available data set and thereafter perform cross-validation. However features are selected with the knowledge of future data (the removed data in the cross-validation procedure). If the data set contains many features then this error may considerably increase the performance. For example, if genes are selected on the whole data set then for the training set of a certain fold from all features which have the same correlation with the target on the training set those features are ranked higher which also show correlation with the test set (the removed fold). From all genes which are up-regulated for all condition 1 and down-regulated for all condition 2 cases on the training set those which show the same behavior on the removed fold are ranked highest. In practical applications, however, we do not know what will be the conditions of future samples.

If comparing different methods it is important to test whether the observed differences in the performances are significant or not. The tests for accessing whether one method is significantly better than another may have two types of error type I error and type II error. Type I errors detect a difference in the case that there is no difference between the methods. Type II errors miss a difference if there was a difference. Here it turned out that a paired t-test has a high probability of type I errors. The paired t-test is performed by multiply dividing the data set into test and training set and training both methods on the training set and evaluating them on the test set. The k-fold cross-validated paired t-test (instead of randomly selecting the test set cross-validation is used) behaves better than the paired t-test but is inferior to McNemar's test and 5x2CV (5 times two fold cross-validation) test.

Another issue in comparing methods is their time and space complexity. Time complexity is most important as main memory is large these days. We must distinguish between learning and testing complexity – the later is the time required if the method is applied to new data. For training complexity two arguments are often used.

On the one hand, if training last long like a day or a week it does not matter in most applications if the outcome is appropriate. For example if we train a stock market prediction tool a whole week and make a lot of money, it will not matter whether we get this money two days earlier or later. On the other hand, if one methods is 10 times faster than another method, it can be averaged over 10 runs and its performance is supposed to be better. Therefore training time can be discussed diversely.

For the test time complexity other arguments hold. For example if the method is used online or as a web service then special requirements must be fulfilled. For example if structure or secondary structure prediction takes too long then the user will not use such web services. Another issue is large scale application like searching in large databases or processing whole genomes. In such cases the application dictates what is an appropriate time scale for the methods. If analyzing a genome takes two years then such a method is not acceptable but one week may not matter.

Chapter 3

Theoretical Background of Machine Learning

In this chapter we focus on the theoretical background of learning methods.

First we want to define quality criteria for selected models in order to pin down a goal for model selection, i.e. learning. In most cases the quality criterion is not computable and we have to find approximations to it. The definition of the quality criterion first focuses on supervised learning.

For unsupervised learning we introduce Maximum Likelihood as quality criterion. In this context we introduce concepts like bias and variance, efficient estimator, and the Fisher information matrix.

Next we revisit supervised learning but treat it as an unsupervised Maximum Likelihood approach using an error model. Here the kind of measurement noise determines the error model which in turn determines the quality criterion of the supervised approach. Here also classification methods with binary output can be treated.

A central question in machine learning is: Does learning from examples help in the future? Obviously, learning helps humans to master the environment they live in. But what is the mathematical reason for that? I might be that task in the future are unique and nothing from the past helps to solve them. Future examples may be different from examples we have already seen.

Learning on the training data is called "empirical risk minimization" (ERM) in statistical learning theory. ERM results that if the complexity is restricted and the dynamics of the environment does not change, learning helps. "Learning helps" means that with increasing number of training examples the selected model converges to the best model for all future data. Under mild conditions the convergence is uniform and even fast, i.e. exponentially. These theoretical theorems found the idea of learning from data because with finite many training examples a model can be selected which is close to the optimal model for future data. How close is governed by the number of training examples, the complexity of the task including noise, the complexity of the model, and the model class.

To measure the complexity of the model we will introduce the VC-dimension (Vapnik-Chervonenkis).

Using model complexity and the model quality on the training set, theoretical bounds on the generalization error, i.e. the performance on future data, will be derived. From these bounds the

principle of "structural risk minimization" will be derived to optimize the generalization error through training.

The last section is devoted to techniques for minimizing the error that is techniques for model selection for a parameterized model class. Here also on-line methods are treated, i.e. methods which do not require a training set but attempt at improving the model (selecting a better model) using only one example at a certain time point.

3.1 Model Quality Criteria

Learning in machine learning is equivalent with model selection. A model from a set of possible models is chosen and will be used to handle future data.

But what is the best model? We need a quality criteria in order to choose a model. The quality criteria should be such that future data is optimally processed with the model. That would be the most common criterion.

However in some cases the user is not interested in future data but only want to visualize the current data or extract structures from the current data, where these structures are not used for future data but to analyze the current data. Topics which are related to the later criteria are data visualization, modeling, data compression. But in many cases the model with best visualization, best world explanation, or highest compression rate is the model where rules derived on a subset of the data can be generalized to the whole data set. Here the rest of the data can be interpreted as future data. Another point of view may be to assume that future data is identical with the training set. These considerations allow also to treat the later criteria also with the former criteria.

Some machine learning approaches like Kohonen networks don't possess a quality criterion as a single scalar value but minimize a potential function. Problem is that different models cannot be compared. Some ML approaches are known to converge during learning to the model which really produces the data if the data generating model is in the model class. But these approaches cannot supply a quality criterion and the quality of the current model is unknown.

The performance on future data will serve as our quality criterion which possesses the advantages of being able to compare models and to know the quality during learning which gives in turn a hint when to stop learning.

For supervised data the performance on future data can be measured directly, e.g. for classification the rate of misclassifications or for regression the distance between model output, the prediction, and the correct value observed in future.

For unsupervised data the quality criterion is not as obvious. The criterion cannot be broke down to single examples as in the supervised case but must include all possible data with its probability for being produced by the data generation process. Typical, quality measures are the likelihood of the data being produced by the model, the ratio of between and within cluster distances in case of clustering, the independence of the components after data projection in case of ICA, the information content of the projected data measured as non-Gaussianity in case of projection pursuit, expected reconstruction error in case of a subset of PCA components or other projection methods.

3.2 Generalization error

In this section we define the performance of a model on future data for the supervised case. The performance of a model on future data is called *generalization error*. For the supervised case an error for each example can be defined and then averaged over all possible examples. The error on one example is called *loss* but also *error*. The expected loss is called *risk*.

3.2.1 Definition of the Generalization Error / Risk

We assume that *objects* $x \in \mathcal{X}$ from an object set $\mathcal{X}$ are represented or described by *feature vectors* $\boldsymbol{x} \in \mathbb{R}^d$.

The *training set* consists of l objects $X = \left\{x^1, \ldots, x^l\right\}$ with a characterization $y^i \in \mathbb{R}$ like a label or an associated value with must be predicted for future objects. For simplicity we assume that y^i is a scalar, the so-called *target*. For simplicity we will write $\boldsymbol{z} = (\boldsymbol{x}, y)$ and $Z = X \times \mathbb{R}$.

The *training data* is $\left\{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\right\}$ ($\boldsymbol{z}^i = (\boldsymbol{x}^i, y^i)$), where we will later use the *matrix of feature vectors* $\boldsymbol{X} = \left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\right)^T$, the *vector of labels* $\boldsymbol{y} = \left(y^1, \ldots, y^l\right)^T$, and the *training data matrix* $\boldsymbol{Z} = \left(\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\right)$ ("^{T}" means the transposed of a matrix and here it makes a column vector out of a row vector).

In order to compute the performance on the future data we need to know the future data and need a quality measure for the deviation of the prediction from the true value, i.e. a *loss function*.

The future data is not known, therefore, we need at least the probability that a certain data point is observed in the future. The data generation process has a density $p(\boldsymbol{z})$ at $\boldsymbol{z}$ over its data space. For finite discrete data $p(\boldsymbol{z})$ is the probability of the data generating process to produce $\boldsymbol{z}$. $p(\boldsymbol{z})$ is the *data probability*.

The loss function is a function of the target and the model prediction. The model prediction is given by a function $g(\boldsymbol{x})$ and if the models are parameterized by a parameter vector $\boldsymbol{w}$ the model prediction is a parameterized function $g(\boldsymbol{x}; \boldsymbol{w})$. Therefore the loss function is $L(y, g(\boldsymbol{x}; \boldsymbol{w}))$. Typical loss functions are the *quadratic loss* $L(y, g(\boldsymbol{x}; \boldsymbol{w})) = (y - g(\boldsymbol{x}; \boldsymbol{w}))^2$ or the zero-one loss function

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = \begin{cases} 0 & \text{for} \quad y = g(\boldsymbol{x}; \boldsymbol{w}) \\ 1 & \text{for} \quad y \neq g(\boldsymbol{x}; \boldsymbol{w}) \end{cases} . \tag{3.1}$$

Now we can define the *generalization error* which is the expected loss on future data, also called *risk* R (a functional, i.e. a operator which maps functions to scalars):

$$R(g(.; \boldsymbol{w})) = \mathrm{E}_{\boldsymbol{z}} \left(L(y, g(\boldsymbol{x}; \boldsymbol{w}))\right) . \tag{3.2}$$

The risk for the quadratic loss is called *mean squared error*.

$$R(g(.; \boldsymbol{w})) = \int_Z L(y, g(\boldsymbol{x}; \boldsymbol{w}))\, p(\boldsymbol{z})\, d\boldsymbol{z} . \tag{3.3}$$

In many cases we assume that y is a function of $\boldsymbol{x}$, the *target function* $f(x)$, which is disturbed by noise

$$y \;=\; f(\boldsymbol{x}) \;+\; \epsilon\,, \tag{3.4}$$

where ϵ is a noise term drawn from a certain distribution $p_n(\epsilon)$, thus

$$p(y \mid \boldsymbol{x}) \;=\; p_n(y \;-\; f(\boldsymbol{x}))\,. \tag{3.5}$$

Here the probabilities can be rewritten as

$$p\,(\boldsymbol{z}) \;=\; p\,(\boldsymbol{x})\;p(y \mid \boldsymbol{x}) \;=\; p\,(\boldsymbol{x})\;p_n(y \;-\; f(\boldsymbol{x}))\,. \tag{3.6}$$

Now the risk can be computed as

$$\begin{aligned} R(g(.;\boldsymbol{w})) \;=\; \int_Z L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p\,(\boldsymbol{x})\;p_n(y \;-\; f(\boldsymbol{x}))\, d\boldsymbol{z} \;= \\ \int_X p\,(\boldsymbol{x}) \int_{\mathbb{R}} L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p_n(y \;-\; f(\boldsymbol{x}))\, dy\, d\boldsymbol{x}\,, \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} R(g(\boldsymbol{x};\boldsymbol{w})) \;=\; \mathrm{E}_{y|\boldsymbol{x}}\left(L(y, g(\boldsymbol{x};\boldsymbol{w}))\right) \;= \\ \int_{\mathbb{R}} L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p_n(y \;-\; f(\boldsymbol{x}))\, dy\,. \end{aligned} \tag{3.8}$$

The noise-free case is $y \;=\; f(\boldsymbol{x})$, where $p_n \;=\; \delta$ can be viewed as a Dirac delta-distribution:

$$\int_{\mathbb{R}} h(\boldsymbol{x})\delta(\boldsymbol{x})d\boldsymbol{x} \;=\; h(\mathbf{0}) \tag{3.9}$$

therefore

$$R(g(\boldsymbol{x};\boldsymbol{w})) \;=\; L(f(\boldsymbol{x}), g(\boldsymbol{x};\boldsymbol{w})) \;=\; L(y, g(\boldsymbol{x};\boldsymbol{w})) \tag{3.10}$$

and eq. (3.3) simplifies to

$$R(g(.;\boldsymbol{w})) \;=\; \int_X p\,(\boldsymbol{x})\, L(f(\boldsymbol{x}), g(\boldsymbol{x};\boldsymbol{w}))d\boldsymbol{x}\,. \tag{3.11}$$

Because we do not know $p\,(\boldsymbol{z})$ the risk cannot be computed; especially we do not know $p(y \mid \boldsymbol{x})$. In practical applications we have to approximate the risk.

To be more precise $\boldsymbol{w} \;=\; \boldsymbol{w}\,(\boldsymbol{Z})$, i.e. the parameters depend on the training set.

3.2.2 Empirical Estimation of the Generalization Error

Here we describe some methods how to estimate the risk (generalization error) for a certain model.

3.2.2.1 Test Set

We assume that data points $\boldsymbol{z} = (\boldsymbol{x}, y)$ are iid (independent identical distributed) and, therefore also $L(y, g(\boldsymbol{x}; \boldsymbol{w}))$, and $E_{\boldsymbol{z}}(|L(y, g(\boldsymbol{x}; \boldsymbol{w}))|) < \infty$.

The risk is an expectation of the loss function:

$$R(g(.; \boldsymbol{w})) = \mathrm{E}_{\boldsymbol{z}}(L(y, g(\boldsymbol{x}; \boldsymbol{w}))) , \tag{3.12}$$

therefore this expectation can be approximated using the (strong) law of large numbers:

$$R(g(.; \boldsymbol{w})) \approx \frac{1}{m} \sum_{i=l+1}^{l+m} L\left(y^i, g(\boldsymbol{x}^i; \boldsymbol{w})\right) , \tag{3.13}$$

where the set of m elements $\left\{\boldsymbol{z}^{l+1}, \ldots, \boldsymbol{z}^{l+m}\right\}$ is called *test set*.

Disadvantage of the test set method is, that the test set cannot be used for learning because $\boldsymbol{w}$ is selected using the training set and, therefore, $L(y, g(\boldsymbol{x}; \boldsymbol{w}))$ is not iid for training data points. Intuitively, if the loss is low for some training data points then we will expect that the loss will also be low for the following training data points.

3.2.2.2 Cross-Validation

If we have only few data points available we want to use them all for learning and not for estimating the performance via a test set. But we want to estimate the performance for our final model.

We can divide the available data multiple into training data and test data and average over the result. Problem here is that the test data is overlapping and we estimate with dependent test data points.

To avoid overlapping test data points we divide the training set into n parts (see Fig. 3.1). Then we make n runs where for the ith run part no. i is used for testing and the remaining parts for training (see Fig. 3.2). That procedure is called *n-fold cross-validation*. The *cross-validation risk* $R_{n-\mathrm{cv}}(\boldsymbol{Z})$ is the cumulative loss over all folds used for testing.

A special case of cross-validation is *leave-one-out cross-validation* (LOO CV) where $n = l$ and a fold contains only one element.

The cross-validation risk is a nearly (almost) unbiased estimate for the risk.

Unbiased means that the expected cross-validation error is equal the expected risk, where the expectation is is over training sets with l elements.

We will write $\boldsymbol{Z}_l := \boldsymbol{Z}$ as a variable for training sets with l elements. The j fold of a n-fold cross-validation is denoted by $\boldsymbol{Z}^j$ or $\boldsymbol{Z}^j_{l/n}$ to include the number l/n of elements of the fold. The

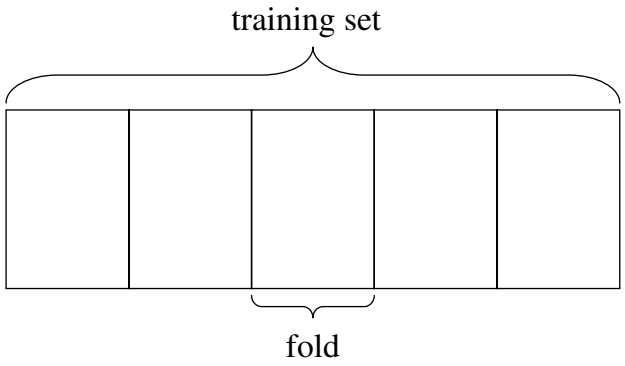


Figure 3.1: Cross-validation: The data set is divided into 5 parts for 5-fold cross-validation — each part is called fold.

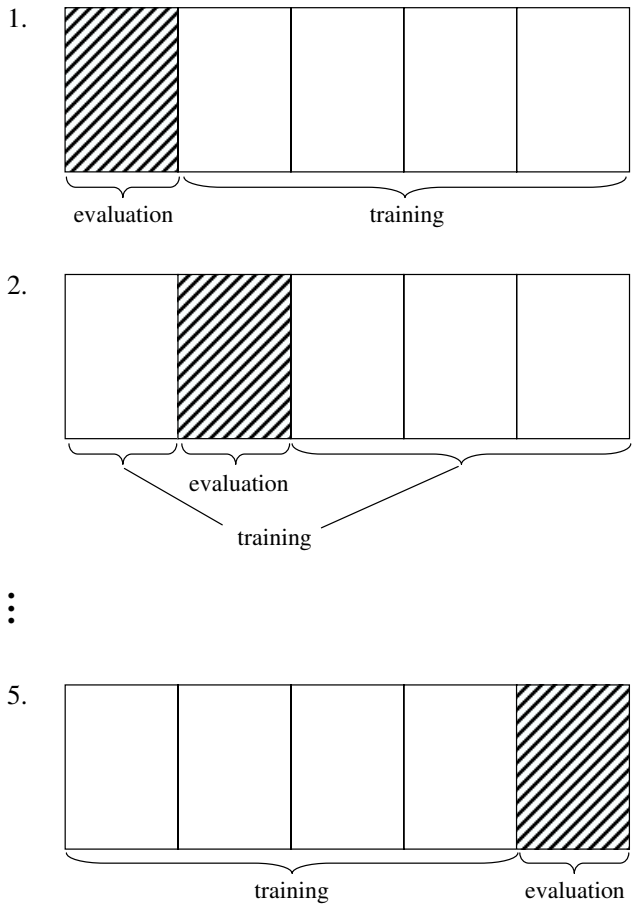


Figure 3.2: Cross-validation: For 5-fold cross-validation there are 5 iterations and in each iteration a different fold is omitted and the remaining folds form the training set. After training the model is tested on the omitted fold. The cumulative error on all folds is the cross-validation error.

n-fold cross-validation risk is

$$R_{n-\mathrm{cv}}(\boldsymbol{Z}_l) \;=\; \frac{1}{n}\sum_{j=1}^{n}\frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}_{l/n}^{j}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}_j\left(\boldsymbol{Z}_l\setminus\boldsymbol{Z}_{l/n}^{j}\right)\right)\right)\right)\;, \tag{3.14}$$

where $\boldsymbol{w}_j$ is the model selected when removing the jth fold and

$$R_{n-\mathrm{cv},j}(\boldsymbol{Z}_l) \;=\; \frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}_{l/n}^{j}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}_j\left(\boldsymbol{Z}_l\setminus\boldsymbol{Z}_{l/n}^{j}\right)\right)\right)\right) \tag{3.15}$$

is the risk for the jth fold.

The statement that the "cross-validation estimate for the risk is almost unbiased" (Luntz and Brailovsky) means

$$\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\left(R\left(g\left(.;\boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;=\; \mathrm{E}_{\boldsymbol{Z}_l}\left(R_{n-\mathrm{cv}}\left(\boldsymbol{Z}_l\right)\right)\;. \tag{3.16}$$

The generalization error on training size l without one fold l/n, namely $l-l/n=l(1-1/n)$ can be estimated by cross-validation on training data of size l by n-fold cross-validation. For large l the training size l or $l(1-1/n)$ should lead similar results, that is the estimate is almost unbiased.

The following two equations will prove eq. (3.16).

The left hand side of eq. (3.16) can be rewritten as

$$\begin{aligned}
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\left(R\left(g\left(.;\boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}\cup\boldsymbol{z}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\mathrm{E}_{\boldsymbol{Z}_{l/n}}\left(\frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}_{l/n}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right)\right)\;.
\end{aligned} \tag{3.17}$$

The second equations stems from the fact that the data points are iid, therefore $\mathrm{E}_{\boldsymbol{z}}\left(f(\boldsymbol{z})\right) \;=\; \frac{1}{k}\sum_{i=1}^{k}\mathrm{E}_{\boldsymbol{z}}\left(f(\boldsymbol{z}^i)\right) \;=\; \mathrm{E}_{\boldsymbol{Z}_k}\left(\frac{1}{k}\sum_{i=1}^{k}f(\boldsymbol{z}^i)\right)$.

The right hand side of eq. (3.16) can be rewritten as

$$\begin{aligned}
&\mathrm{E}_{\boldsymbol{Z}_l}\left(R_{n-\mathrm{cv}}\left(\boldsymbol{Z}_l\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_l}\left(\frac{1}{n}\sum_{j=1}^{n}\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}_{l/n}^{j}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}_j\left(\boldsymbol{Z}_l\setminus\boldsymbol{Z}_{l/n}^{j}\right)\right)\right)\right)\right) \;= \\
&\frac{1}{n}\sum_{j=1}^{n}\mathrm{E}_{\boldsymbol{Z}_l}\left(\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}_{l/n}^{j}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}_j\left(\boldsymbol{Z}_l\setminus\boldsymbol{Z}_{l/n}^{j}\right)\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\mathrm{E}_{\boldsymbol{Z}_{l/n}}\left(\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}_{l/n}}\left(L\left(y,g\left(\boldsymbol{x};\boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right)\right)\;.
\end{aligned} \tag{3.18}$$

The first equality comes from the fact that sum and integral as interchangeable. Therefore it does not matter whether first the data is drawn and then the different folds are treated or the data is drawn again for treating each fold. The second equality comes from the fact that $\mathrm{E}\left(\boldsymbol{Z}_{l/n}^{j}\right) = \mathrm{E}\left(\boldsymbol{Z}_{l/n}\right)$.

Therefore both sides of eq. (3.16) are equal.

The term "almost" addresses the fact that the estimation is made with $l(1-1/n)$ training data using the risk and with l training data using n-fold cross-validation.

However the cross-validation estimate has high variance. The high variance stems from the fact that the training data is overlapping. Also test and training data are overlapping. Intuitively speaking, if data points are drawn which make the task very complicated, then these data points appear in many training sets and at least in one test set. These data points strongly increase the estimate of the risk. The opposite is true for data points which make learning more easy. That means single data points may strongly influence the estimate of the risk.

3.3 Minimal Risk for a Gaussian Classification Task

We will show an example for the optimal risk for a classification task.

We assume that we have a binary classification task where class $y = 1$ data points come from a Gaussian and class $y = -1$ data points come from a different Gaussian.

Class $y = 1$ data points are drawn according to

$$p(\boldsymbol{x} \mid y = 1) \propto \mathcal{N}\left(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) \tag{3.19}$$

and Class $y = -1$ according to

$$p(\boldsymbol{x} \mid y = -1) \propto \mathcal{N}\left(\boldsymbol{\mu}_{-1}, \boldsymbol{\Sigma}_{-1}\right) \tag{3.20}$$

where the Gaussian $\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ has density

$$p(\boldsymbol{x}) = \frac{1}{(2\,\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) . \tag{3.21}$$

$\boldsymbol{x}$ is the d-dimensional feature vector, $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the $d \times d$-dimensional covariance matrix.

As depicted in Fig. 3.3, the linear transformation $\boldsymbol{A}$ leads to the Gaussian $\mathcal{N}\left(\boldsymbol{A}^T\boldsymbol{\mu}, \boldsymbol{A}^T\boldsymbol{\Sigma}\,\boldsymbol{A}\right)$. All projections $\boldsymbol{P}$ of a Gaussian are Gaussian. A certain transformation $A_w = \boldsymbol{\Sigma}^{-1/2}$ ("whitening") leads to a Gaussian with the identity matrix as covariance matrix. On the other hand each Gaussian with covariance matrix $\boldsymbol{\Sigma}$ can be obtained from a Gaussian with covariance matrix $\boldsymbol{I}$ by the linear transformation $\boldsymbol{\Sigma}^{1/2}$.

Affine transformation (translation and linear transformation) allow to interpret all Gaussians as stemming from a Gaussian with zero mean and the identity as covariance matrix.

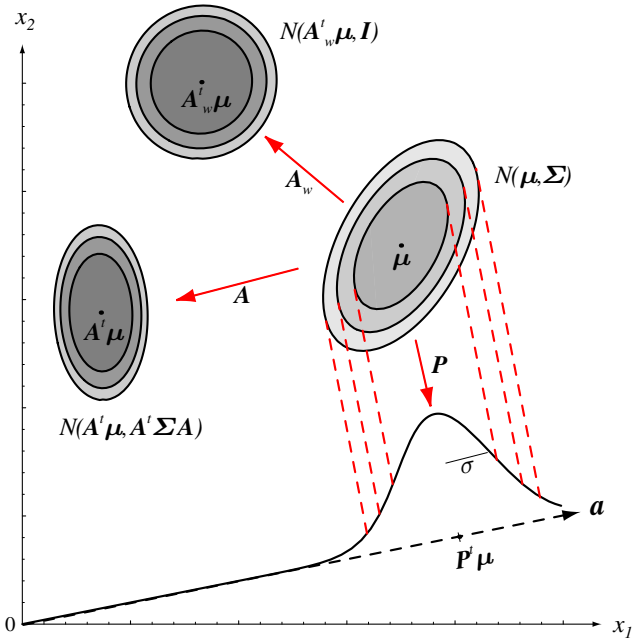


Figure 3.3: Linear transformations of the Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The linear transformation $\boldsymbol{A}$ leads to the Gaussian $\mathcal{N}\left(\boldsymbol{A}^T\boldsymbol{\mu}, \boldsymbol{A}^T\boldsymbol{\Sigma}\,\boldsymbol{A}\right)$. All projections $\boldsymbol{P}$ of a Gaussian are Gaussian. A certain transformation A_w ("whitening") leads to a Gaussian with the identity matrix as covariance matrix.

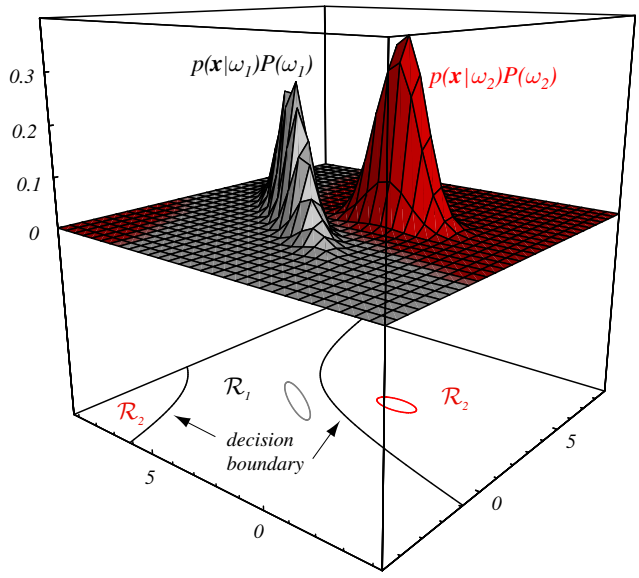


Figure 3.4: A two-dimensional classification task where the data for each class are drawn from a Gaussian (black: class 1, red: class -1). The optimal decision boundaries are two hyperbolas. Here $\omega_1 \equiv y = 1$ and $\omega_2 \equiv y = -1$. In the gray regions $p(y = 1 \mid \boldsymbol{x}) > p(y = -1 \mid \boldsymbol{x})$ holds and in the red regions the opposite holds. Copyright © 2001 John Wiley & Sons, Inc.

At a certain point $\boldsymbol{x}$ in the feature space the probability $p(\boldsymbol{x}, y = 1)$ of observing a point from class $y = 1$ is the probability $p(y = 1)$ of choosing class $y = 1$ multiplied by the Gaussian density for class $y = 1$

$$p(\boldsymbol{x}, y = 1) \;=\; p(\boldsymbol{x} \mid y = 1)\, p(y = 1)\;. \tag{3.22}$$

Fig. 3.4 shows a two-dimensional classification task where the data for each class are drawn from a Gaussian (black: class 1, red: class -1). The discriminant functions are are two hyperbolas forming the optimal decision boundaries.

The probability of observing a point at $\boldsymbol{x}$ not depending on the class is

$$p(\boldsymbol{x}) \;=\; p(\boldsymbol{x}, y = 1) \;+\; p(\boldsymbol{x}, y = -1)\;. \tag{3.23}$$

Here the variable y is "integrated out".

The probability of observing a point from class $y = 1$ at $\boldsymbol{x}$ is

$$p(y = 1 \mid \boldsymbol{x}) \;=\; \frac{p(\boldsymbol{x} \mid y = 1)\, p(y = 1)}{p(\boldsymbol{x})}\;. \tag{3.24}$$

This formula is obtained by applying the Bayes rule.

We define the regions of class 1 as

$$X_1 \;=\; \{\boldsymbol{x} \mid g(\boldsymbol{x}) > 0\} \tag{3.25}$$

and regions of class -1 as

$$X_{-1} \;=\; \{\boldsymbol{x} \mid g(\boldsymbol{x}) < 0\}\;. \tag{3.26}$$

and the loss function as

$$L(y, g(\boldsymbol{x};\boldsymbol{w})) \;=\; \left\{ \begin{array}{lll} 0 & \text{for} & y\; g(\boldsymbol{x};\boldsymbol{w}) > 0 \\ 1 & \text{for} & y\; g(\boldsymbol{x};\boldsymbol{w}) < 0 \end{array} \right. \;. \tag{3.27}$$

The risk of eq. (3.3) is for the zero-one loss

$$\begin{aligned} R(g(.;\boldsymbol{w})) \;&=\; \int_{X_1} p\,(\boldsymbol{x}, y=-1)\; d\boldsymbol{x} \;+\; \int_{X_{-1}} p\,(\boldsymbol{x}, y=1)\; d\boldsymbol{x} \;= \\ &\int_{X_1} p\,(y=-1 \mid \boldsymbol{x})\; p(\boldsymbol{x})\; d\boldsymbol{x} \;+\; \int_{X_{-1}} p\,(y=1 \mid \boldsymbol{x})\; p(\boldsymbol{x})\; d\boldsymbol{x} \;= \\ &\int_X \left\{ \begin{array}{lll} p\,(y=-1 \mid \boldsymbol{x}) & \text{for} & g(\boldsymbol{x}) > 0 \\ p\,(y=1 \mid \boldsymbol{x}) & \text{for} & g(\boldsymbol{x}) < 0 \end{array} \right\} p(\boldsymbol{x})\; d\boldsymbol{x}\;. \end{aligned} \tag{3.28}$$

In the last equation it obvious how the risk can be minimized by choosing the smaller value of $p\,(y=-1 \mid \boldsymbol{x})$ and $p\,(y=1 \mid \boldsymbol{x})$. Therefore, the risk is minimal if

$$g(\boldsymbol{x};\boldsymbol{w}) \left\{ \begin{array}{lll} > 0 & \text{for} & p\,(y=1 \mid \boldsymbol{x}) \;>\; p\,(y=-1 \mid \boldsymbol{x}) \\ < 0 & \text{for} & p\,(y=-1 \mid \boldsymbol{x}) \;>\; p\,(y=1 \mid \boldsymbol{x}) \end{array} \right. \;. \tag{3.29}$$

The minimal risk is

$$\begin{aligned} R_{\min} \;&=\; \int_X \min\{p\,(\boldsymbol{x}, y=-1)\,, p\,(\boldsymbol{x}, y=1)\}\; d\boldsymbol{x} \;= \\ &\int_X \min\{p\,(y=-1 \mid \boldsymbol{x})\,, p\,(y=1 \mid \boldsymbol{x})\}\; p(\boldsymbol{x})\; d\boldsymbol{x}\;. \end{aligned} \tag{3.30}$$

Because at each point either class $y=1$ or class $y=-1$ will be misclassified we classify the point a belonging to the class with higher probability. This is demonstrated in Fig. 3.5 where at each position $\boldsymbol{x}$ either the red or the black line determines the probability of misclassification. The ratio of misclassification is given by integrating along the curve according to eq. (3.28). The minimal integration value is obtained if one chooses the lower curve as misclassification probability that is classifying the point as belonging to the class of the upper curve.

For a linear classifier there is in one dimension only a point x, the *decision point*, where values larger than x are classified to one class and values smaller than x are classified into the other class. The optimal decision point minimizes the misclassification rate. Fig. 3.6 shows such an example.

We call function g a *discriminant function* if it has a positive value if at $\boldsymbol{x}$ point are classified as belonging to the positive class and vice versa. Such functions are also called *classification functions*. The class estimation $\hat{y}(\boldsymbol{x})$ ($\hat{}$ indicates estimation), i.e. the classifier is

$$\hat{y}(\boldsymbol{x}) \;=\; \text{sign} g(\boldsymbol{x})\;. \tag{3.31}$$

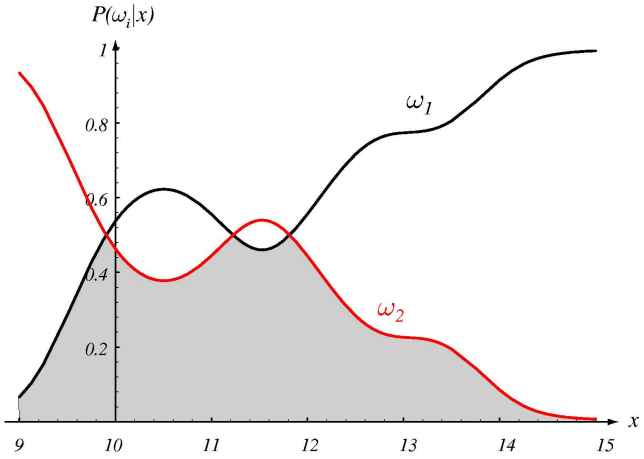


Figure 3.5: Posterior densities $p(y = 1 \mid \boldsymbol{x})$ and $p(y = -1 \mid \boldsymbol{x})$ as a function of $\boldsymbol{x}$. If using the optimal discriminant function the gray region is the integral eq. (3.28) and gives the probability misclassifying a data point. Modified figure with copyright © 2001 John Wiley & Sons, Inc.

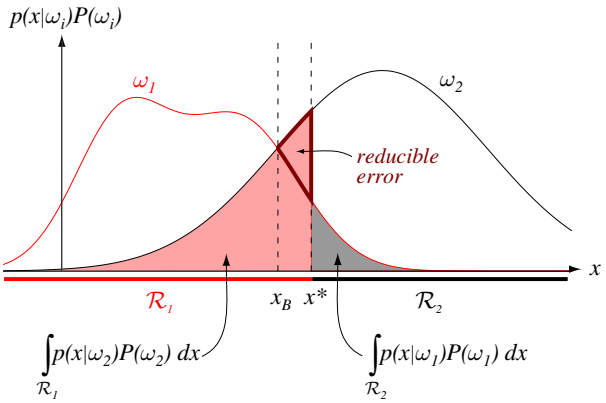


Figure 3.6: x^* is a non-optimal decision point because for some regions the posterior $y = 1$ is above the posterior $y = -1$ but data is classified as $y = -1$. The misclassification rate is given by the filled region. However the misclassification mass in the red triangle can be saved if using as decision point x_B. Copyright © 2001 John Wiley & Sons, Inc.

A discriminant function which minimizes the future risk is

$$g(\boldsymbol{x}) \;=\; p(y=1 \mid \boldsymbol{x}) \;-\; p(y=-1 \mid \boldsymbol{x}) \;= \tag{3.32}$$
$$\frac{1}{p(\boldsymbol{x})} \left(p(\boldsymbol{x} \mid y=1)\, p(y=1) \;-\; p(\boldsymbol{x} \mid y=-1)\, p(y=-1) \right) ,$$

where only the difference in the last brackets matters because $p(\boldsymbol{x}) > 0$. Note, that the optimal discriminant function is not unique because the difference of strict monotone mappings of $p(y = 1 \mid \boldsymbol{x})$ and $p(y = -1 \mid \boldsymbol{x})$ keep the sign of discriminant function and lead to the same classification rule.

Using this fact we take the logarithm to obtain a more convenient discriminant function which also minimized the future risk:

$$g(\boldsymbol{x}) \;=\; \ln p(y=1 \mid \boldsymbol{x}) \;-\; \ln p(y=-1 \mid \boldsymbol{x}) \;= \tag{3.33}$$
$$\ln \frac{p(\boldsymbol{x} \mid y=1)}{p(\boldsymbol{x} \mid y=-1)} \;+\; \ln \frac{p(y=1)}{p(y=-1)} \,.$$

For our Gaussian case we obtain

$$g(\boldsymbol{x}) \;=\; -\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\, \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) \;-\; \frac{d}{2}\ln 2\pi \;- \tag{3.34}$$
$$\frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \ln p(y=1) \;+$$
$$\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_{-1})^T\, \boldsymbol{\Sigma}_{-1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{-1}) \;+\; \frac{d}{2}\ln 2\pi \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;-\; \ln p(y=-1) \;=$$
$$-\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\, \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) \;-\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \ln p(y=1) \;+$$
$$\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_{-1})^T\, \boldsymbol{\Sigma}_{-1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{-1}) \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;-\; \ln p(y=-1) \;=$$
$$-\frac{1}{2}\boldsymbol{x}^T \left(\boldsymbol{\Sigma}_1^{-1} \;-\; \boldsymbol{\Sigma}_{-1}^{-1}\right) \boldsymbol{x} \;+\; \boldsymbol{x}^T \left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 \;-\; \boldsymbol{\Sigma}_{-1}^{-1}\boldsymbol{\mu}_{-1}\right) \;-$$
$$\frac{1}{2}\,\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\,\boldsymbol{\mu}_{-1}^T\boldsymbol{\Sigma}_{-1}^{-1}\boldsymbol{\mu}_{-1} \;-\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;+$$
$$\ln p(y=1) \;-\; \ln p(y=-1) \;=$$
$$-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \;+\; \boldsymbol{w}^T\boldsymbol{x} \;+\; b\,.$$

The function $g(\boldsymbol{x}) \;=\; 0$ defines the class boundaries which are hyper-quadrics (hyper-ellipses or hyper-hyperbolas).

If the covariance matrices of both classes are equal, $\boldsymbol{\Sigma}_1 \;=\; \boldsymbol{\Sigma}_{-1} \;=\; \boldsymbol{\Sigma}$, then the discriminant function is

$$g(\boldsymbol{x}) \;=\; \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\,(\boldsymbol{\mu}_1 \;-\; \boldsymbol{\mu}_{-1}) \;+ \tag{3.35}$$
$$\boldsymbol{\mu}_{-1}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_{-1} \;-\; \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 \;+\; \ln p(y=1) \;-\; \ln p(y=-1) \;=$$
$$\boldsymbol{w}^T\boldsymbol{x} \;+\; b\,.$$

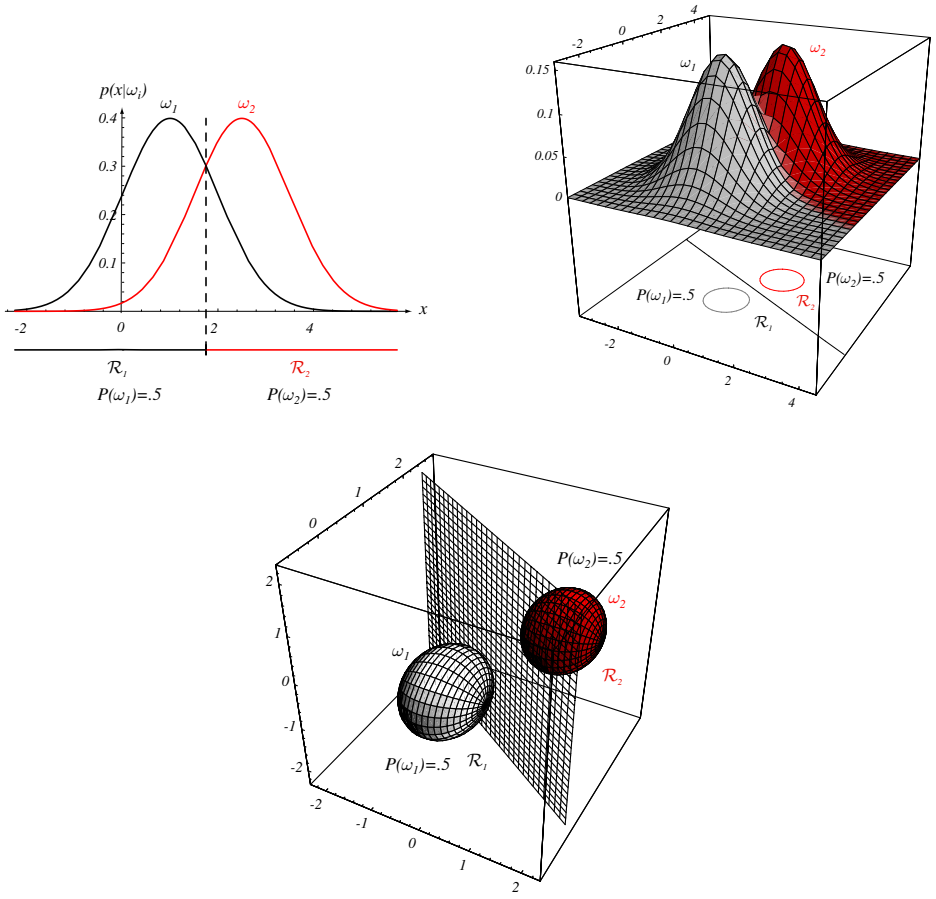


Figure 3.7: Two classes with covariance matrix $\mathbf{\Sigma} = \sigma^2 \boldsymbol{I}$ each in one (top left), two (top right), and three (bottom) dimensions. The optimal boundary is a hyperplane. Copyright © 2001 John Wiley & Sons, Inc.

The boundary function $g(\boldsymbol{x}) = 0$ is a hyperplane in the d-dimensional space. See examples for $d = 1$, $d = 2$, and $d = 3$ in Fig. 3.7.

For the general case, where $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_{-1}$ the boundary functions can be hyperplanes, hyper-ellipsoids, hyper-paraboloids etc. Examples for the 2-dim. case are given in Fig. 3.8.

3.4 Maximum Likelihood

So far we only considered the supervised task, where we have a label y which should be predicted correctly for future data. We were able to define the loss via the distance between the predicted value and the true value.

In unsupervised task defining a loss function and a risk is not as straight forward as in supervised learning.

3.4.1 Loss for Unsupervised Learning

3.4.1.1 Projection Methods

Unsupervised task include projection of the data into another space in order to fulfill desired requirements. Fig. 3.9 depicts a projection model.

For example with "Principal Component Analysis" (PCA) the data is projected into a lower dimensional space. Here a trade-off between losing information and low dimensionality appears. It is difficult to define a loss function which takes both the dimension and the information loss into account.

For example with "Independent Component Analysis" (ICA) the data is projected into a space where the components of the data vectors are mutually independent. As loss function may serve the minimal difference of the actual distribution of the projected data to a factorial distribution (a distribution where the components are independent from each other).

Often only characteristics of a factorial distribution are optimized such as entropy (factorial distribution has maximal entropy under constant variance), cummulants (some are maximal and some are zero for factorial distributions), and other.

For some cases a prototype factorial distribution (e.g. the product of super-Gaussians) is used and the projected data should be aligned to this prototype distribution as good as possible.

For example with "Projection Pursuit" the components have to be maximally non-Gaussian. Here the ideas as for ICA hold, too.

3.4.1.2 Generative Model

One of the most common unsupervised task is to build a *generative model* that is a model which simulates the world and produces the same data as the world. Fig. 3.10 depicts a generative model.

The data generation process of the world is assumed to be probabilistic. Therefore, the observed data of the world are only random examples and we do not want to exactly reproduce the

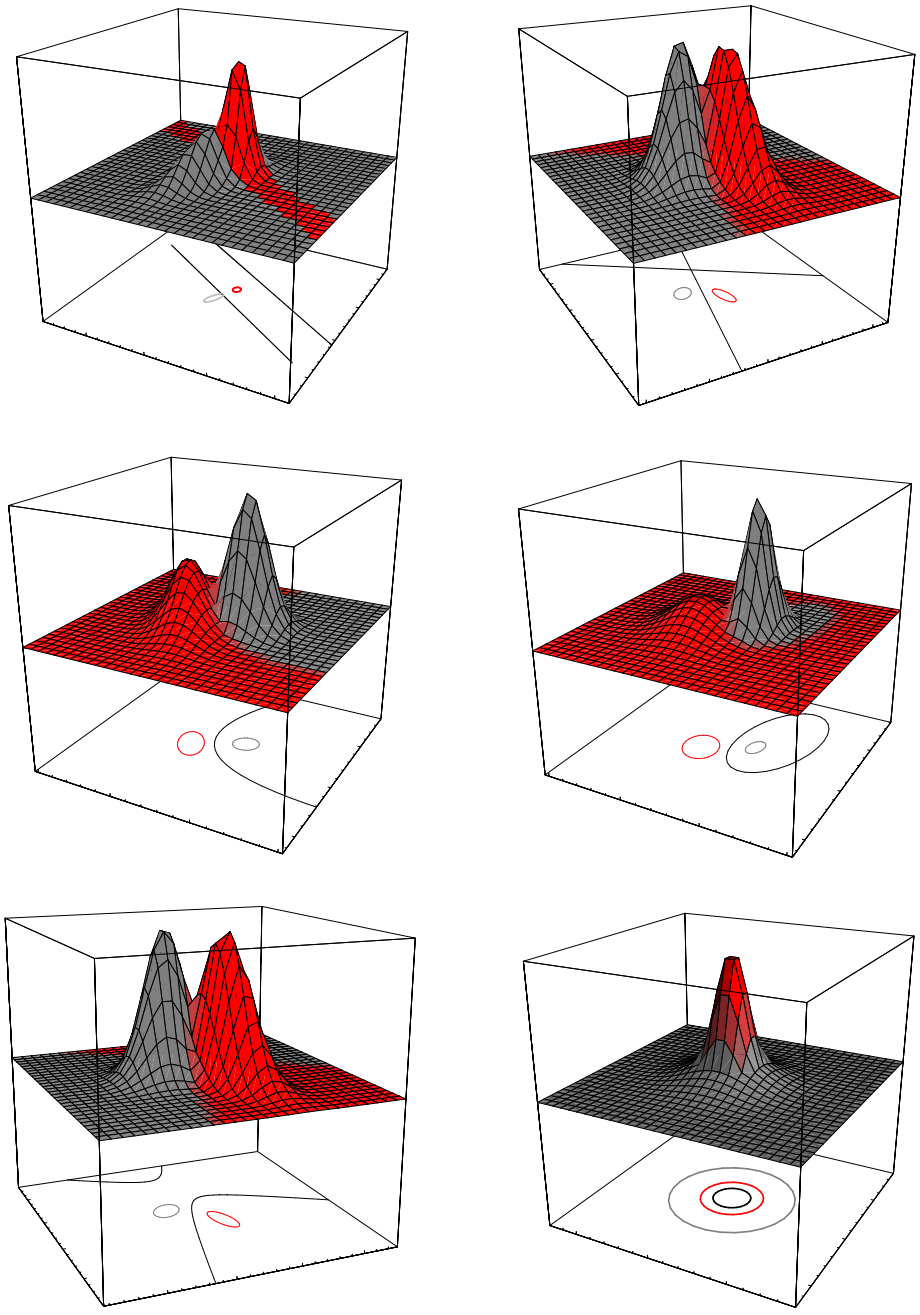


Figure 3.8: Two classes with arbitrary Gaussian covariance lead to boundary functions which are hyperplanes, hyper-ellipsoids, hyperparaboloids etc.

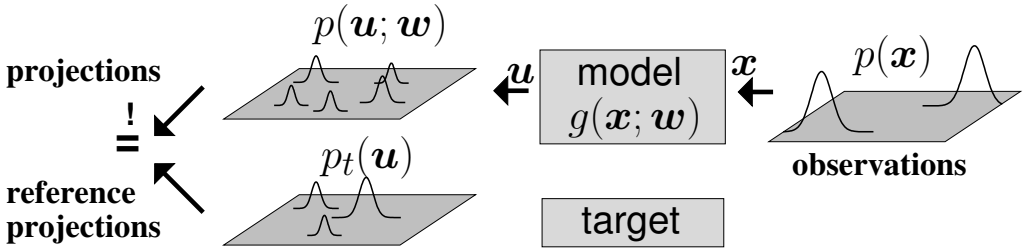


Figure 3.9: Projection model, where the observed data $\boldsymbol{x}$ is the input to the model $\boldsymbol{u} = g(\boldsymbol{x};\boldsymbol{w})$. The model output distribution should match a target distribution or the output distribution should fulfill some constraints. The later can be replaced by the distribution which fulfills the constraints and is closest to the target distribution.

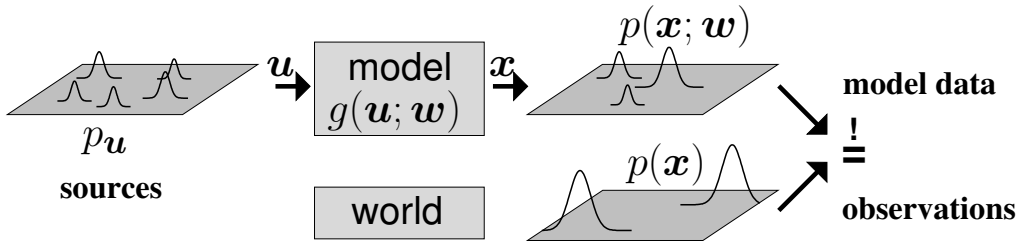


Figure 3.10: Generative model, where the data $\boldsymbol{x}$ is observed and the model $\boldsymbol{x} = g(\boldsymbol{u};\boldsymbol{w})$ should produce the same distribution as the observed distribution. The vectors $\boldsymbol{u}$ are random vectors supplied to the model in order to drive it, i.e. to make it a data generation process.

observed data (that can be done by storing the data in data bases). However, we assume that the data generation process produces the data according to some distribution.

The generative model approach attempt at approximation the distribution of the data generation process as good as possible. As loss function the distance between the distribution of the data generation process and the model output distribution is suitable.

Generative models include “Factor Analysis”, “Latent Variable Models”, and Boltzmann Machines.

3.4.1.3 Parameter Estimation

In another approach to unsupervised learning we assume that we know the model which produces the data but the model is parameterized and the actual parameters are not known. The task of the unsupervised learning method is to estimate the actual parameters.

Here the loss would be the difference between the actual (true) parameter and the estimated parameter. However we have no future data points. But we can evaluate the estimator through the expected difference, where the expectation is made over the training set.

In the next sections we will focus on parameter estimation and will evaluate estimation methods based on expectation over the training set.

3.4.2 Mean Squared Error, Bias, and Variance

In unsupervised tasks the *training data* is $\{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\}$, where $\boldsymbol{z}^i = \boldsymbol{x}^i$ and, therefore, it is $\{\boldsymbol{x}\} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\}$ for which we will often simply write $\boldsymbol{X}$ (the matrix of training data).

The true parameter vector is denoted by $\boldsymbol{w}$ and its estimate by $\hat{\boldsymbol{w}}$.

An estimator is *unbiased* if

$$\mathrm{E}_{\boldsymbol{X}}\hat{\boldsymbol{w}} = \boldsymbol{w}\,, \tag{3.36}$$

i.e. on the average the estimator will yield the true parameter.

The *bias* is

$$b(\hat{\boldsymbol{w}}) = \mathrm{E}_{\boldsymbol{X}}\hat{\boldsymbol{w}} - \boldsymbol{w}\,. \tag{3.37}$$

The *variance* of the estimator is defined as

$$\mathrm{var}(\hat{\boldsymbol{w}}) = \mathrm{E}_{\boldsymbol{X}}\left((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T(\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))\right)\,. \tag{3.38}$$

An evaluation criterion for unsupervised methods is the *mean squared error* (MSE) in the text after eq. (3.2). The MSE in eq. (3.2) was defined as an expectation over future data points.

Here we define the MSE as expectation over the training set, because we deal with unsupervised learning and evaluate the estimator. The MSE gives the expected error as squared distance between the estimated parameter and the true parameter.

$$\mathrm{mse}(\hat{\boldsymbol{w}}) = \mathrm{E}_{\boldsymbol{X}}\left((\hat{\boldsymbol{w}} - \boldsymbol{w})^T(\hat{\boldsymbol{w}} - \boldsymbol{w})\right)\,. \tag{3.39}$$

We can reformulate the MSE:

$$\begin{aligned}
&\mathrm{mse}(\hat{\boldsymbol{w}}) = \mathrm{E}_{\boldsymbol{X}}\left((\hat{\boldsymbol{w}} - \boldsymbol{w})^T(\hat{\boldsymbol{w}} - \boldsymbol{w})\right) = \\
&\mathrm{E}_{\boldsymbol{X}}\Big(((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}})) + (\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w}))^T \\
&((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}})) + (\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w}))\Big) = \\
&\mathrm{E}_{\boldsymbol{X}}\Big((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T(\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}})) - \\
&2\,(\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w}) + \\
&(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w})^T(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w})\Big) = \\
&\mathrm{E}_{\boldsymbol{X}}\left((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T(\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))\right) + \\
&(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w})^T(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w}) = \\
&\mathrm{var}(\hat{\boldsymbol{w}}) + b^2(\hat{\boldsymbol{w}})\,.
\end{aligned} \tag{3.40}$$

where the last but one equality comes from the fact that only $\hat{\boldsymbol{w}}$ depends on $\boldsymbol{X}$ and therefore

$$\begin{aligned} &\mathrm{E}_{\boldsymbol{X}}\left((\hat{\boldsymbol{w}} - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T (\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w})\right) = \\ &(\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}))^T (\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}}) - \boldsymbol{w}) = 0 . \end{aligned} \tag{3.41}$$

The MSE is decomposed into a variance term $\mathrm{var}(\hat{\boldsymbol{w}})$ and a bias term $b^2(\hat{\boldsymbol{w}})$. The variance has high impact on the performance because large deviations from the true parameter have strong influence on the MSE through the quadratic error term.

Note that averaging linearly reduces the variance. The average is

$$\hat{\boldsymbol{w}_{a_N}} = \frac{1}{N}\sum_{p=1}^{N} \hat{\boldsymbol{w}}_i , \tag{3.42}$$

where

$$\begin{aligned} \hat{\boldsymbol{w}}_i &= \hat{\boldsymbol{w}}_i(\boldsymbol{X}_i) \\ \boldsymbol{X}_i &= \left\{\boldsymbol{x}^{(i-1)\ l/N\ +\ 1}, \ldots, \boldsymbol{x}^{i\ l/N}\right\} , \end{aligned} \tag{3.43}$$

i.e. $\boldsymbol{X}_i$ is the i-th subset of $\boldsymbol{X}$ and contains l/N elements. The size of the data is l and the examples of $\boldsymbol{X}_i$ range from $(i-1)\ l/N\ +\ 1$ to $i\ l/N$.

The average is unbiased:

$$\mathrm{E}_{\boldsymbol{X}}(\hat{\boldsymbol{w}_{a_N}}) = \frac{1}{N}\sum_{p=1}^{N} \mathrm{E}_{\boldsymbol{X}_i}\hat{\boldsymbol{w}}_i = \frac{1}{N}\sum_{p=1}^{N} \boldsymbol{w} = \boldsymbol{w} . \tag{3.44}$$

The variance is linearly reduced

$$\begin{aligned} &\mathrm{covar}_{\boldsymbol{X}}(\hat{\boldsymbol{w}_{a_N}}) = \frac{1}{N^2}\sum_{p=1}^{N} \mathrm{covar}_{\boldsymbol{X}_i}(\hat{\boldsymbol{w}}_i) = \\ &\frac{1}{N^2}\sum_{p=1}^{N} \mathrm{covar}_{\boldsymbol{X},l/N}(\hat{\boldsymbol{w}}) = \frac{1}{N}\mathrm{covar}_{\boldsymbol{X},l/N}(\hat{\boldsymbol{w}}) , \end{aligned} \tag{3.45}$$

where $\mathrm{covar}_{\boldsymbol{X},l/N}(\hat{\boldsymbol{w}})$ is the estimator with l/N training points.

We used the facts:

$$\begin{aligned} &\mathrm{covar}_{\boldsymbol{X}}(\boldsymbol{a} + \boldsymbol{b}) = \\ &\mathrm{covar}_{\boldsymbol{X}}(\boldsymbol{a}) + \mathrm{covar}_{\boldsymbol{X}}(\boldsymbol{b}) + \mathrm{var}_{\boldsymbol{X}}(\boldsymbol{a}^T\boldsymbol{b}) + \mathrm{var}_{\boldsymbol{X}}(\boldsymbol{b}^T\boldsymbol{a}) \\ &\mathrm{covar}_{\boldsymbol{X}}(\lambda\,\boldsymbol{a}) = \lambda^2\,\mathrm{covar}_{\boldsymbol{X}}(\boldsymbol{a}) . \end{aligned} \tag{3.46}$$

For averaging it is important that the training sets $\boldsymbol{X}_i$ are independent from each other and do not overlap. Otherwise the estimators are dependent and the covariance terms between the estimators do not vanish.

One approach to find an optimal estimator is to construct from all unbiased estimators the one with minimal variance, which is called Minimal Variance Unbiased (MVU) estimator.

A MVU estimator does not always exist. However there are methods to check whether a given estimator is a MVU estimator.

3.4.3 Fisher Information Matrix, Cramer-Rao Lower Bound, and Efficiency

In the following we will define a lower bound, the *Cramer-Rao Lower Bound* for the variance of an unbiased estimator. That induces a lower bound on the MSE of an estimator.

We need the *Fisher information matrix* $\boldsymbol{I}_F$ to define this lower bound. The Fisher information matrix $\boldsymbol{I}_F$ for a parameterized model

$$\boldsymbol{I}_F(\boldsymbol{w}) : \ [\boldsymbol{I}_F(\boldsymbol{w})]_{ij} \ = \ - \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})} \left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_i} \ \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_j} \right) , \tag{3.47}$$

and $[\boldsymbol{A}]_{ij} \ = \ A_{ij}$ selects the ijth element of a matrix and

$$\begin{aligned} &\mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})} \left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_i} \ \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_j} \right) \ = \\ &\int \ \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_i} \ \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial w_j} \ p(\boldsymbol{x};\boldsymbol{w}) \ d\boldsymbol{x} \ . \end{aligned} \tag{3.48}$$

If the density function $p(\boldsymbol{x};\boldsymbol{w})$ satisfies

$$\forall_{\boldsymbol{w}} : \ \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})} \left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}} \right) \ = \ \boldsymbol{0} \tag{3.49}$$

then the Fisher information matrix is

$$\boldsymbol{I}_F(\boldsymbol{w}) : \ \boldsymbol{I}_F(\boldsymbol{w}) \ = \ - \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})} \left(\frac{\partial^2 \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w} \ \partial \boldsymbol{w}} \right) \ . \tag{3.50}$$

The last equation follows from the fact that

$$0 \;=\; \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}} d\boldsymbol{x}\right) \;= \tag{3.51}$$

$$\int_X \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\, p(\boldsymbol{x};\boldsymbol{w})\; d\boldsymbol{x}$$

$$\Longrightarrow \quad \frac{\partial}{\partial \boldsymbol{w}} \int_X \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\, p(\boldsymbol{x};\boldsymbol{w})\; d\boldsymbol{x} \;=\; \mathbf{0} \tag{3.52}$$

$$\Longrightarrow \quad \int_X \left(\frac{\partial^2 \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}\, \partial \boldsymbol{w}}\, p(\boldsymbol{x};\boldsymbol{w}) + \right. \tag{3.53}$$

$$\left. \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}} \frac{\partial p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\right)\; d\boldsymbol{x} \;=\; \mathbf{0}$$

$$\Longrightarrow \quad \int_X \left(\frac{\partial^2 \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}\, \partial \boldsymbol{w}}\, p(\boldsymbol{x};\boldsymbol{w}) + \right. \tag{3.54}$$

$$\left. \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\; \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\, p(\boldsymbol{x};\boldsymbol{w})\right)\; d\boldsymbol{x} \;=\; \mathbf{0}$$

$$\Longrightarrow \quad -\,\mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\frac{\partial^2 \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}\, \partial \boldsymbol{w}}\right) \;= \tag{3.55}$$

$$\mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\; \frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\right)\,.$$

The Fisher information gives the amount of information that an observable random variable $\boldsymbol{x}$ carries about an unobservable parameter $\boldsymbol{w}$ upon which the parameterized density function $p(\boldsymbol{x};\boldsymbol{w})$ of $\boldsymbol{x}$ depends.

Theorem 3.1 (Cramer-Rao Lower Bound (CRLB))
Assume that

$$\forall_{\boldsymbol{w}}: \;\; \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}\right) \;=\; \mathbf{0} \tag{3.56}$$

and that the estimator $\hat{\boldsymbol{w}}$ is unbiased.

Then,

$$\mathrm{covar}(\hat{\boldsymbol{w}}) \;-\; \boldsymbol{I}_F^{-1}(\boldsymbol{w}) \tag{3.57}$$

is positive definite:

$$\mathrm{covar}(\hat{\boldsymbol{w}}) \;-\; \boldsymbol{I}_F^{-1}(\boldsymbol{w}) \;\geq\; \mathbf{0}\,. \tag{3.58}$$

An unbiased estimator attains the bound in that $\mathrm{covar}(\hat{\boldsymbol{w}}) \;=\; \boldsymbol{I}_F^{-1}(\boldsymbol{w})$ *if and only if*

$$\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}} \;=\; \boldsymbol{A}(\boldsymbol{w})\;(\boldsymbol{g}(\boldsymbol{x}) \;-\; \boldsymbol{w}) \tag{3.59}$$

for some function $\boldsymbol{g}$ and square matrix $\boldsymbol{A}(\boldsymbol{w})$. In this case the MVU estimator is

$$\hat{\boldsymbol{w}} \;=\; \boldsymbol{g}(\boldsymbol{x}) \quad \textit{with} \;\; \mathrm{covar}(\hat{\boldsymbol{w}}) \;=\; \boldsymbol{A}^{-1}(\boldsymbol{w})\,. \tag{3.60}$$

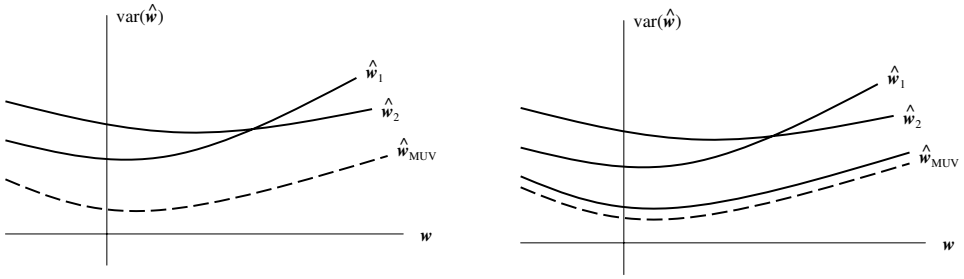


Figure 3.11: The variance of an estimator $\hat{\boldsymbol{w}}$ as a curve of the true parameter is shown. We show three estimators: $\hat{\boldsymbol{w}}_1$, $\hat{\boldsymbol{w}}_2$, and the minimal variance unbiased $\hat{\boldsymbol{w}}_{\mathrm{MVU}}$. Left: The MVU estimator $\hat{\boldsymbol{w}}_{\mathrm{MVU}}$ is efficient because it reaches the CRLB. Right: The MVU estimator $\hat{\boldsymbol{w}}_{\mathrm{MVU}}$ does not reach the CRLB an is not efficient.

Note that

$$\left[\mathrm{covar}(\hat{\boldsymbol{w}}) \; - \; \boldsymbol{I}_F^{-1}(\boldsymbol{w})\right]_{ii} \; \geq \; 0 \; , \tag{3.61}$$

therefore

$$\mathrm{var}(\hat{w}_i) \; = \; [\mathrm{covar}(\hat{\boldsymbol{w}})]_{ii} \; \geq \; \left[\boldsymbol{I}_F^{-1}(\boldsymbol{w})\right]_{ii} \; . \tag{3.62}$$

An estimator is said to be *efficient* if it reaches the CRLB. It is efficient in that it efficiently makes use of the data and extracts information to estimate the parameter.

A MVU estimator can be efficient but it also can not be as depicted in Fig. 3.11.

3.4.4 Maximum Likelihood Estimator

In many cases of unsupervised learning as parameter estimation the MVU estimator is unknown or does not exist.

A very popular estimator is the Maximum Likelihood Estimator (MLE) on which almost all practical estimation tasks are based. The popularity stems from the fact that it can be applied to a broad range of problems and it approximates the MVU estimator for large data sets. The MLE is even asymptotically efficient and unbiased. That means the MLE does everything right and this efficiently if enough data is available.

The likelihood $\mathcal{L}$ of the data set $\{\boldsymbol{x}\} \; = \; \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\}$ is

$$\mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}) \; = \; p(\{\boldsymbol{x}\}; \boldsymbol{w}) \; , \tag{3.63}$$

i.e. the probability of the model $p(\boldsymbol{x}; \boldsymbol{w})$ to produce the data set. However the set $\{\boldsymbol{x}\}$ has zero measure and therefore the density at the data set $\{\boldsymbol{x}\}$ must be used.

For iid data sampling the likelihood is

$$\mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}) \; = \; p(\{\boldsymbol{x}\}; \boldsymbol{w}) \; = \; \prod_{i=1}^{l} p(\boldsymbol{x}^i; \boldsymbol{w}) \; . \tag{3.64}$$

Instead of maximizing the likelihood $\mathcal{L}$ the log-likelihood $\ln \mathcal{L}$ or the negative log-likelihood $-\ln \mathcal{L}$ is minimized. The logarithm transforms the product of the iid data sampling into a sum:

$$-\ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}) \;=\; -\sum_{i=1}^{l} \ln p(\boldsymbol{x}^i; \boldsymbol{w}) \,. \tag{3.65}$$

To motivate the use of the density in the likelihood one can assume that if $p(\boldsymbol{x}^i; \boldsymbol{w})$ is written actually $p(\boldsymbol{x}^i; \boldsymbol{w})\; d\boldsymbol{x}$ is meant, which gives the probability of observing $\boldsymbol{x}$ in a region of volume $d\boldsymbol{x}$ around $\boldsymbol{x}^i$. In this case the likelihood gives the probability of the model to produce similar data points as $\{\boldsymbol{x}\}$, where similar means data points in a volume $d\boldsymbol{x}$ around the actual observed data points.

However the fact that the MLE is so popular is based on its simple use and its properties (given in the next section) especially that it is optimal for the number of training points going to infinity.

3.4.5 Properties of Maximum Likelihood Estimator

In the next subsections different properties of the MLE are given. First, MLE is invariant under parameter change. Then, most importantly, the MLE is asymptotically unbiased and efficient, i.e. asymptotically optimal. Finally, the MLE is consistent for zero CRLB.

3.4.5.1 MLE is Invariant under Parameter Change

Theorem 3.2 (Parameter Change Invariance)
Let g be a function changing the parameter $\boldsymbol{w}$ into parameter $\boldsymbol{u}$: $\boldsymbol{u} \;=\; g(\boldsymbol{w})$, then

$$\hat{\boldsymbol{u}} \;=\; g(\hat{\boldsymbol{w}}) \,, \tag{3.66}$$

where the estimators are MLE. If g changes $\boldsymbol{w}$ into different $\boldsymbol{u}$ then $\hat{\boldsymbol{u}} \;=\; g(\hat{\boldsymbol{w}})$ maximizes the likelihood function

$$\max_{\boldsymbol{w}:\boldsymbol{u}=g(\boldsymbol{w})} p(\{\boldsymbol{x}\}; \boldsymbol{w}) \,. \tag{3.67}$$

This theorem is important because for some models parameter changes simplify the expressions for the densities.

3.4.5.2 MLE is Asymptotically Unbiased and Efficient

Note, that an estimator $\hat{\boldsymbol{w}} \;=\; \hat{\boldsymbol{w}}(\boldsymbol{X})$ changes its properties with the size l of the training set $\boldsymbol{X}$. For example for reasonable estimator the variance should decrease with increasing l.

The maximum likelihood estimator is *asymptotically unbiased*

$$\mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\hat{\boldsymbol{w}}\right) \;\overset{l\to\infty}{\to}\; \boldsymbol{w} \tag{3.68}$$

and it is *asymptotically efficient*

$$\operatorname{covar}(\hat{\boldsymbol{w}}) \overset{l\to\infty}{\to} \operatorname{CRLB} . \tag{3.69}$$

These properties are derived from the following theorem

Theorem 3.3 (MLE Asymptotic Properties)
If $p(\boldsymbol{x};\boldsymbol{w})$ *satisfies*

$$\forall_{\boldsymbol{w}}: \ \mathrm{E}_{p(\boldsymbol{x};\boldsymbol{w})}\left(\frac{\partial \ln p(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}} d\boldsymbol{x}\right) = \mathbf{0} \tag{3.70}$$

then the MLE which maximizes $p(\{\boldsymbol{x}\};\boldsymbol{w})$ *is asymptotically distributed according to*

$$\hat{\boldsymbol{w}} \overset{l\to\infty}{\propto} \mathcal{N}\left(\boldsymbol{w}, \boldsymbol{I}_F^{-1}(\boldsymbol{w})\right) , \tag{3.71}$$

where $\boldsymbol{I}_F(\boldsymbol{w})$ *is the Fisher information matrix evaluated at the unknown parameter* $\boldsymbol{w}$*.*

This quite general theorem is the basis of the asymptotic optimal properties of the MLE.

However for practical applications the number l is finite and the performance of the MLE is not known.

For example consider the general linear model

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{\epsilon} , \tag{3.72}$$

where $\boldsymbol{\epsilon} \propto \mathcal{N}(\mathbf{0}, \boldsymbol{C})$ is an additive Gaussian noise vector.

Then the MLE is

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{A}^T\boldsymbol{C}^{-1}\boldsymbol{A}\right)^{-1} A^T\boldsymbol{C}^{-1}\boldsymbol{x} . \tag{3.73}$$

which is also efficient and, therefore, MVU. The density of $\hat{\boldsymbol{w}}$ is

$$\hat{\boldsymbol{w}} \propto \mathcal{N}\left(\boldsymbol{w}, \left(\boldsymbol{A}^T\boldsymbol{C}^{-1}\boldsymbol{A}\right)^{-1}\right) . \tag{3.74}$$

Note for factor analysis which will be considered later also $\boldsymbol{C}$ has to be estimated.

3.4.5.3 MLE is Consistent for Zero CRLB

A estimator is said to be *consistent* if

$$\hat{\boldsymbol{w}} \overset{l\to\infty}{\to} \boldsymbol{w} , \tag{3.75}$$

i.e. for large training sets the estimator approaches the true value.

Later – in the empirical risk minimization treatment by V. Vapnik in 3.6.2 – we need a more formal definition for consistency as

$$\lim_{l\to\infty} p\left(|\hat{\boldsymbol{w}} - \boldsymbol{w}| > \epsilon\right) = 0 . \tag{3.76}$$

The MLE is consistent if the CRLB is zero. This follows directly from the fact that MLE is asymptotically unbiased and efficient, i.e. the variance will approach zero.

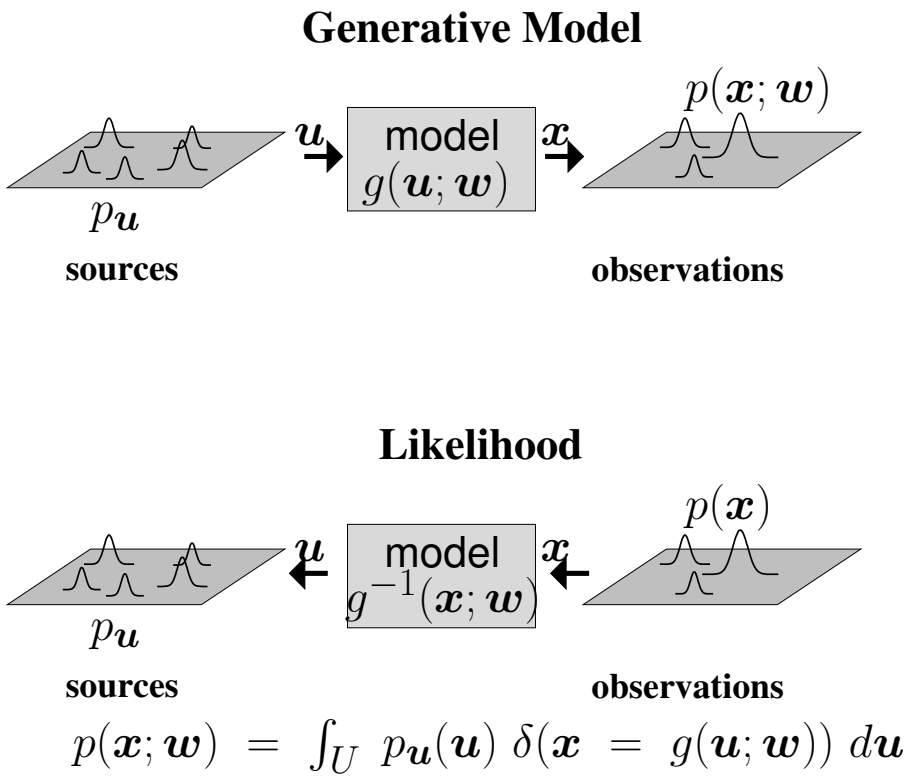


Figure 3.12: The maximum likelihood problem. Top: the generative model which produces data points. Bottom: in order to compute the likelihood for $\boldsymbol{x}$ all points $\boldsymbol{u}$ which are mapped to $\boldsymbol{x}$ must be determined and then multiplied with the probability $p_{\boldsymbol{u}}(\boldsymbol{u})$ that $\boldsymbol{u}$ is observed in the model.

3.4.6 Expectation Maximization

The likelihood can be optimized by gradient descent methods as will be described in Chapter 5. However the likelihood must be expressed analytically to obtain the derivatives. For some models the likelihood cannot be computed analytically because of hidden states of the model, of a many-to-one output mapping of the model, or of non-linearities. As depicted in Fig. 3.12, to compute the likelihood the inverse of a function – more precise, all elements $\boldsymbol{u}$ which are mapped to a certain observed point $\boldsymbol{x}$ – must be computed in order to obtain the likelihood that the point is generated by the model. If g is highly nonlinear, then the integral which determines the likelihood is difficult to compute analytically. To guess the likelihood numerically is difficult as the density of the model output at a certain point in space must be estimated.

The variables $\boldsymbol{u}$ in Fig. 3.12 can be interpreted as unobserved variables, i.e. *hidden variables* or *latent variables*. For models with hidden variables the likelihood is determined by all possible values of the hidden variables which can produce output $\boldsymbol{x}$.

For many models the joint probability $p(\{\boldsymbol{x}\}, \boldsymbol{u}; \boldsymbol{w})$ of the hidden variables $\boldsymbol{u}$ and observations $\{\boldsymbol{x}\}$ is easier to compute than the likelihood of the observations. If we can also estimate $p(\boldsymbol{u} \mid \{\boldsymbol{x}\}; \boldsymbol{w})$ the hidden variables $\boldsymbol{u}$ using the parameters $\boldsymbol{w}$ and given the observations $\{\boldsymbol{x}\}$ then we can apply the Expectation Maximization (EM) algorithm.

Let us assume we have an estimation $Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})$ for $p(\boldsymbol{u} \mid \{\boldsymbol{x}\}; \boldsymbol{w})$, which is some density

with respect to $\boldsymbol{u}$. The following inequality is the basic for the EM algorithm:

$$\begin{aligned}
&\ln \mathcal{L}(\{\boldsymbol{x}\};\boldsymbol{w}) \;=\; \ln p(\{\boldsymbol{x}\};\boldsymbol{w}) \;= \\
&\ln \int_U p(\{\boldsymbol{x}\},\boldsymbol{u};\boldsymbol{w})\, d\boldsymbol{u} \;= \\
&\ln \int_U \frac{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})}{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})} p(\{\boldsymbol{x}\},\boldsymbol{u};\boldsymbol{w})\, d\boldsymbol{u} \;\geq \\
&\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln \frac{p(\{\boldsymbol{x}\},\boldsymbol{u};\boldsymbol{w})}{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})}\, d\boldsymbol{u} \;= \\
&\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln p(\{\boldsymbol{x}\},\boldsymbol{u};\boldsymbol{w})\, d\boldsymbol{u} \;- \\
&\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})\, d\boldsymbol{u} \;= \\
&\mathcal{F}(Q,\boldsymbol{w})\,.
\end{aligned} \tag{3.77}$$

where the "$\geq$" is the application of Jensen's inequality. Jensen's inequality states that the value of a convex function of an integral is smaller or equal to to the integral of the convex function applied to the integrand. Therefore a convex function of an expectation is smaller or equal to to the expectation of the convex function. Here the expectation with respect to $Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})$ is used and the fact that $-\ln$ is a convex function.

Above inequality states that $\mathcal{F}(Q,\boldsymbol{w})$ is an lower bound to the log-likelihood $\ln \mathcal{L}(\{\boldsymbol{x}\};\boldsymbol{w})$.

The EM algorithm is an iteration between two steps, the "E"-step and the "M"-step:

$$\begin{aligned}
&\textbf{E-step:} \\
&Q_{k+1} \;=\; \arg\max_Q \; \mathcal{F}(Q,\boldsymbol{w}_k) \\
&\textbf{M-step:} \\
&\boldsymbol{w}_{k+1} \;=\; \arg\max_{\boldsymbol{w}} \; \mathcal{F}(Q_{k+1},\boldsymbol{w})\,.
\end{aligned} \tag{3.78}$$

It is important to note that in the E-step the maximal Q is

$$\begin{aligned}
Q_{k+1}(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;&=\; p(\boldsymbol{u} \mid \{\boldsymbol{x}\};\boldsymbol{w}_k) \\
\mathcal{F}(Q_{k+1},\boldsymbol{w}_k) \;&=\; \ln \mathcal{L}(\{\boldsymbol{x}\};\boldsymbol{w}_k)\,.
\end{aligned} \tag{3.79}$$

To see the last statements:

$$p(\boldsymbol{u},\{\boldsymbol{x}\};\boldsymbol{w}_k) \;=\; p(\boldsymbol{u} \mid \{\boldsymbol{x}\};\boldsymbol{w}_k)\, p(\{\boldsymbol{x}\};\boldsymbol{w}_k)\,, \tag{3.80}$$

therefore

$$\begin{aligned}
&\mathcal{F}(Q,\boldsymbol{w}) \;=\; \int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln \frac{p(\{\boldsymbol{x}\},\boldsymbol{u};\boldsymbol{w})}{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})}\, d\boldsymbol{u} \;= \\
&\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln \frac{p(\boldsymbol{u} \mid \{\boldsymbol{x}\};\boldsymbol{w})}{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})}\, d\boldsymbol{u} \;+\; \ln p(\{\boldsymbol{x}\};\boldsymbol{w}) \;= \\
&-\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;\ln \frac{Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})}{p(\boldsymbol{u} \mid \{\boldsymbol{x}\};\boldsymbol{w})}\, d\boldsymbol{u} \;+\; \ln \mathcal{L}(\{\boldsymbol{x}\};\boldsymbol{w})
\end{aligned} \tag{3.81}$$

The expression $\int_U Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})\ \ln \frac{Q(\boldsymbol{u}|\{\boldsymbol{x}\})}{p(\boldsymbol{u}|\{\boldsymbol{x}\};\boldsymbol{w})}\ d\boldsymbol{u}$ is the Kullback-Leibler divergence $D_{\mathrm{KL}}(Q \parallel p)$ between $Q(\boldsymbol{u} \mid \{\boldsymbol{x}\})$ and $p(\boldsymbol{u} \mid \{\boldsymbol{x}\}; \boldsymbol{w})$. The Kullback-Leibler divergence $\mathrm{KL}(p_1, p_2)$ is defined as

$$D_{\mathrm{KL}}(p_1 \parallel p_2) \;=\; \int_U p_1(\boldsymbol{u})\ \ln \frac{p_1(\boldsymbol{u})}{p_2(\boldsymbol{u})}\ d\boldsymbol{u} \tag{3.82}$$

and the *cross entropy* as

$$-\ \int_U p_1(\boldsymbol{u})\ \ln p_2(\boldsymbol{u})\ d\boldsymbol{u}\ . \tag{3.83}$$

Kullback-Leibler divergence is always larger than zero:

$$D_{\mathrm{KL}}(p_1 \parallel p_2) \;\geq\; 0 \tag{3.84}$$

because

$$\begin{aligned} &0 \;=\; \ln 1 \;=\; \ln \int_U p_2(\boldsymbol{u})\ d\boldsymbol{u} \;= \\ &\ln \int_U p_1(\boldsymbol{u})\ \frac{p_2(\boldsymbol{u})}{p_1(\boldsymbol{u})}\ d\boldsymbol{u} \;\geq \\ &\int_U p_1(\boldsymbol{u})\ \ln \frac{p_2(\boldsymbol{u})}{p_1(\boldsymbol{u})}\ d\boldsymbol{u} \;=\; -\ D_{\mathrm{KL}}(p_1 \parallel p_2)\ . \end{aligned} \tag{3.85}$$

Thus, if $D_{\mathrm{KL}}(Q \parallel p) \;=\; 0$ then $\mathcal{F}(Q, \boldsymbol{w}_k)$ is maximized because the Kullback-Leibler divergence, which enters the equation with a negative sign, is minimal. We have $Q(\boldsymbol{u} \mid \{\boldsymbol{x}\}) \;=\; p(\boldsymbol{u} \mid \{\boldsymbol{x}\}; \boldsymbol{w})$ and obtain

$$\mathcal{F}(Q, \boldsymbol{w}) \;=\; \ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w})\ . \tag{3.86}$$

In the M-step only the expression $\int_U Q_{k+1}(\boldsymbol{u} \mid \{\boldsymbol{x}\})\ \ln p(\{\boldsymbol{x}\}, \boldsymbol{u}; \boldsymbol{w})\ d\boldsymbol{u}$ must be considered because the other term (the entropy of Q_{k+1}) is independent of the parameters $\boldsymbol{w}$.

The EM algorithm can be interpreted as:

- E-step: Tighten the lower bound to equality: $\mathcal{F}(Q, \boldsymbol{w}) \;=\; \ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w})$.
- M-step: Maximize the lower bound which is at the equality and therefore increase the likelihood. This might lead to a lower bound which is no longer tight.

The EM algorithm increases the lower bound because in both steps the lower bound is maximized.

Can it happen that maximizing the lower bound may decrease the likelihood? No! At the beginning of the M-step we have $\mathcal{F}(Q_{k+1}, \boldsymbol{w}_k) \;=\; \ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}_k)$, and the E-step does not change the parameters $\boldsymbol{w}$:

$$\begin{aligned} &\ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}_k) \;=\; \mathcal{F}(Q_{k+1}, \boldsymbol{w}_k) \;\leq \\ &\mathcal{F}(Q_{k+1}, \boldsymbol{w}_{k+1}) \;\leq\; \mathcal{F}(Q_{k+2}, \boldsymbol{w}_{k+1}) \;=\; \ln \mathcal{L}(\{\boldsymbol{x}\}; \boldsymbol{w}_{k+1})\ , \end{aligned} \tag{3.87}$$

where the first "$\leq$" is from the M-step which gives w_{k+1} and the second "$\leq$" from the E-step which gives Q_{k+2}.

The EM algorithms will later be derived for hidden Markov models, mixture of Gaussians, factor analysis, independent component analysis, etc.

3.5 Noise Models

In this section we will make a connection between unsupervised and supervised learning in terms of the quality measure. Towards this end we introduce additional noise on the targets, that means we do not know the exact values of the targets. If we know the noise distribution then we can look for the most likely target. Therefore we can apply maximum likelihood to supervised learning.

3.5.1 Gaussian Noise

We consider the case of Gaussian target noise and a simple linear model:

$$\boldsymbol{s} = \boldsymbol{X}\,\boldsymbol{w} \tag{3.88}$$

$$\boldsymbol{y} = \boldsymbol{s} + \boldsymbol{\epsilon} = \boldsymbol{X}\,\boldsymbol{w} + \boldsymbol{\epsilon}\,, \tag{3.89}$$

where $\boldsymbol{s}$ is the true signal, $\boldsymbol{X}$ is the observed data, $\boldsymbol{w}$ is the parameter vector, y is the observed target, and $\boldsymbol{\epsilon}$ is the Gaussian noise vector with zero mean and covariance $\boldsymbol{\Sigma}$. Note, that the covariance $\boldsymbol{\Sigma}$ is the noise distribution for each measurement or observation $\boldsymbol{x}$.

The value $\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w}$ is distributed according to the Gaussian, therefore the likelihood of $(\boldsymbol{y}, \boldsymbol{X})$ is

$$\mathcal{L}((\boldsymbol{y}, \boldsymbol{X}); \boldsymbol{w}) = \frac{1}{(2\,\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}}\,\exp\left(-\frac{1}{2}\,(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})\right)\,. \tag{3.90}$$

The log-likelihood is

$$\ln \mathcal{L}((\boldsymbol{y}, \boldsymbol{X}); \boldsymbol{w}) = -\frac{d}{2}\ln(2\,\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\,(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})\,. \tag{3.91}$$

To maximize the log-likelihood we have to minimize

$$(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})\,, \tag{3.92}$$

because other terms are independent of $\boldsymbol{w}$.

The minimum of this term, called *least square criterion*, is the *linear least square* estimator.

Multiplying out the criterion gives

$$\begin{aligned}&(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{X}\,\boldsymbol{w}) = \\ &\boldsymbol{y}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\,\boldsymbol{X}\,\boldsymbol{w}\end{aligned} \tag{3.93}$$

and the derivative with respect to $\boldsymbol{w}$ is

$$-\,2\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{y} + 2\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\,\boldsymbol{X}\,\boldsymbol{w}\,, \tag{3.94}$$

which are called the Wiener-Hopf equations (correlation between features $\boldsymbol{X}$ and target $\boldsymbol{y}$ should be equal to the correlation between features $\boldsymbol{X}$ and model prediction $\boldsymbol{X}\,\boldsymbol{w}$). Setting the derivative to zero gives the least square estimator

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\,\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{y} \tag{3.95}$$

The minimal least square criterion is

$$\boldsymbol{y}^T\left(\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\boldsymbol{X}\left(\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\,\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{\Sigma}^{-1}\right)\,\boldsymbol{y}\,. \tag{3.96}$$

In many cases the noise covariance matrix is

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{\sigma}\,\boldsymbol{I}\,, \tag{3.97}$$

which means that for each observation we have the same noise assumption.

We obtain

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}\,, \tag{3.98}$$

where $\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T$ is known as the pseudo inverse or Moore-Penrose inverse of $\boldsymbol{X}$. The minimal value is

$$\frac{1}{\sigma}\,\boldsymbol{y}^T\left(\boldsymbol{I} - \boldsymbol{X}\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\right)\,\boldsymbol{y}\,. \tag{3.99}$$

Note that we can derive the squared error criterion also for other models $\boldsymbol{g}(\boldsymbol{X};\boldsymbol{w})$ instead of the linear model $\boldsymbol{X}\boldsymbol{w}$. However, in general the estimator must be selected by using an optimization technique which minimizes the least square criterion

$$(\boldsymbol{y} - \boldsymbol{g}(\boldsymbol{X};\boldsymbol{w}))^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{g}(\boldsymbol{X};\boldsymbol{w}))\,. \tag{3.100}$$

These considerations are the basis for the least square fit and also for the mean squared error as a risk function. These approaches to loss and risk are derived from maximum likelihood and Gaussian noise assumptions. Therefore the mean squared error in 3.2 can be justified by Gaussian noise assumption.

3.5.2 Laplace Noise and Minkowski Error

Even if the Gaussian noise assumption is the most widely used noise model, other noise models may be more adequate for certain problems.

For example the loss function

$$\|\boldsymbol{y} \, - \, \boldsymbol{g}(\boldsymbol{X}; \boldsymbol{w})\|_1 \tag{3.101}$$

corresponds to Laplace noise assumption. Or for one dimension

$$|\boldsymbol{y} \, - \, \boldsymbol{g}(\boldsymbol{x}; \boldsymbol{w})| \ . \tag{3.102}$$

For one dimensional output we obtain for the Laplacian noise model

$$p(y \, - \, g(\boldsymbol{x}; \boldsymbol{w})) \, = \, \frac{\beta}{2} \exp\left(- \, \beta \, |y \, - \, g(\boldsymbol{x}; \boldsymbol{w})|\right) \tag{3.103}$$

with loss function

$$|y \, - \, g(\boldsymbol{x}; \boldsymbol{w})| \, . \tag{3.104}$$

For the Minkowski error

$$|y \, - \, g(\boldsymbol{x}; \boldsymbol{w})|^r \tag{3.105}$$

the corresponding noise model is

$$p(y \, - \, g(\boldsymbol{x}; \boldsymbol{w})) \, = \, \frac{r \, \beta^{1/r}}{2 \, \Gamma(1/r)} \exp\left(- \, \beta \, |y \, - \, g(\boldsymbol{x}; \boldsymbol{w})|^r\right) \ , \tag{3.106}$$

where Γ is the gamma function

$$\begin{aligned} \Gamma(a) \, &= \, \int_0^\infty \, u^{a-1} \, e^{-u} \, du \\ \Gamma(n) \, &= \, (n-1)! \, . \end{aligned} \tag{3.107}$$

For $r < 2$ large errors are down-weighted compared to the quadratic error and vice versa for $r > 2$. That means for data with outliers $r < 2$ may be an appropriate choice for the noise model. See examples of error functions in Fig. 3.13.

If random large fluctuations of the output are possible then $l < 2$ should be used in order to give these fluctuations more probability in the noise model and down-weight their influence onto the estimator.

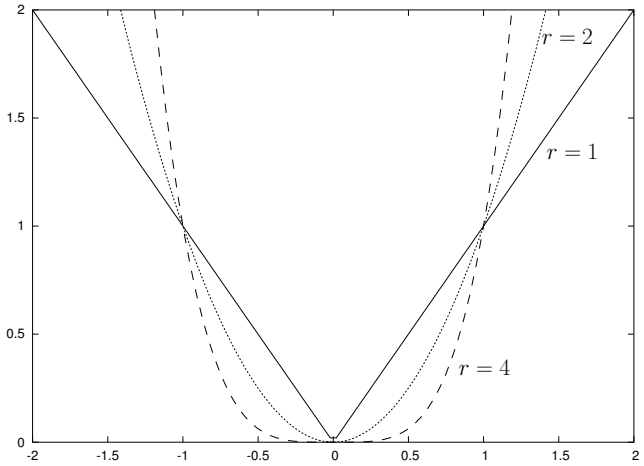


Figure 3.13: Different noise assumptions lead to different Minkowski error functions: $r = 1$ (Laplace noise), $r = 2$ (Gaussian noise), and $r = 4$.

3.5.3 Binary Models

Above noise considerations do not hold for binary y as used for classification. If the class label is disturbed then y is assigned to the opposite class.

Also for binary y the likelihood approach can be applied.

3.5.3.1 Cross-Entropy

For a classification problem with K classes, we assume that the model output is a probability:

$$g_k(\boldsymbol{x}; \boldsymbol{w}) \;=\; p(\boldsymbol{y} = \boldsymbol{e}_k \mid \boldsymbol{x}) \;. \tag{3.108}$$

and that

$$\boldsymbol{y} \;\in\; \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_K\} \;. \tag{3.109}$$

If $\boldsymbol{x}$ is in the k-th class then $\boldsymbol{y} \;=\; (0, \ldots, 0, 1, 0, \ldots, 0)$, where the "1" is at position k in the vector $\boldsymbol{y}$.

The likelihood of iid data is

$$\mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) \;=\; p(\{\boldsymbol{z}\}; \boldsymbol{w}) \;=\; \prod_{i=1}^{l} \prod_{k=1}^{K} p(\boldsymbol{y}^i \;= \boldsymbol{e}_k \mid \boldsymbol{x}^i; \boldsymbol{w})^{\left[\boldsymbol{y}^i\right]_k} p(\boldsymbol{x}^i) \tag{3.110}$$

because

$$\prod_{k=1}^{K} p(\boldsymbol{y}^i \;= \boldsymbol{e}_k \mid \boldsymbol{x}^i; \boldsymbol{w})^{\left[\boldsymbol{y}^i\right]_k} \;=\; p(\boldsymbol{y}^i \;= \boldsymbol{e}_r \mid \boldsymbol{x}^i; \boldsymbol{w}) \text{ for } \boldsymbol{y}^i \;=\; \boldsymbol{e}_r \;. \tag{3.111}$$

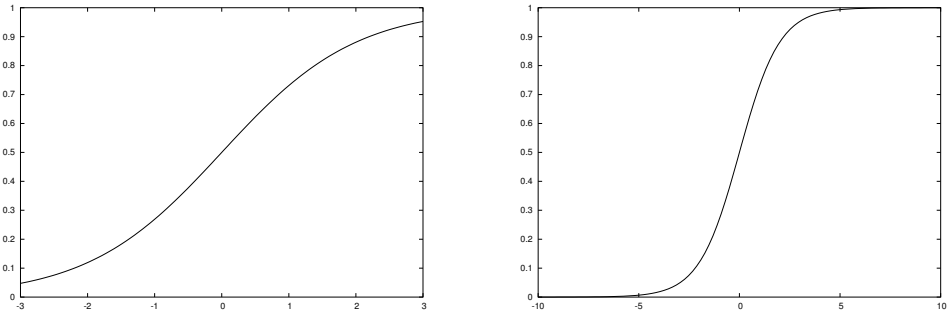


Figure 3.14: The sigmoidal function $\frac{1}{1+\exp(-x)}$.

The log-likelihood is

$$\ln \mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) \;=\; \sum_{k=1}^{K} \sum_{i=1}^{l} \left[\boldsymbol{y}^i\right]_k \ln p(\boldsymbol{y}^i \;= \boldsymbol{e}_k \mid \boldsymbol{x}^i; \boldsymbol{w}) \;+\; \sum_{i=1}^{l} \ln p(\boldsymbol{x}^i)\;. \tag{3.112}$$

Therefore the criterion

$$\sum_{k=1}^{K} \sum_{i=1}^{l} \left[\boldsymbol{y}^i\right]_k \ln p(\boldsymbol{y}^i \;= \boldsymbol{e}_k \mid \boldsymbol{x}^i; \boldsymbol{w}) \tag{3.113}$$

is an natural loss function which is called *cross entropy*. Note that $\left[\boldsymbol{y}^i\right]_k$ is the observed probability $p(\boldsymbol{y}^i \;= \boldsymbol{e}_k)$ which is one if $\boldsymbol{y}^i \;= \boldsymbol{e}_k$ and zero otherwise.

Therefore above formula is indeed the cross entropy as defined in eq. (3.83) for discrete distributions.

3.5.3.2 Logistic Regression

A function g mapping $\boldsymbol{x}$ onto $\mathbb{R}$ can be transformed into a probability by the sigmoidal function

$$\frac{1}{1 \;+\; e^{-\;g(\boldsymbol{x};\boldsymbol{w})}} \tag{3.114}$$

which is depicted in Fig. 3.14.

Note that

$$1 \;-\; \frac{1}{1 \;+\; e^{-\;g(\boldsymbol{x};\boldsymbol{w})}} \;=\; \frac{e^{-\;g(\boldsymbol{x};\boldsymbol{w})}}{1 \;+\; e^{-\;g(\boldsymbol{x};\boldsymbol{w})}}\;. \tag{3.115}$$

We set

$$p(y \;=\; 1 \mid \boldsymbol{x}; \boldsymbol{w}) \;=\; \frac{1}{1 \;+\; e^{-\;g(\boldsymbol{x};\boldsymbol{w})}} \tag{3.116}$$

and

$$p(y \;=\; 0 \mid \boldsymbol{x}; \boldsymbol{w}) \;=\; \frac{e^{-\,g(\boldsymbol{x};\boldsymbol{w})}}{1 \;+\; e^{-\,g(\boldsymbol{x};\boldsymbol{w})}} \;. \tag{3.117}$$

We obtain

$$g(\boldsymbol{x}; \boldsymbol{w}) \;=\; \ln\left(\frac{p(y \;=\; 1 \mid \boldsymbol{x})}{1 \;-\; p(y \;=\; 1 \mid \boldsymbol{x})}\right) \;. \tag{3.118}$$

According to eq. (3.65) the log-likelihood is

$$\ln \mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) \;=\; \sum_{i=1}^{l} \ln p(\boldsymbol{z}_i; \boldsymbol{w}) \;=\; \sum_{i=1}^{l} \ln p(y^i, \boldsymbol{x}^i; \boldsymbol{w}) \;= \tag{3.119}$$
$$\sum_{i=1}^{l} \ln p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) \;+\; \sum_{i=1}^{l} \ln p(\boldsymbol{x}^i) \;.$$

Therefore maximum likelihood maximizes

$$\sum_{i=1}^{l} \ln p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) \tag{3.120}$$

Next we will consider the derivative of the log-likelihood. First we will need some algebraic properties:

$$\frac{\partial}{\partial w_j} \ln p(y = 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \;=\; \frac{\partial}{\partial w_j} \ln \frac{1}{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \;= \tag{3.121}$$
$$\left(1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}\right) \left(-\,\frac{e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{\left(1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}\right)^2}\right) \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \;=$$
$$-\,\frac{e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \;=\; -\,p(y \;=\; 0 \mid \boldsymbol{x}^i; \boldsymbol{w}) \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j}$$

and

$$\frac{\partial}{\partial w_j} \ln p(y = 0 \mid \boldsymbol{x}^i; \boldsymbol{w}) \;=\; \frac{\partial}{\partial w_j} \ln \frac{e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \;= \tag{3.122}$$
$$\frac{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \left(\frac{e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \;-\; \frac{e^{-\,2\,g(\boldsymbol{x}^i;\boldsymbol{w})}}{\left(1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}\right)^2}\right) \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \;=$$
$$\frac{1}{1 \;+\; e^{-\,g(\boldsymbol{x}^i;\boldsymbol{w})}} \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \;=\; p(y \;=\; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \frac{\partial\, g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j}$$

We can rewrite the likelihood as

$$\sum_{i=1}^{l} \ln p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) \; = \tag{3.123}$$
$$\sum_{i=1}^{l} y^i \ln p(y = 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; + \; \sum_{i=1}^{l} (1 \; - \; y^i) \; \ln p(y = 0 \mid \boldsymbol{x}^i; \boldsymbol{w})$$

which gives for the derivative

$$\frac{\partial}{\partial w_j} \sum_{i=1}^{l} \ln p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) \; = \tag{3.124}$$
$$\sum_{i=1}^{l} y^i \frac{\partial}{\partial w_j} \ln p(y = 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; +$$
$$\sum_{i=1}^{l} (1 \; - \; y^i) \; \frac{\partial}{\partial w_j} \ln p(y = 0 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; =$$
$$\sum_{i=1}^{l} - \, y^i \; p(y \; = \; 0 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; \frac{\partial \; g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \; +$$
$$\sum_{i=1}^{l} (1 \; - \; y^i) \; p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; \frac{\partial \; g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \; =$$
$$\sum_{i=1}^{l} \big(- \, y^i \; (1 \; - \; p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}))$$
$$(1 \; - \; y^i) \; p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \big) \; \frac{\partial \; g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \; =$$
$$\sum_{i=1}^{l} \big(p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; - \; y^i \big) \; \frac{\partial \; g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \; ,$$

where

$$p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; = \; \frac{1}{1 \; + \; e^{- \; g(\boldsymbol{x}^i; \boldsymbol{w})}} \tag{3.125}$$

For computing the maximum the derivatives have to be set to zero

$$\forall_j : \; \sum_{i=1}^{l} \big(p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; - \; y^i \big) \; \frac{\partial \; g(\boldsymbol{x}^i; \boldsymbol{w})}{\partial w_j} \; = \; 0 \, . \tag{3.126}$$

Note that the derivatives are products between the prediction error

$$\big(p(y \; = \; 1 \mid \boldsymbol{x}^i; \boldsymbol{w}) \; - \; y^i \big) \tag{3.127}$$

and the derivatives of the function g.

This is very similar to the derivative of the quadratic loss function in the regression case, where we would have
$\left(g(\boldsymbol{x}^i;\boldsymbol{w}) \; - \; y^i\right)$ instead of $\left(p(y \; = \; 1 \mid \boldsymbol{x}^i;\boldsymbol{w}) \; - \; y^i\right)$.

If a neural network h with a sigmoid output unit, the mean squared error as objective function, and the class labels as target is trained, then the derivative is

$$\begin{aligned} \forall_j : \; \sum_{i=1}^{l} \left(h(\boldsymbol{x}^i;\boldsymbol{w}) \; - \; y^i\right) \; \frac{\partial \, h(\boldsymbol{x}^i;\boldsymbol{w})}{\partial w_j} \; = \\ \left(p(y \; = \; 1 \mid \boldsymbol{x}^i;\boldsymbol{w}) \; - \; y^i\right) \; h(\boldsymbol{x}^i;\boldsymbol{w}) \, (1 \; - \; h(\boldsymbol{x}^i;\boldsymbol{w})) \, \frac{\partial \, g(\boldsymbol{x}^i;\boldsymbol{w})}{\partial w_j} \; , \end{aligned} \tag{3.128}$$

where

$$h(\boldsymbol{x}^i;\boldsymbol{w}) \; = \; p(y \; = \; 1 \mid \boldsymbol{x}^i;\boldsymbol{w}) \; = \; \frac{1}{1 \; + \; e^{-\, g(\boldsymbol{x}^i;\boldsymbol{w})}} \tag{3.129}$$

$$\frac{\partial \, h(\boldsymbol{x}^i;\boldsymbol{w})}{\partial g(\boldsymbol{x}^i;\boldsymbol{w})} \; = \; h(\boldsymbol{x}^i;\boldsymbol{w}) \, (1 \; - \; h(\boldsymbol{x}^i;\boldsymbol{w})) \; . \tag{3.130}$$

Therefore the gradient for logistic regression and neural networks differs only in the factor $h(\boldsymbol{x}^i;\boldsymbol{w}) \, (1 \; - \; h(\boldsymbol{x}^i;\boldsymbol{w}))$. The effect of the factor is that the neural network does not push the output towards 1 or 0.

Alternative formulation with $y \in +1, -1$

We now give an alternative formulation of logistic regression with $y \in +1, -1$.

We remember

$$p(y \; = \; 1 \mid \boldsymbol{x};\boldsymbol{w}) \; = \; \frac{1}{1 \; + \; e^{-\, g(\boldsymbol{x};\boldsymbol{w})}} \tag{3.131}$$

and

$$p(y \; = \; -1 \mid \boldsymbol{x};\boldsymbol{w}) \; = \; \frac{e^{-\, g(\boldsymbol{x};\boldsymbol{w})}}{1 \; + \; e^{-\, g(\boldsymbol{x};\boldsymbol{w})}} \; = \; \frac{1}{1 \; + \; e^{\, g(\boldsymbol{x};\boldsymbol{w})}} \; . \tag{3.132}$$

Therefore we have

$$-\ln p(y \; = \; y^i \mid \boldsymbol{x}^i;\boldsymbol{w}) \; = \; \ln \left(1 \; + \; e^{-\, y^i \; g(\boldsymbol{x}^i;\boldsymbol{w})}\right) \tag{3.133}$$

and the objective which minimization maximizes the likelihood is

$$L \; = \; - \; \sum_{i=1}^{l} \ln p(y^i \mid \boldsymbol{x}^i;\boldsymbol{w}) \; = \; - \; \sum_{i=1}^{l} \ln \left(1 \; + \; e^{-\, y^i \; g(\boldsymbol{x}^i;\boldsymbol{w})}\right) \tag{3.134}$$

The derivatives of the objective with respect to the parameters are

$$\frac{\partial L}{\partial w_j} \;=\; -\sum_{i=1}^{l} y^i \,\frac{\partial\, g(\boldsymbol{x}^i;\boldsymbol{w})}{\partial w_j}\,\frac{e^{-\,y^i\; g(\boldsymbol{x}^i;\boldsymbol{w})}}{1\;+\;e^{-\,y^i\; g(\boldsymbol{x}^i;\boldsymbol{w})}} \;= \tag{3.135}$$
$$-\sum_{i=1}^{l} y^i \,\frac{\partial\, g(\boldsymbol{x}^i;\boldsymbol{w})}{\partial w_j}\;\big(1\;-\;p(y^i\mid\boldsymbol{x};\boldsymbol{w})\big)\;.$$

The last equation is similar to eq. (3.124).

In matrix notation we have

$$\frac{\partial L}{\partial \boldsymbol{w}} \;=\; -\sum_{i=1}^{l} y^i\,\big(1\;-\;p(y^i\mid\boldsymbol{x};\boldsymbol{w})\big)\,\frac{\partial\, g(\boldsymbol{x}^i;\boldsymbol{w})}{\partial \boldsymbol{w}}\;. \tag{3.136}$$

3.5.3.3 (Regularized) Linear Logistic Regression is Strictly Convex

Following Jason D. M. Rennie, we show that linear Logistic Regression is strictly convex.

In the linear case we have

$$g(\boldsymbol{x}^i;\boldsymbol{w}) \;=\; \boldsymbol{w}^T\boldsymbol{x}^i\;. \tag{3.137}$$

For labels $y \in +1, -1$ we have

$$\frac{\partial L}{\partial w_j} \;=\; -\sum_{i=1}^{l} y^i x_{ij}\;\big(1\;-\;p(y^i\mid\boldsymbol{x};\boldsymbol{w})\big)\;. \tag{3.138}$$

The second derivatives of the objective L that is minimized are

$$H_{jk} \;=\; \frac{\partial L}{\partial w_j\,\partial w_k} \;=\; \sum_{i=1}^{l} \big(y^i\big)^2\;x_{ij}\;x_{ik}\;p(y^i\mid\boldsymbol{x};\boldsymbol{w})\;\big(1\;-\;p(y^i\mid\boldsymbol{x};\boldsymbol{w})\big)\;, \tag{3.139}$$

where $\boldsymbol{H}$ is the Hessian.

Because $p(1-p) \geq 0$ for $p \leq 1$, we can define

$$\rho_{ij} \;=\; x_{ij}\,\sqrt{p(y^i\mid\boldsymbol{x};\boldsymbol{w})\;(1\;-\;p(y^i\mid\boldsymbol{x};\boldsymbol{w}))}\;. \tag{3.140}$$

The bilinear form of the Hessian with a vector $\boldsymbol{a}$ is

$$\boldsymbol{a}^T \boldsymbol{H} \boldsymbol{a} = \sum_{i=1}^{l} \sum_{j=1}^{d} \sum_{k=1}^{d} x_{ij} x_{ik} a_j a_k p(y^i \mid \boldsymbol{x}; \boldsymbol{w}) \left(1 - p(y^i \mid \boldsymbol{x}; \boldsymbol{w})\right) = \tag{3.141}$$

$$\sum_{i=1}^{l} \sum_{j=1}^{d} a_j x_{ij} \sqrt{p(y^i \mid \boldsymbol{x}; \boldsymbol{w}) \left(1 - p(y^i \mid \boldsymbol{x}; \boldsymbol{w})\right)}$$

$$\sum_{k=1}^{d} a_k x_{ik} \sqrt{p(y^i \mid \boldsymbol{x}; \boldsymbol{w}) \left(1 - p(y^i \mid \boldsymbol{x}; \boldsymbol{w})\right)} =$$

$$\sum_{i=1}^{l} \left(\boldsymbol{a}^T \boldsymbol{\rho}_i\right) \left(\boldsymbol{a}^T \boldsymbol{\rho}_i\right) = \sum_{i=1}^{l} \left(\boldsymbol{a}^T \boldsymbol{\rho}_i\right)^2 \geq 0 .$$

Because we did not make any restriction on $\boldsymbol{a}$, the Hessian is positive definite.

Adding a term like $\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}$ to the objective for regularization, then the Hessian of the objective is strict positive definite.

3.5.3.4 Softmax

For multi-class problems logistic regression can be generalized by Softmax.

We assume K classes with $y \in \{1, \ldots, K\}$ and the probability of $\boldsymbol{x}$ belonging to class k is

$$p(y = k \mid \boldsymbol{x}; g_1, \ldots, g_K, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = \frac{e^{\, g_k(\boldsymbol{x}; \boldsymbol{w}_k)}}{\sum_{j=1}^{K} e^{\, g_j(\boldsymbol{x}; \boldsymbol{w}_j)}} \tag{3.142}$$

which gives a multinomial distribution across the classes.

The objective which is minimized in order to maximize the likelihood is

$$L = -\sum_{i=1}^{l} \ln p(y = y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) = \sum_{i=1}^{l} \ln \left(\sum_{j=1}^{K} e^{g_j(\boldsymbol{x}; \boldsymbol{w}_j)} \right) - g_{y^i}(\boldsymbol{x}; \boldsymbol{w}_{y^i}) . \tag{3.143}$$

In the following we set

$$p(y = k \mid \boldsymbol{x}; g_1, \ldots, g_K, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = p(k \mid \boldsymbol{x}; \boldsymbol{W}) . \tag{3.144}$$

The derivatives are

$$\frac{\partial L}{\partial w_{kn}} = \sum_{i=1}^{l} \frac{\partial \, g_k(\boldsymbol{x}^i; \boldsymbol{w}_k)}{\partial w_{kn}} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) - \delta_{y^i = k} \sum_{i=1}^{l} \frac{\partial \, g_k(\boldsymbol{x}^i; \boldsymbol{w}_k)}{\partial w_{kn}} . \tag{3.145}$$

3.5.3.5 (Regularized) Linear Softmax is Strictly Convex

Following Jason D. M. Rennie, we show that linear Softmax is strictly convex.

In the linear case we have

$$g_k(\boldsymbol{x}^i; \boldsymbol{w}_k) \;=\; \boldsymbol{w}_k^T \boldsymbol{x}^i \tag{3.146}$$

or in vector notation

$$\boldsymbol{g}(\boldsymbol{x}^i; \boldsymbol{W}) \;=\; \boldsymbol{W}^T \boldsymbol{x}^i \;. \tag{3.147}$$

The derivatives are

$$\frac{\partial L}{\partial w_{kn}} \;=\; \sum_{i=1}^{l} x_{in} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \;-\; \delta_{y^i=k} \sum_{i=1}^{l} x_{in} \;. \tag{3.148}$$

To compute the second derivatives of the objective, we need the derivatives of the probabilities with respect to the parameters:

$$\begin{aligned} \frac{\partial p(v \mid \boldsymbol{x}^i; \boldsymbol{W})}{\partial w_{vm}} \;&=\; x_{im}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \;\left(1 \;-\; p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \\ \frac{\partial p(k \mid \boldsymbol{x}^i; \boldsymbol{W})}{\partial w_{vm}} \;&=\; x_{im}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\, p(v \mid \boldsymbol{x}^i; \boldsymbol{W}))\;. \end{aligned} \tag{3.149}$$

The second derivatives of L with respect to the parameters w's are

$$\begin{aligned} &H_{kn,vm} \;=\; \frac{\partial L}{\partial w_{kn} \partial w_{vm}} \;= \\ &\sum_{i=1}^{l} x_{in}\, x_{im}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \;\left(\delta_{k=v}\,(1 \;-\; p(k \mid \boldsymbol{x}^i; \boldsymbol{W})) \;- \right. \\ &\left. (1 \;-\; \delta_{k=v}) p(v \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \;. \end{aligned} \tag{3.150}$$

Again we define a vector $\boldsymbol{a}$ with components a_{uj} (note, the double index is considered as single index so that a matrix is written as vector).

We consider the bilinear from

$$
\begin{aligned}
&\boldsymbol{a}^T \boldsymbol{H}\, \boldsymbol{a} \;= \\
&\sum_{k,n}\sum_{v,m}\sum_{i} a_{kn}\, a_{vm}\, x_{in}\, x_{im}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\, \left(\delta_{k=v}\, (1 \;-\; p(k \mid \boldsymbol{x}^i; \boldsymbol{W})) - \right. \\
&\left. (1 \;-\; \delta_{k=v})\, p(v \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \;= \\
&\sum_{k,n}\sum_{i} a_{kn}\, x_{in}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \sum_{m} x_{im} \left(a_{km} \;-\; \sum_{v} a_{vm} p(v \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \;= \\
&\sum_{i}\sum_{n} x_{in} \sum_{k} a_{kn}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \sum_{m} x_{im} \left(a_{km} \;-\; \sum_{v} a_{vm} p(v \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \;= \\
&\sum_{i} - \left\{ \left(\sum_{n} x_{in} \sum_{k} a_{kn}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \left(\sum_{m} x_{im} \sum_{v} a_{vm} p(v \mid \boldsymbol{x}^i; \boldsymbol{W})\right) \right\} + \\
&\left\{ \sum_{n} x_{in} \sum_{k} a_{kn}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \sum_{m} x_{im}\, a_{km} \right\} \;= \\
&\sum_{i} - \left\{ \left(\sum_{n} x_{in} \sum_{k} a_{kn}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\right)^2 \right\} + \\
&\left\{ \sum_{k} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \left(\sum_{n} x_{in}\, a_{kn}\right) \left(\sum_{m} x_{im}\, a_{km}\right) \right\} \;= \\
&\sum_{i} - \left\{ \left(\sum_{n} x_{in} \sum_{k} a_{kn}\, p(k \mid \boldsymbol{x}^i; \boldsymbol{W})\right)^2 \right\} + \\
&\left\{ \sum_{k} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \left(\sum_{n} x_{in}\, a_{kn}\right)^2 \right\} .
\end{aligned}
\tag{3.151}
$$

If for each summand of the sum over i

$$
\begin{aligned}
&\sum_{k} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \left(\sum_{n} x_{in}\, a_{kn}\right)^2 \;-\; \left(\sum_{k} p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \sum_{n} x_{in}\, a_{kn}\right)^2 \\
&\geq\; 0
\end{aligned}
\tag{3.152}
$$

holds, then the Hessian $\boldsymbol{H}$ is positive semidefinite. This holds for arbitrary number of samples as each term corresponds to a sample.

In last equation the $p(k \mid \boldsymbol{x}^i; \boldsymbol{W})$ can be viewed as a multinomial distribution over k. The terms $\sum_n x_{in}\, a_{kn}$ can be viewed as functions depending on k.

In this case $\sum_k p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \left(\sum_n x_{in}\, a_{kn}\right)^2$ is the second moment and the squared expectation is $\left(\sum_k\; p(k \mid \boldsymbol{x}^i; \boldsymbol{W}) \sum_n x_{in}\, a_{kn}\right)^2$. Therefore the left hand side of inequality (3.152) is the second central moment, which is larger than zero.

Alternatively inequality (3.152) can be proven by applying Jensen's inequality with the square function as a convex function.

We have proven that the Hessian $\boldsymbol{H}$ is positive semidefinite.

Adding a term like $\frac{1}{2} \sum_k \boldsymbol{w}_k^T \boldsymbol{w}_k$ to the objective for regularization, then the Hessian of the objective is strict positive definite.

3.6 Statistical Learning Theory

In this section we address the question whether learning from training data that means selecting a model based on training examples is useful for processing future data. Is a model which explains the training data an appropriate model of the world, i.e. also explains new data?

We will see that how useful a model selected based on training data is determined by its complexity. We will introduce the VC-dimension as complexity measure.

A main issue in statistical learning theory is to derive error bounds for the generalization error. The error bound is expressed as the probability of reaching a certain error rate on future data if the model is selected according to the training data.

Finally from the error bounds it will be seen that model complexity and training data mismatch of the model must be simultaneously minimized. This principle is called "structural risk minimization".

This statistical learning theory is based on two simple principles (1) the uniform law of large numbers (for inductive interference, i.e. the empirical risk minimization) and (2) complexity constrained models (structural risk minimization).

A first theoretical error bound on the mean squared error was given as the bias-variance formulation in eq. (3.40). The bias term corresponds to training data mismatch of the model whereas the variance term corresponds to model complexity. Higher model complexity leads to more models which fit the training data equally good, therefore the variance is larger. However the bias-variance formulation was derived for the special case of mean squared error. We will generalize this formulation in this section. Also the variance term will be expressed as model complexity for which measurements are available.

First we will start with some examples of error bounds.

3.6.1 Error Bounds for a Gaussian Classification Task

We revisit the Gaussian classification task from Section 3.3.

The minimal risk is given in eq. (3.30) as

$$R_{\min} = \int_X \min\{p(\boldsymbol{x}, y=-1), p(\boldsymbol{x}, y=1)\}\, d\boldsymbol{x}\,, \tag{3.153}$$

which can be written as

$$R_{\min} = \int_X \min\{p(\boldsymbol{x} \mid y=-1)\ p(y=-1)\ ,\ p(\boldsymbol{x} \mid y=1)\ p(y=1)\}\, d\boldsymbol{x}\,. \tag{3.154}$$

For transforming the minimum into a continuous function we will use the inequality

$$\forall_{a,b>0} : \forall_{0\leq\beta\leq 1} : \ \min\{a,b\} \ \leq \ a^{\beta}\, b^{1-\beta} \, . \tag{3.155}$$

To proof this inequality, without loss of generality we assume $a \geq b$ and have to show that $b \ \leq \ a^{\beta}\, b^{1-\beta}$. This is equivalent to $b \ \leq \ (a/b)^{\beta}\, b$ which is valid because $(a/b)^{\beta} \ \geq \ 1$.

Now we can bound the error by

$$\begin{aligned} &\forall_{0\leq\beta\leq 1} : \ R_{\min} \ \leq \ (p(y=1))^{\beta} \ (p(y=-1))^{1-\beta} \\ &\int_X (p\,(\boldsymbol{x} \mid y=1))^{\beta} \ (p\,(\boldsymbol{x} \mid y=-1))^{1-\beta} \ d\boldsymbol{x} \, . \end{aligned} \tag{3.156}$$

Up to now we only assumed a two class problem and did not make use of the Gaussian assumption.

The Gaussian assumption allows to evaluate above integral analytically:

$$\int_X (p\,(\boldsymbol{x} \mid y=1))^{\beta} \ (p\,(\boldsymbol{x} \mid y=-1))^{1-\beta} \ d\boldsymbol{x} \ = \ \exp(-\,v(\beta)) \, , \tag{3.157}$$

where

$$\begin{aligned} v(\beta) \ &= \ \frac{\beta(1-\beta)}{2} \\ &(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \ (\beta\, \boldsymbol{\Sigma}_1 \ + \ (1-\beta)\, \boldsymbol{\Sigma}_2)^{-1} \, (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ &+ \frac{1}{2} \ln \frac{|\beta\, \boldsymbol{\Sigma}_1 \ + \ (1-\beta)\, \boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|^{\beta} \ |\boldsymbol{\Sigma}_2|^{1-\beta}} \, . \end{aligned} \tag{3.158}$$

The *Chernoff bound* is obtained by maximizing $v(\beta)$ with respect to β and substituting this β into eq. (3.156).

The optimization has to be done in an one dimensional space which is very efficient.

The bound obtained by setting $\beta = \frac{1}{2}$ is called the *Bhattacharyya bound*:

$$\begin{aligned} v(1/2) \ &= \ \frac{1}{4} \, (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \ (\boldsymbol{\Sigma}_1 \ + \ \boldsymbol{\Sigma}_2)^{-1} \, (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ &+ \frac{1}{2} \ln \frac{\left| \frac{\boldsymbol{\Sigma}_1 \, + \, \boldsymbol{\Sigma}_2}{2} \right|}{\sqrt{|\boldsymbol{\Sigma}_1| \ |\boldsymbol{\Sigma}_2|}} \, . \end{aligned} \tag{3.159}$$

3.6.2 Empirical Risk Minimization

The empirical risk minimization principle states that if the training set is explained by the model then the model generalizes to future examples.

In the following considerations we need to restrict the complexity of the model class in order to obtain statements about the empirical error.

empirical risk minimization: minimize training error

3.6.2.1 Complexity: Finite Number of Functions

In this subsection we give an intuition why complexity matters. We restrict the definition of the complexity to the number M of functions from which the model can be selected.

First we are interested on the difference between the training error, *the empirical risk*, and the test error, *the risk*.

In eq. (3.2) the risk has been defined as

$$R(g) \;=\; \mathrm{E}_{\boldsymbol{z}}\left(L(y, g(\boldsymbol{x}))\right) \;. \tag{3.160}$$

We define the empirical risk R_{emp} analog to the cross-validation risk in eq. (3.14) as

$$R_{\mathrm{emp}}(g, \boldsymbol{Z}) \;=\; \frac{1}{l}\sum_{i=1}^{l} L\left(y^i, g\left(\boldsymbol{x}^i\right)\right) \;. \tag{3.161}$$

We will write $R_{\mathrm{emp}}(g, l)$ instead of $R_{\mathrm{emp}}(g, \boldsymbol{Z})$ to indicate the size of the training set.

We assume that we chose our model g from a finite set of functions

$$\{g_1, \ldots, g_M\} \;. \tag{3.162}$$

The difference between the empirical risk R_{emp} and the risk R can be different for each of the functions g_i. A priori we do not know which function g_i will be selected by the training procedure, therefore we will consider the worst case that is the maximal distance between R_{emp} and R on the set of functions:

$$\max_{j=1,\ldots,M} \left\| R_{\mathrm{emp}}(g_j, l) \;-\; R(g_j) \right\| \;. \tag{3.163}$$

We now consider the probability that the difference is large than ϵ:

$$\begin{aligned} & p\left(\max_{j=1,\ldots,M} \left\| R_{\mathrm{emp}}(g_j, l) \;-\; R(g_j) \right\| \;>\; \epsilon\right) \;\leq \\ & \sum_{j=1}^{M} p\left(\left\| R_{\mathrm{emp}}(g_j, l) \;-\; R(g_j) \right\| \;>\; \epsilon\right) \;\leq \\ & M\; 2\; \exp\left(-\,2\,\epsilon^2\, l\right) \;=\; 2\; \exp\left(\left(\frac{\ln M}{l} \;-\; 2\,\epsilon^2\right)\; l\right) \;=\; \delta \;, \end{aligned} \tag{3.164}$$

where the first "$\leq$" comes from the fact that $p(a \text{ OR } b) \;\leq\; p(a) \;+\; p(b)$ (this is called the "union bound") and the second "$\leq$" is the Chernoff inequality for each g_j. The Chernoff inequality bounds the difference between empirical mean (the average) and the expectation. The one-sided Chernoff inequality is

$$p(\mu_l \;-\; s \;>\; \epsilon) \;<\; \exp\left(-\,2\,\epsilon^2\, l\right) \;, \tag{3.165}$$

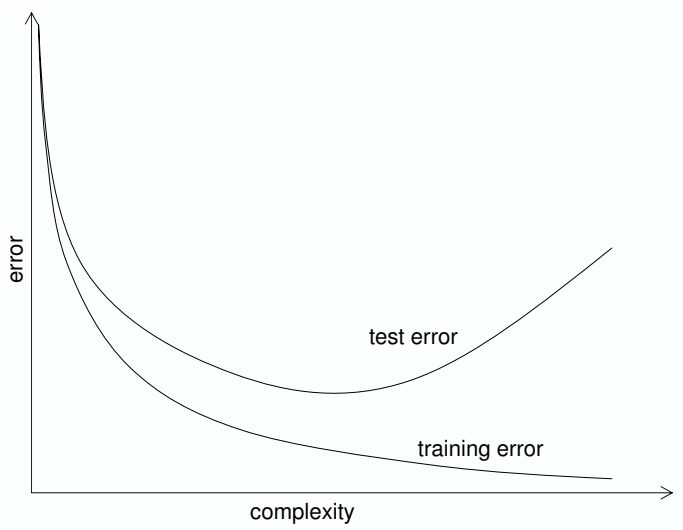


Figure 3.15: Typical example where the test error first decreases and then increases with increasing complexity. The training error decreases with increasing complexity. The test error, the risk, is the sum of training error and a complexity term. At some complexity point the training error decreases slower than the complexity term increases – this is the point of the optimal test error.

where μ_l is the empirical mean of the true value s for l trials.

Above last equation is valid for a two-sided bound. For a one-sided bound, we obtain

$$\epsilon(l, M, \delta) \;=\; \sqrt{\frac{\ln M \;-\; \ln(\delta)}{2\,l}}\;. \tag{3.166}$$

The value $\epsilon(l, M, \delta)$ is a complexity term depending on the number l of training examples, the number of possible functions M, and the confidence $(1 - \delta)$.

Theorem 3.4 (Finite Set Error Bound)
With probability of at least $(1 \;-\; \delta)$ *over possible training sets with* l *elements and for* M *possible functions we have*

$$R(g) \;\leq\; R_{\mathrm{emp}}(g, l) \;+\; \epsilon(l, M, \delta)\;. \tag{3.167}$$

Fig. 3.15 shows the relation between the test error $R(g)$ and the training error as a function of the complexity. The test error R first decreases and then increases with increasing complexity. The training error decreases with increasing complexity. The test error R is the sum of training error and a complexity term. At some complexity point the training error decreases slower than the complexity term increases – this is the point of the optimal test error.

In order that $\epsilon(l, M, \delta)$ converges to zero with increasing l we must assure that

$$\frac{\ln M}{l} \;\overset{l\to\infty}{\to}\; 0\;. \tag{3.168}$$

Because M is finite this expression is true.

However in most machine learning applications models are chosen from an infinite set of functions. Therefore we need another measure for the complexity instead the measure based on M, the number of functions.

3.6.2.2 Complexity: VC-Dimension

The main idea in this subsection is that on a given training set only a finite number of functions can be distinguished from one another. For example in a classification task all discriminant functions g which lead to the same classification function sign$g(.)$ build one equivalence class.

We will again use parametric models $g(.; \boldsymbol{w})$ with parameter vector $\boldsymbol{w}$.

We first want to answer the following question. Does minimizing the empirical risk with respect to parameter vector (e.g. minimizing the training error) convergence to the best solution with increasing training set, i.e. do we select better models with larger training sets? This question asks whether the empirical risk minimization (ERM) is consistent or not.

We first have to define the parameter $\hat{\boldsymbol{w}}_l$ which minimizes the empirical risk as

$$\hat{\boldsymbol{w}}_l \;=\; \arg\min_{\boldsymbol{w}} R_{\text{emp}}(g(.; \boldsymbol{w}), l) \;. \tag{3.169}$$

The ERM is *consistent* if

$$R(g(.; \hat{\boldsymbol{w}}_l)) \;\overset{l\to\infty}{\to}\; \inf_{\boldsymbol{w}} \; R(g(.; \boldsymbol{w})) \tag{3.170}$$

$$R_{\text{emp}}(g(.; \hat{\boldsymbol{w}}_l), l) \;\overset{l\to\infty}{\to}\; \inf_{\boldsymbol{w}} \; R(g(.; \boldsymbol{w})) \tag{3.171}$$

hold, where the convergence is in probability.

The ERM is consistent if it generates sequences of $\hat{\boldsymbol{w}}_l,\; l = 1, 2, \ldots$, for which both the risk evaluated with the function parameterized by $\hat{\boldsymbol{w}}_l$ and the empirical risk evaluated with the same function converge in probability to the minimal possible risk given the parameterized functions. The consistency is depicted in Fig. 3.16.

The ERM is *strictly consistent* if for all

$$\Lambda(c) \;=\; \left\{ \boldsymbol{w} \mid \boldsymbol{z} = (\boldsymbol{x}, y),\; \int L(y, g(\boldsymbol{x}; \boldsymbol{w}))p(\boldsymbol{z})d\boldsymbol{z} \;\geq\; c \right\} \tag{3.172}$$

the convergence

$$\inf_{\boldsymbol{w}\in\Lambda(c)} \; R_{\text{emp}}(g(.; \boldsymbol{w}), l) \;\overset{l\to\infty}{\to}\; \inf_{\boldsymbol{w}\in\Lambda(c)} \; R(g(.; \boldsymbol{w})) \tag{3.173}$$

holds, where the convergence is in probability.

The convergence holds for all subsets of functions where the functions with risk smaller that c are removed.

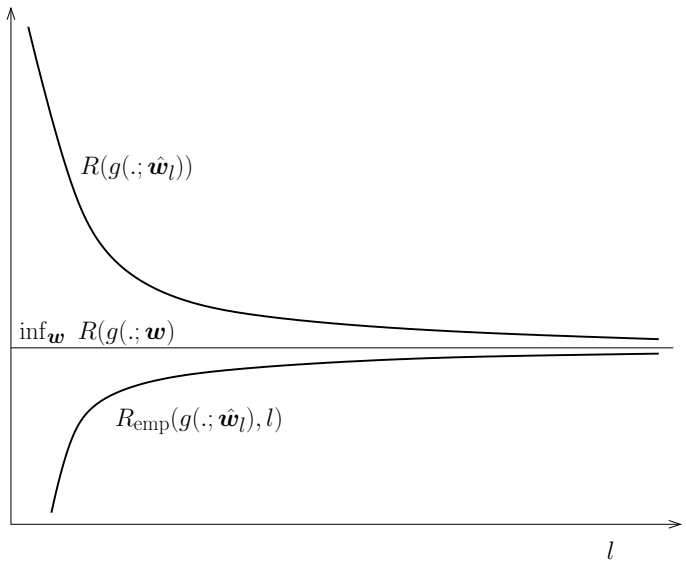


Figure 3.16: The consistency of the empirical risk minimization is depicted. The risk $R(g(.;\hat{\boldsymbol{w}}_l))$ of the optimal training parameter $\hat{\boldsymbol{w}}_l$ and the empirical risk $R_{\text{emp}}(g(.;\hat{\boldsymbol{w}}_l), l)$ for the optimal training parameter $\hat{\boldsymbol{w}}_l$ converge to the minimal possible risk $\inf_{\boldsymbol{w}}\ R(g(.;\boldsymbol{w}))$.

In the following we only focus on strictly consistency and mean "strictly consistent" if we write "consistent".

The maximum likelihood method for a set of densities with $0 < a \leq p(\boldsymbol{x};\boldsymbol{w}) \leq A < \infty$ is (strictly) consistent if

$$\forall_{\boldsymbol{w}_1}: \tag{3.174}$$
$$\inf_{\boldsymbol{w}} \frac{1}{l} \sum_{i=1}^{l} (-\ \ln p(\boldsymbol{x};\boldsymbol{w}))\ \stackrel{l\to\infty}{\to}\ \inf_{\boldsymbol{w}} \int_X (-\ \ln p(\boldsymbol{x};\boldsymbol{w}))\ p(\boldsymbol{x};\boldsymbol{w}_1)\ \boldsymbol{dx}\ .$$

If above convergence takes place for just one specific density $p(\boldsymbol{x};\boldsymbol{w}_0)$ then maximum likelihood is consistent.

Under what conditions is the ERM consistent?

In order to express these conditions we have to introduce new capacity measures: number of points to be shattered, the entropy, the annealed entropy, the growth function, and finally the VC-dimension.

For the complexity measure we first restrict ourselves to classification. Regression can be approximated through classification by dividing the output range into intervals of length ϵ and defining for each interval a class.

How many possibilities exist to label the input data $\boldsymbol{x}^i$ by binary labels $y^i \in \{-1, 1\}$? Clearly each binary vector of length l represents a labeling, therefore we obtain 2^l labelings.

Is our model class complex enough to produce any labeling vector based on the inputs? Not all model classes can do that. Therefore we can define as a complexity measure the number of data points a model class can assign all binary vectors. Assigning all possible binary vectors is

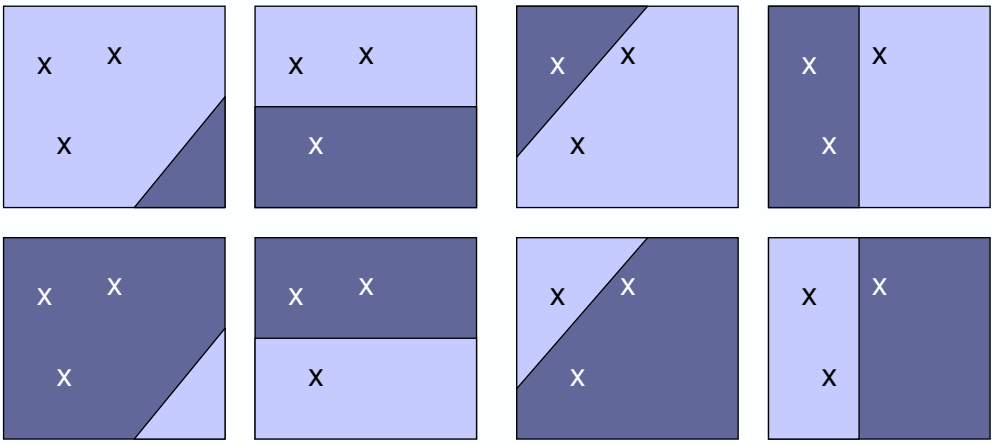


Figure 3.17: Linear decision boundaries can shatter any 3 points in a 2-dimensional space. Black crosses are assigned to -1 and white to +1.

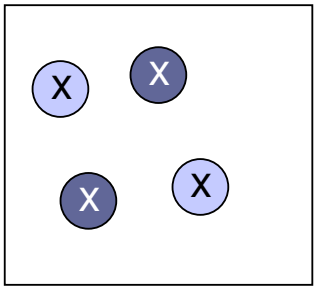


Figure 3.18: Linear decision boundaries *cannot* shatter any 4 points in a 2-dimensional space. Black crosses are assigned to -1 and white to +1, this label assignment cannot be represented by a linear function.

called *shattering* the points. Fig. 3.17 depicts the shattering of 3 points 2-dimensional space. Fig. 3.18 shows a specific labeling of 4 points in a 2-dimensional space which cannot be represented by a linear function. The complexity of linear functions in a 2-dimensional space is that they can shatter 3 points.

The number of points a function class can shatter will be introduced as the *VC-dimension*.

However we will do it more formally.

The *shattering coefficient* of a function class $\mathcal{F}$ with respect to inputs $x^i,\ 1 \leq i \leq l$ is the cardinality of $\mathcal{F}$ if restricted to the l input vectors $x^i,\ 1 \leq i \leq l$ (on input vectors distinguishable functions in $\mathcal{F}$). The shattering coefficient is denoted by

$$N_{\mathcal{F}}(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l)\ . \tag{3.175}$$

The *entropy of a function class* is

$$H_{\mathcal{F}}(l) \ = \ \mathrm{E}_{(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^l)}\ \ln N_{\mathcal{F}}(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l)\ . \tag{3.176}$$

The *annealed entropy of a function class* is

$$H_{\mathcal{F}}^{\mathrm{ann}}(l) \;=\; \ln \mathrm{E}_{(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^l)}\, N_{\mathcal{F}}(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^l)\;. \tag{3.177}$$

Until now we defined entropies, which are based on a probability measure on the observations in order to have a well-defined expectation.

The next definition avoids any probability measure. The *growth function of a function class* is

$$G_{\mathcal{F}}(l) \;=\; \ln \sup_{(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^l)} N_{\mathcal{F}}(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^l)\;. \tag{3.178}$$

Note that

$$H_{\mathcal{F}}(l) \;\leq\; H_{\mathcal{F}}^{\mathrm{ann}}(l) \;\leq\; G_{\mathcal{F}}(l)\;, \tag{3.179}$$

where the first inequality comes from Jensen's inequality and the second is obvious as the supremum is larger than or equal to the expectation.

Theorem 3.5 (Sufficient Condition for Consistency of ERM)
If

$$\lim_{l\to\infty} \frac{H_{\mathcal{F}}(l)}{l} \;=\; 0 \tag{3.180}$$

then ERM is consistent.

For the next theorem we need to define what fast rate of convergence means. Fast rate of convergence means exponential convergence. ERM has a *fast rate of convergence* if

$$p\left(\sup_{\boldsymbol{w}} |R(g(.;\boldsymbol{w})) \;-\; R_{\mathrm{emp}}(g(.;\boldsymbol{w}),l)| \;>\; \epsilon\right) \;<\; b\, \exp\left(-\, c\, \epsilon^2\, l\right) \tag{3.181}$$

holds true.

Theorem 3.6 (Sufficient Condition for Fast Rate of Convergence of ERM)
If

$$\lim_{l\to\infty} \frac{H_{\mathcal{F}}^{\mathrm{ann}}(l)}{l} \;=\; 0 \tag{3.182}$$

then ERM has a fast rate of convergence.

The last two theorems were valid for a given probability measure on the observations. The probability measure enters the formulas via the expectation E. The growth function however does not use a probability measure.

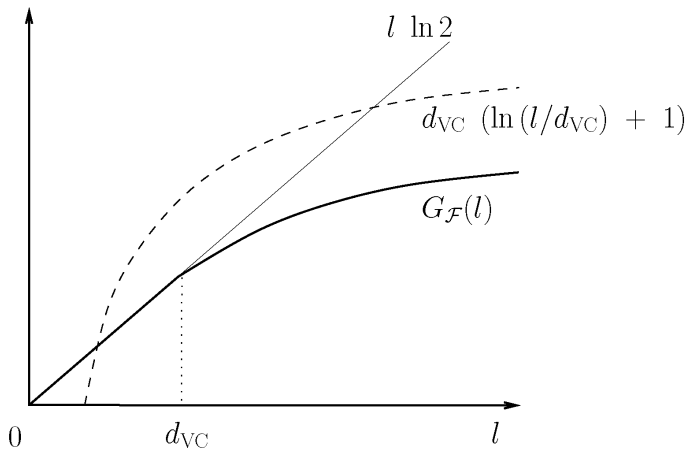


Figure 3.19: The growth function is either linear or logarithmic in l.

Theorem 3.7 (Consistency of ERM for Any Probability)
The condition

$$\lim_{l \to \infty} \frac{G_{\mathcal{F}}(l)}{l} = 0 \tag{3.183}$$

is necessary and sufficient for the ERM to be consistent and also ensures a fast rate of convergence.

As can be seen from above theorems the growth function is very important as it is valid for arbitrary distributions of the $\boldsymbol{x}$.

We define d_{VC} as the largest integer for which $G_{\mathcal{F}}(l) = l \ln 2$ holds:

$$d_{\mathrm{VC}} = \max_{l} \{ l \mid G_{\mathcal{F}}(l) = l \ln 2 \} . \tag{3.184}$$

If the maximum does not exists then we set $d_{\mathrm{VC}} = \infty$. The value d_{VC} is called the *VC-dimension* of the function class $\mathcal{F}$. The name VC-dimension is an abbreviation of Vapnik-Chervonenkis dimension. The VC-dimension d_{VC} is the maximum number of vectors that can be shattered by the function class $\mathcal{F}$.

Theorem 3.8 (VC-Dimension Bounds the Growth Function)
The growth function is bounded by

$$G_{\mathcal{F}}(l) \begin{cases} = & l \ln 2 & \text{if } l \leq d_{\mathrm{VC}} \\ \leq & d_{\mathrm{VC}} \left(1 + \ln \frac{l}{d_{\mathrm{VC}}} \right) & \text{if } l > d_{\mathrm{VC}} \end{cases} . \tag{3.185}$$

Fig. 3.19 depicts the statement of this theorem, that the growth function $G_{\mathcal{F}}(l)$ is either linear in l or logarithmic in l.

It follows immediately that a function class with finite VC-dimension is consistent and converges fast.

However the VC-dimension allows to derive bounds on the risk as we already have shown for function classes with finite many functions.

We now want to give examples for the VC dimension of some function classes.

- *Linear functions*: The VC dimension of linear discriminant functions $g(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$ is $d_{\mathrm{VC}} = d$, where d is the dimension of the input space. The VC dimension of linear discriminant functions $g(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x} + b$ with offset b is

$$d_{\mathrm{VC}} = d + 1 \,. \tag{3.186}$$

- *Nondecreasing nonlinear one-dimensional functions*: The VC dimension of discriminant functions in 1D of the form $\sum_{i=1}^{k} \left| a_i \, x^i \right| \, \mathrm{sign} x + a_0$ is one. These functions are nondecreasing in x, therefore they can shatter only one point: $d_{\mathrm{VC}} = 1$. The VS-dimension is independent of number of parameters.

- *Nonlinear one-dimensional functions*: The VC dimension of discriminant functions in 1D of the form $\sin(w \, z)$ defined on $[0, 2\pi]$ is infinity: $d_{\mathrm{VC}} = \infty$. This can be seen because there exist l points $x^1, \ldots, x^l$ for which a w_0 exists for which $\sin(w_0 \, z)$ is a discriminant function.

- *Neural Networks:* $d_{\mathrm{VC}} \leq 2 \, W \, \log_2(e \, M)$ for multi-layer perceptions, where M are the number of units, W is the number of weights, e is the base of the natural logarithm (Baum & Haussler 89, Shawe-Taylor & Anthony 91). $d_{\mathrm{VC}} \leq 2 \, W \, \log_2(24 \, e \, W \, D)$ according to Bartlett & Williamson (1996) for inputs restricted to $[-D; D]$.

In the following subsections we will report error bound which can be expressed by the VC-dimension instead of the growth function, which allows to compute the actual bounds for some functions classes.

3.6.3 Error Bounds

The idea of deriving the error bounds is to define the set of distinguishable functions. This set has cardinality of $N_{\mathcal{F}}$, the number of different separations of inputs.

Now we can proceed as in 3.6.2.1, where we had a finite set of functions. In eq. (3.164) we replace the maximum by the sum over all functions. For finite many functions the supremum reduces to the maximum and we can proceed as in eq. (3.164) and using $N_{\mathcal{F}}$ as the number of functions in the class.

Another trick in the proof of the bounds is to use two half-samples and their difference

$$p\left(\sup_{\boldsymbol{w}} \left| \frac{1}{l} \sum_{i=1}^{l} L\left(y^i, g\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) - \frac{1}{l} \sum_{i=l+1}^{2l} L\left(y^i, g\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \right| > \epsilon - \frac{1}{l} \right) \geq$$

$$\frac{1}{2} \, p\left(\sup_{\boldsymbol{w}} \left| \frac{1}{l} \sum_{i=1}^{l} L\left(y^i, g\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) - R(g(.; \boldsymbol{w}) \right| > \epsilon \right)$$

The probability that the difference of half samples exceeding a threshold is a bound on the probability that the difference of one half-sample to the risk exceeds a threshold ("symmetrization"). The symmetrization step reduced the risk which is defined on all possible samples to finite sample size. The difference on the half-samples counts how the loss on the first sample half differs from the second sample half.

Above considerations clarify why in the following bounds values derived from $N_{\mathcal{F}}$ (finite distinguishable functions) will appear and appear with arguments $2l$ (two half-samples).

Before we report the bounds, we define the minimal possible risk and its parameter:

$$\boldsymbol{w}_0 \;=\; \arg\min_{\boldsymbol{w}} R(g(.;\boldsymbol{w})) \tag{3.187}$$

$$R_{\min} \;=\; \min_{\boldsymbol{w}} R(g(.;\boldsymbol{w})) \;=\; R(g(.;\boldsymbol{w}_0))\;. \tag{3.188}$$

Theorem 3.9 (Error Bound)
With probability of at least $(1 \,-\, \delta)$ *over possible training sets with* l *elements, the parameter* $\boldsymbol{w}_l$ *(more precisely* $\boldsymbol{w}_l = \boldsymbol{w}\,(\boldsymbol{Z}_l)$*) which minimizes the empirical risk we have*

$$R(g(.;\boldsymbol{w}_l)) \;\leq\; R_{\text{emp}}(g(.;\boldsymbol{w}_l), l) \;+\; \sqrt{\epsilon(l,\delta)}\;. \tag{3.189}$$

With probability of at least $(1 \,-\, 2\delta)$ *the difference between the optimal risk and the risk of* $\boldsymbol{w}_l$ *is bounded by*

$$R(g(.;\boldsymbol{w}_l)) \;-\; R_{\min} \;<\; \sqrt{\epsilon(l,\delta)} \;+\; \sqrt{\frac{-\,\ln\delta}{l}}\;. \tag{3.190}$$

Here $\epsilon(l,\delta)$ *can be defined for a specific probability as*

$$\epsilon(l,\delta) \;=\; \frac{8}{l}\;(H^{\text{ann}}_{\mathcal{F}}(2l) \;+\; \ln(4/\delta)) \tag{3.191}$$

or for any probability as

$$\epsilon(l,\delta) \;=\; \frac{8}{l}\;(G_{\mathcal{F}}(2l) \;+\; \ln(4/\delta)) \tag{3.192}$$

where the later can be expressed though the VC-dimension d_{VC}

$$\epsilon(l,\delta) \;=\; \frac{8}{l}\;(d_{\text{VC}}\,(\ln(2l/d_{\text{VC}}) \;+\; 1) \;+\; \ln(4/\delta))\;. \tag{3.193}$$

The complexity measures depend all on the ratio $\frac{d_{\text{VC}}}{l}$, the VC-dimension of the class of function divided by the number of training examples.

The bound above is from [Schölkopf and Smola, 2002], whereas an older bound from Vapnik is

$$R(g(.;\boldsymbol{w}_l)) \;\leq\; R_{\text{emp}}(g(.;\boldsymbol{w}_l), l) \;+ \tag{3.194}$$
$$\frac{\epsilon(l,\delta)}{2}\left(1 \;+\; \sqrt{1 \;+\; \frac{R_{\text{emp}}(g(.;\boldsymbol{w}_l), l)}{\epsilon(l,\delta)}}\right)\;.$$

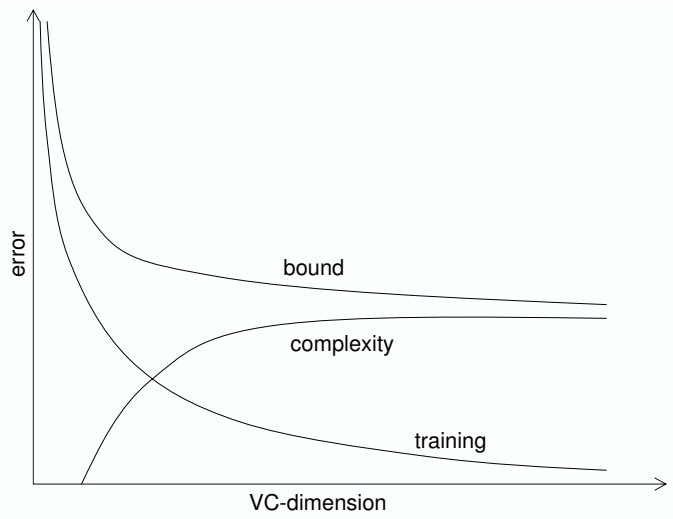


Figure 3.20: The error bound is the sum of the empirical error, the training error, and a complexity term. The complexity increases with increasing VC dimension whereas the training error decreases with increasing complexity.

It can be seen that the complexity term decreases with $\frac{1}{\sqrt{l}}$. If we have zero empirical risk then the bound on the risk decreases with $\frac{1}{\sqrt{l}}$.

Later in Section 4.5 we will see a bound on the *expected* risk which decreases with $\frac{1}{l}$ for the method of support vector machines.

The bound on the risk for the parameter $\boldsymbol{w}_l$ which minimized the empirical risk has again the form

$$R \leq R_{\text{emp}} + \text{complexity} . \tag{3.195}$$

This sum is depicted in Fig. 3.20, where it is also shown that the complexity term increases with the VC-dimension whereas the empirical error decreases with increasing VC-dimension.

Note that we again arrived at a bound which is similar to the bias-variance formulation from eq. (3.40), where the means squared error was expressed as bias term and a variance term. Bias corresponds to R_{emp} and variance to the complexity term. With increasing complexity of a function class the number of solutions with the same training error increases, that means the variance of the solutions increases.

In many practical cases the bound is not useful because only for large number of training examples l the bound gives a nontrivial value (trivial values are for example that the misclassification rate is smaller equal 1). In Fig. 3.21 the bound is shown as being far above the actual test error. However in many practical cases the minimum of the bound is close (in terms of complexity) to the minimum of the test error.

For regression instead of the shattering coefficient *covering numbers* can be used. The *ε-covering number of $\mathcal{F}$ with respect to metric d* is $\mathcal{N}(\varepsilon, \mathcal{F}, d)$ which is defined as the smallest number which ε-cover $\mathcal{F}$ using metric d. Usually the metric d is the distance of the function on

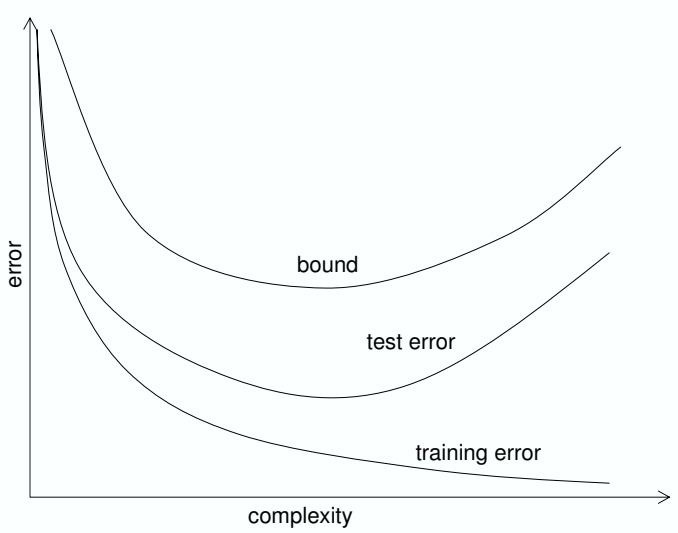


Figure 3.21: The bound on the risk, the test error, is depicted. However the bound can be much larger than the test error because it is valid for any distribution of the input.

the data set $\boldsymbol{X}$. For example the maximum norm on $\boldsymbol{X}$, that is the distance of two functions is the maximal difference of these two on the set $\boldsymbol{X}$, defines the covering number $\mathcal{N}(\varepsilon, \mathcal{F}, \boldsymbol{X}_\infty)$ The *ε-growth function* is the defined as

$$G(\varepsilon, \mathcal{F}, l) \;=\; \ln \sup_{\boldsymbol{X}} \mathcal{N}(\varepsilon, \mathcal{F}, \boldsymbol{X}_\infty) \,. \tag{3.196}$$

We obtain similar bounds on the generalization error like

$$R(g(.; \boldsymbol{w}_l)) \;\leq\; R_{\text{emp}}(g(.; \boldsymbol{w}_l), l) \;+\; \sqrt{\epsilon(\varepsilon, l, \delta)} \,, \tag{3.197}$$

where

$$\epsilon(\varepsilon, l, \delta) \;=\; \frac{36}{l} \; (\ln(12\, l) \;+\; G(\varepsilon/6, \mathcal{F}, l) \;-\; \ln \delta) \;. \tag{3.198}$$

Instead to minimizing the empirical risk it would be better to minimize the risk or at least a bound on them.

3.6.4 Structural Risk Minimization

The Structural Risk Minimization (SRM) principle minimizes the guaranteed risk that is a bound on the risk instead of the empirical risk alone.

In the SRM a nested set of function classes is introduced:

$$\mathcal{F}_1 \;\subset\; \mathcal{F}_2 \;\subset\; \ldots \;\subset\; \mathcal{F}_n \;\subset\; \ldots \;, \tag{3.199}$$

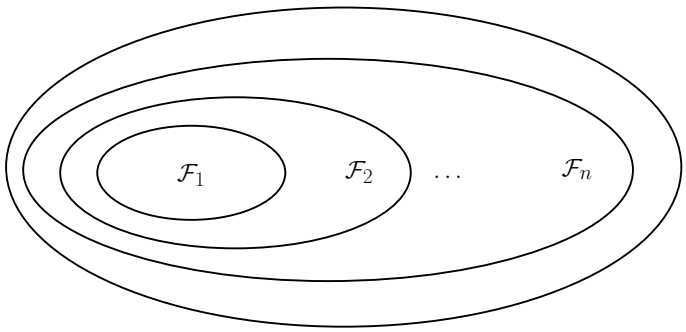


Figure 3.22: The structural risk minimization principle is based on a structure on a set of functions which is nested subsets $\mathcal{F}_n$ of functions.

where class $\mathcal{F}_n$ possesses VC-dimension d_{VC}^n and

$$d_{\mathrm{VC}}^1 \leq d_{\mathrm{VC}}^2 \leq \ldots d_{\mathrm{VC}}^n \leq \ldots \quad , \tag{3.200}$$

holds.

One realization of the SRM principle is the *minimum description length* [Rissanen, 1978] or *minimum message length* [Wallace and Boulton, 1968] principle. In the minimum description length principle a sender transmits a model and the inputs $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^l$ and the receiver has to recover the labels $y^1, y^2, \ldots, \boldsymbol{y}^l$ from the model and the inputs. If the model does not supply the exact y from the input x then the sender has also to transmit the error. Goal is to minimize the transmission costs, i.e. the description length.

For fixed l the error is R_{emp} and if the model complexity corresponds to the number of bits to describe it, then the risk R is analog to the transmission cost:

$$\mathrm{transmissioncosts} = R_{\mathrm{emp}} + \mathrm{complexity}\,. \tag{3.201}$$

Minimizing the transmissions costs is equivalent to minimizing the risk for appropriate error (coding the error) and appropriate model coding which defines the complexity.

If the model codes main structures in the data, then for many training examples (assume large l) the error description can be reduces. If however the model codes specific values for one or few training points which may even correspond to random noise then is should not be transmitted via the model. Transmitting the specific values through the error would be more efficient in terms of bits than coding these values into the model. That means the model should contain rules which apply to as many data points as possible whereas data point specific variations or noise should be contained in the error.

Is the SRM principle consistent? How fast does it converge?

The SRM is always consistent and even supplies a bound on the rate of convergence. That means the SRM procedure converges to the best possible solution with probability one as the number of examples l goes to infinity.

The asymptotic rate of convergence is

$$r(l) = |R_{\mathrm{min}}^n - R_{\mathrm{min}}| + \sqrt{\frac{d_{\mathrm{VC}}^n \ln l}{l}}\,, \tag{3.202}$$

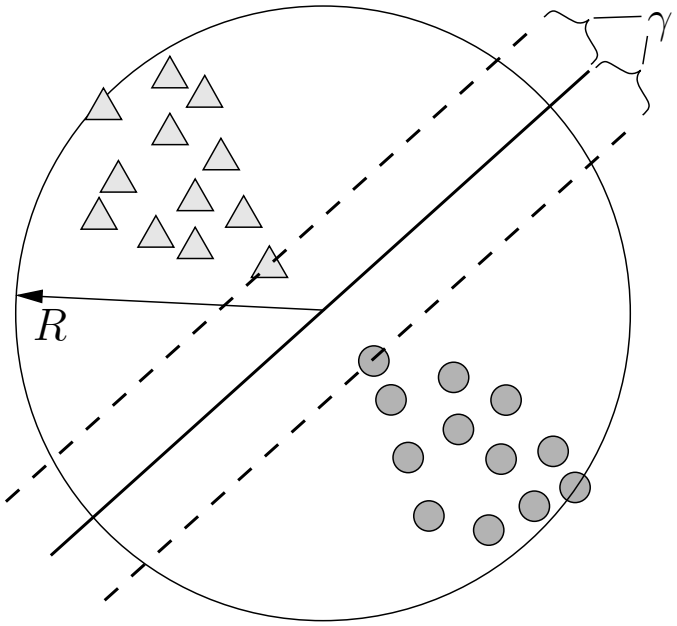


Figure 3.23: Data points are contained in a sphere of radius R at the origin. A linear discriminant function with a boundary having distance γ to all data points is depicted.

where $R^n_{\min}$ is the minimal risk of the function class $\mathcal{F}_n$. Asymptotic rate of convergence means that

$$p\left(\lim_{l\to\infty} \sup\, r^{-1}(l)\, \left|R\left(g\left(.\,;\, \boldsymbol{w}_l^{\mathcal{F}_n}\right)\right) \,-\, R_{\min}\right| \,<\, \infty\right) \,=\, 1\,. \tag{3.203}$$

We assume that $n \,=\, n(l)$ increases with the number of training examples so that for large enough training examples $|R^n_{\min} \,-\, R_{\min}| \overset{l\to\infty}{\to} 0$.

If the optimal solution belongs to some class $\mathcal{F}_n$ then the convergence rate is

$$r(l) \,=\, O\left(\sqrt{\frac{\ln l}{l}}\right)\,. \tag{3.204}$$

3.6.5 Margin as Complexity Measure

The VC-dimension can be bounded by different restrictions on the class of functions. The most famous restriction is that the zero isoline of the discriminant function (the boundary function), provided it separates the classes properly, has maximal distance γ (this distance will later be called "margin") to all data points which are contained in a sphere with radius R. Fig. 3.23 depicts such a discriminant function. Fig. 3.24 gives an intuition why a margin reduces the number of hyperplanes and therefore the VC-dimension.

The linear discriminant functions $\boldsymbol{w}^T\boldsymbol{x} \,+b$ can be scaled (scaling $\boldsymbol{w}$ and b) and give the same classification function sign $\left(\boldsymbol{w}^T\boldsymbol{x} \,+b\right)$. Of the class of discriminant functions leading to the same classification function we can choose one representative.

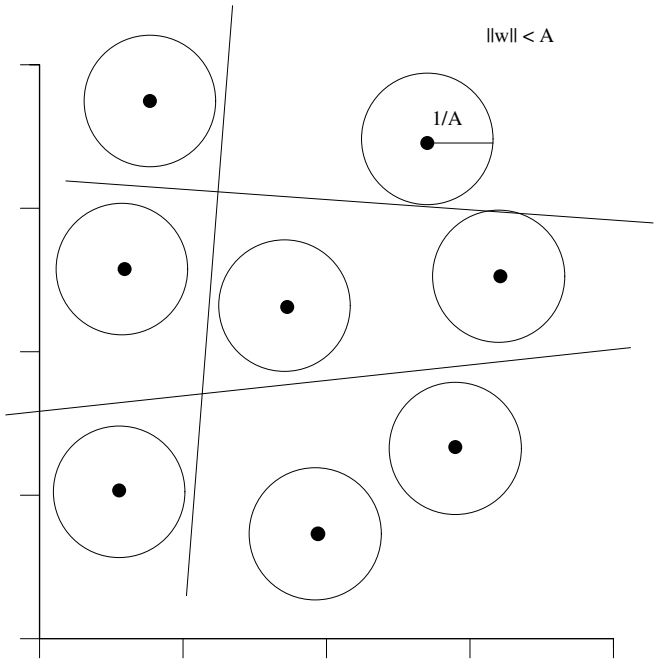


Figure 3.24: Margin means that hyperplanes must keep outside the spheres. Therefore the possible number of hyperplanes is reduced. Copyright © 1997 [Osuna et al., 1997].

The representative is chosen with respect to the training data and is called *the canonical form* w.r.t. the training data $\boldsymbol{X}$. In the canonical form $\boldsymbol{w}$ and b are scaled that

$$\min_{i=1,\ldots,l} \left| \boldsymbol{w}^T \boldsymbol{x}^i \; + \; b \right| \; = \; 1 \, . \tag{3.205}$$

Theorem 3.10 (Margin Bounds VC-dimension)
The class of classification functions sign $\left(\boldsymbol{w}^T \boldsymbol{x} \; + b\right)$*, where the discriminant function* $\boldsymbol{w}^T \boldsymbol{x} \; + b$ *is in its canonical form versus* $\boldsymbol{X}$ *which is contained in a sphere of radius* R*, and where* $\|\boldsymbol{w}\| \; \leq \; \frac{1}{\gamma}$ *satisfy*

$$d_{\mathrm{VC}} \; \leq \; \frac{R^2}{\gamma^2} \, . \tag{3.206}$$

This gives with the fact from eq. (3.186)

$$d_{\mathrm{VC}} \; \leq \; \min\{ \left[\frac{R^2}{\gamma^2} \right] , d\} \; + 1 \, , \tag{3.207}$$

where $[.]$ is the floor of a real number.

Remark: The VC-dimension is defined for a model class and should not depend on training set.

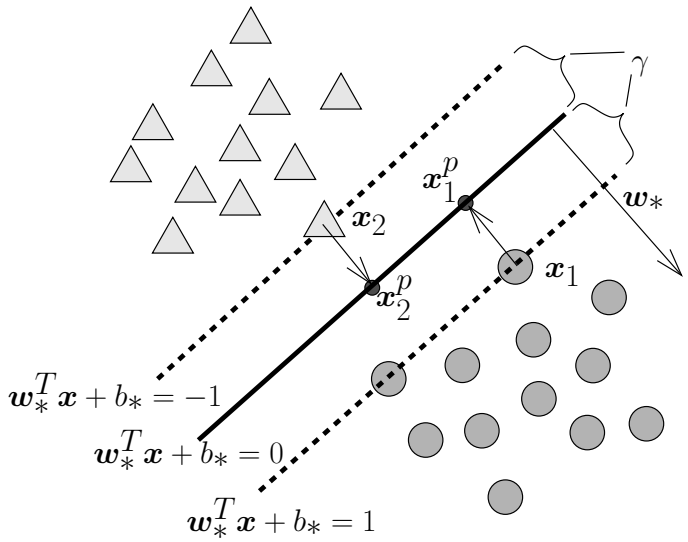


Figure 3.25: The offset b is optimized in order to obtain the largest $\|\boldsymbol{w}\|$ for the canonical form which is $\|\boldsymbol{w}_*\|$ for the optimal value b_*. Now there exist points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ with $\boldsymbol{w}_*^T\boldsymbol{x}^1 + b_* = 1$ and $\boldsymbol{w}_*^T\boldsymbol{x}^2 + b_* = -1$. The distance of $\boldsymbol{x}^1$ to the boundary function $\boldsymbol{w}_*^T\boldsymbol{x}^2 + b_* = 0$ is $\gamma = \frac{1}{\|\boldsymbol{w}_*\|}$. In the figure also $\left(\boldsymbol{x}^1\right)^p$ ($\left(\boldsymbol{x}^2\right)^p$), the projection of $\boldsymbol{x}^1$ ($\boldsymbol{x}^2$) onto the boundary functions is depicted.

If at least one data point exists for which the discriminant function $\boldsymbol{w}^T\boldsymbol{x} + b$ is positive and at least one data point exists for which it is negative, then we can optimize b and re-scale $\|\boldsymbol{w}\|$ in order to obtain the smallest $\|\boldsymbol{w}\|$ for discriminant function in the canonical form.

This gives the tightest bound $\frac{1}{\gamma}$ on $\|\boldsymbol{w}\|$ and therefore the smallest VC-dimension.

The optimization of b leads to the result that there exists an data point, without loss of generalization we denote it by $\boldsymbol{x}^1$, for which $\boldsymbol{w}^T\boldsymbol{x}^1 + b = 1$, and a data point without loss of generalization we denote it by $\boldsymbol{x}^2$, for which $\boldsymbol{w}^T\boldsymbol{x}^2 + b = -1$.

Fig. 3.25 depicts the situation.

To see above result, we consider (without loss of generalization) the case that the distance to the negative class is larger than 1:

$$\boldsymbol{w}^T\boldsymbol{x}^1 + b = 1 \tag{3.208}$$

and

$$\boldsymbol{w}^T\boldsymbol{x}^2 + b = -1 - \delta\,, \tag{3.209}$$

where $\delta > 0$, then

$$\boldsymbol{w}^T(\boldsymbol{x}^1 - \boldsymbol{x}^2) = 2 + \delta\,. \tag{3.210}$$

We set

$$\boldsymbol{w}_*^T = \frac{2}{2 + \delta}\,\boldsymbol{w} \tag{3.211}$$

which gives

$$\|\boldsymbol{w}_*\| \; < \; \|\boldsymbol{w}\| \; . \tag{3.212}$$

Further we set

$$b_* \; = \; 1 \; - \; \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \boldsymbol{x}^1 \; . \tag{3.213}$$

We obtain

$$\boldsymbol{w}_*^T \boldsymbol{x}^1 \; + \; b_* \; = \; \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \boldsymbol{x}^1 \; + \; 1 \; - \; \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \boldsymbol{x}^1 \; = \; 1. \tag{3.214}$$

and

$$\begin{aligned} & \boldsymbol{w}_*^T \boldsymbol{x}^2 \; + \; b_* \; = \; \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \boldsymbol{x}^2 \; + \; 1 \; - \; \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \boldsymbol{x}^1 \; = \\ & - \frac{2}{2 \; + \; \delta} \, \boldsymbol{w}^T \left(\boldsymbol{x}^1 - \boldsymbol{x}^2 \right) \; + \; 1 \; = \; - \frac{2}{2 \; + \; \delta} (2 \; + \; \delta) \; + \; 1 \; = \; - \, 1 \; . \end{aligned} \tag{3.215}$$

We can generalize above example. Without loss of generalization assume that

$$\begin{aligned} \boldsymbol{x}^1 \; & = \; \arg \min_{\boldsymbol{x}^i \colon \, y^i = 1} \{ \boldsymbol{w}^T \boldsymbol{x}^i \} \; \text{ and} \\ \boldsymbol{x}^2 \; & = \; \arg \max_{\boldsymbol{x}^i \colon \, y^i = -1} \{ \boldsymbol{w}^T \boldsymbol{x}^i \} \; . \end{aligned} \tag{3.216}$$

Then we set

$$\boldsymbol{w}_*^T \; = \; \frac{2}{\boldsymbol{w}^T \, (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2)} \, \boldsymbol{w} \tag{3.217}$$

which gives

$$\|\boldsymbol{w}_*\| \; < \; \|\boldsymbol{w}\| \tag{3.218}$$

because both $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ have at least a distance of 1 to the boundary functions which guarantees that $\boldsymbol{w}^T \, (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2) \; > \; 2$. Further we set

$$\begin{aligned} & b_* \; = \; \frac{2}{\boldsymbol{w}^T \, (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2)} \, \left(- \, \frac{1}{2} \, \boldsymbol{w}^T \, (\boldsymbol{x}^1 \; + \; \boldsymbol{x}^2) \right) \; = \\ & - \, \frac{\boldsymbol{w}^T \, (\boldsymbol{x}^1 \; + \; \boldsymbol{x}^2)}{\boldsymbol{w}^T \, (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2)} \; . \end{aligned} \tag{3.219}$$

This gives

$$\boldsymbol{w}_*^T \boldsymbol{x}^1 \; + \; b_* \; = \; \frac{2}{\boldsymbol{w}^T \left(\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2\right)} \left(\boldsymbol{w}^T \boldsymbol{x}^1 \; - \; \frac{1}{2} (\boldsymbol{x}^1 \; + \; \boldsymbol{x}^2) \right) \; = \tag{3.220}$$
$$\frac{2}{\boldsymbol{w}^T \left(\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2\right)} \left(\frac{1}{2} \boldsymbol{w}^T \boldsymbol{x}^1 \; - \; \frac{1}{2} \boldsymbol{w}^T \boldsymbol{x}^2 \right) \; = \; 1$$

and similarly

$$\boldsymbol{w}_*^T \boldsymbol{x}^2 \; + \; b_* \; = \; -1 \; . \tag{3.221}$$

We see that

$$\boldsymbol{w}_*^T (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2) \; = \; \boldsymbol{w}_*^T \boldsymbol{x}^1 \; + \; b_* \; - \; \boldsymbol{w}_*^T \boldsymbol{x}^2 \; + \; b_* \; = \; 2 \; . \tag{3.222}$$

For $\|\boldsymbol{w}\| \; = \; \alpha \; \|\boldsymbol{w}_*\|$ with $0 < \alpha < 1$ we would obtain $\boldsymbol{w}^T (\boldsymbol{x}^1 \; - \; \boldsymbol{x}^2) \; < \; 2$ and either $\boldsymbol{x}^1$ or $\boldsymbol{x}^2$ is closer than 1 to the boundary function which contradicts that the discriminant function is in the canonical form.

Therefore the optimal $\boldsymbol{w}_*$ and b_* are unique.

We want to compute the distance of $\boldsymbol{x}^1$ to the boundary function. The projection of $\boldsymbol{x}^1$ onto the boundary function is $\left(\boldsymbol{x}^1\right)^p \; = \; \boldsymbol{x}^1 \; - \; \alpha \boldsymbol{w}_*$ and fulfills

$$\begin{aligned} & \boldsymbol{w}_*^T \left(\boldsymbol{x}^1\right)^p \; + \; b_* = \; 0 \\ & \Rightarrow \; \boldsymbol{w}_*^T \left(\boldsymbol{x}^1 \; - \; \alpha \boldsymbol{w}_*\right) \; = \\ & \boldsymbol{w}_*^T \boldsymbol{x}^1 \; - \; \alpha \; \|\boldsymbol{w}_*\|^2 \; + \; b_* \; = \; 1 \; - \; \alpha \; \|\boldsymbol{w}_*\|^2 \; = \; 0 \\ & \Rightarrow \; \alpha \; = \; \frac{1}{\|\boldsymbol{w}_*\|^2} \\ & \Rightarrow \; \left(\boldsymbol{x}^1\right)^p \; = \; \boldsymbol{x}^1 \; - \; \frac{1}{\|\boldsymbol{w}_*\|^2} \boldsymbol{w}_* \end{aligned} \tag{3.223}$$

The distance of $\boldsymbol{x}^1$ to the boundary function is

$$\left\| \boldsymbol{x}^1 \; - \; \left(\boldsymbol{x}^1\right)^p \right\| \; = \; \left\| \boldsymbol{x}^1 \; - \; \boldsymbol{x}^1 \; - \; \frac{1}{\|\boldsymbol{w}_*\|^2} \boldsymbol{w}_* \right\| \; = \frac{1}{\|\boldsymbol{w}_*\|} \; = \; \gamma \; . \tag{3.224}$$

Similar the distance of $\boldsymbol{x}^2$ to the boundary function is

$$\frac{1}{\|\boldsymbol{w}_*\|} \; = \; \gamma \; . \tag{3.225}$$

Theorem 3.11 (Margin Error Bound)
The classification functions sign $\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$ *are restricted to* $\|\boldsymbol{w}\| \leq \frac{1}{\gamma}$ *and* $\|\boldsymbol{x}\| < R$*. Let* ν *be the fraction of training examples which have a margin (distance to* $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ *) smaller than* $\frac{\rho}{\|\boldsymbol{w}\|}$.

With probability at least of $(1 - \delta)$ *of drawing* l *examples, the probability to misclassify a new example is bounded from above by*

$$\nu + \sqrt{\frac{c}{l}\left(\frac{R^2}{\rho^2\,\gamma^2}\ln^2 l + \ln(1/\delta)\right)}, \tag{3.226}$$

where c *is a constant.*

The probability $(1 - \delta)$ is the confidence in drawing appropriate training samples whereas the bound is on the probability of drawing a test example. The bound is from Bartlett and Shawe-Taylor.

Again the bound is of the order $\nu + \frac{1}{\sqrt{l}}$.

In the next chapter we will introduce support vector machines as the method for structural risk minimization, where the margin is maximized.

Chapter 4

Support Vector Machines

In this chapter we focus on supervised learning with a method called "support vector machine" (SVM).

The SVM is based directly on the results from the end of previous chapter and realizes the principle of structural risk minimization (SRM).

SVM minimize the empirical risk simultaneously with a bound on the complexity, namely the margin.

Since the mid 90s with the papers of Cortes and Vapnik [Cortes and Vapnik, 1995] and Boser, Guyon, and Vapnik [Boser et al., 1992] and the book [Vapnik, 1995] by V. Vapnik, "Support Vector Machines" (SVMs) became very popular in the machine learning community.

Because SVMs allow for bounds on the future error, the risk, and have been proven very successful in different applications they were preferred of neural networks or other supervised learning methods. In contrast to neural networks the SVMs have an unique solution and can be solved by a convex quadratic optimization problem.

4.1 Support Vector Machines in Bioinformatics

Machine learning methods are the best performing methods in various bioinformatics domains.

For protein 3D structure prediction support vector machines showed better performance than "threading" methods in template identification (Cheng and Baldi, 2006).

Threading was the golden standard for protein 3D structure recognition if the structure is known (almost all structures are known).

Support vector machines were applied to the recognition of alternative splice sites and provided the so far best results (Gunnar Rätsch).

Protein Homology Detection. For protein homology detection SVMs were used in different was. First, the SVM-Fisher method [Jaakkola et al., 1999, 2000] couples an iterative HMM training scheme with the SVM. For the same task SVMs used the mismatch-kernel [Leslie et al., 2004b,a] The mismatch kernel measures sequence similarity through amino acid identities between the sequences where the frequency of almost identical subsequences is taken into account. The mismatch-kernel is related to the BLAT alignment algorithm [Kent, 2002]. The SVM-pairwise method according to [Liao and Noble, 2002] use as the feature vector the Smith-Waterman alignment scores to all other training sequences. In the SW-kernel the SW-pairwise scores are used as

kernel [Vert et al., 2004] (note, that this is not a valid kernel because it is not ensured to produce positive semi-definite kernels and the SVM-optimization is not well defined). Then local alignment (LA) kernel [Vert et al., 2004] which is similar to a local Smith-Waterman score and is based on gap-penalties and the BLOSUM similarity matrix. Recently the oligomer based distance SVM approach [Lingner and Meinicke, 2006] was proposed. The "HMMSTR" from [Hou et al., 2004] PSI-BLAST against SwissProt to generate a PSSM and thereafter uses a HMM and finally a SVM to classify sequences. In [Kuang et al., 2005] the mismatch kernel is generalized to process profiles obtained by PSI-BLAST applied to the NR data base and in [Rangwala and Karypis, 2005] profiles ("position-specific scoring matrix", PSSM) are used for local alignment-based and other kernels in connection to the SVM.

Logan et al. [Logan et al., 2001] propose kernels which are based on motifs as are the kernels by Ben-hur and Brutlag (2003) which use the eBLOCKS database (`http://motif.stanford.edu/eblocks`).

Gene Classification. Pavlidis et al. [Pavlidis et al., 2001] utilize the Fisher kernel for classifying genes according to the characteristics of their switching mechanisms. SVMs are successfully used to classify co-regulated genes in yeast. Zien et al. [Zien et al., 2000] use SVMs for gene finding.

Splice Sites. Degroeve et al. [Degroeve et al., 2002] recognize the starts of introns by SVMs.

Phylogenetic. Vert [Vert, 2002c,b] uses kernels and SVMs to mark phylogenetic profiles.

Protein Classification. Hua and Sun [Hua and Sun, 2001b] classify proteins according to their subcellular localization by SVMs.

Zavaljevski and Reifman [Zavaljevski and Reifman, 2002] apply SVMs to classify human antibody light chains into benign or pathogenic categories.

Vert [Vert, 2002a] uses an SVMs approach to recognizing the position at which a signal peptide is cleaved from the main protein.

RNA Processing. Carter et al. [Carter et al., 2001] identified functional RNAs in genomic DNA by SVMs.

Protein Secondary Structure Prediction. For example Hua and Sun [Hua and Sun, 2001a] used SVMs instead of the typically used neural networks for protein secondary structure prediction.

Microarrays and Gene Expression Profiles. SVMs are used for prognosis or therapy outcome prediction based on microarray data. SVM are first used for gene selection and then again for classification and prognosis.

The first SVM applications are by Golub et al. [Golub et al., 1999] and Mukherjee et al. [Mukherjee et al., 1999, 2000] who apply SVMs to leukemia data of the AML and ALL kind.

Pomeroy [Pomeroy et al., 2002] predicts the outcome of a therapy of embryonal brain tumors according to tissue samples analyzed by microarrays. Brown et al. [Brown et al., 2000] classify of yeast genes into functional categories based on SVMs. Moler et al. [Moler et al., 2000] use SVMs for the recognition of colon cancer described by microarrays. Segal et al. [Segal et al., 2003] apply SVMs to the analysis of a tumor called "clear cell sarcoma". The outcome of a breast cancer radiation or chemotherapy was predicted by van't Veer et al. [van't Veer et al., 2002] which was improved by [Hochreiter and Obermayer, 2004] by SVM-techniques. Gene expression changes

in Drosophila embryos were investigated by Myasnikova et al. [Myasnikova et al., 2002] using SVMs. Further publications where SVM are used to analyze microarray data include [Komura et al., 2005, Huang and Kecman, 2005, Zhang et al., 2006, Wang et al., 2006, Tang et al., 2006, Shen and Tan, 2006].

Gene Selection form Gene Expression Profiles. Furey et al. [Furey et al., 2000] focus on gene selection for AML/ALL leukemia and colon cancer. Guyon et al. [Guyon et al., 2002] developed a method called "recursive feature elimination (RFE)" for support vector machines and applied it to microarray data for gene selection. Su et al. [Su et al., 2003] offers a package called "RankGene" for gene ranking. In [Vert and Kanehisa, 2003] graph-driven feature extraction based on kernel methods are proposed. The breast cancer data published in [van't Veer et al., 2002] were overestimated because of a selection bias which comes from choosing relevant genes prior to cross-validation. This error was reported by Ambroise and McLachlan [Ambroise and McLachlan, 2002] and by Tibshirani and Efron [Tibshirani and Efron, 2002].

Methylation Data. Model et al. [Model et al., 2001] use SVMs for feature selection and classification on methylation arrays instead of cDNA microarrays.

Protein-Protein Interactions. Bock and Gough [Bock and Gough, 2001] apply SVM for protein-protein interaction tasks.

Mass Spectrometry. In tandem mass spectrometry samples of unknown proteins are fragmented into short sequences, called "peptides" which are measured according to mass and charge. Anderson et al. [Anderson et al., 2003] improved the determination of the peptides from data bases by using SVMs.

In above Bioinformatics applications support vector machines improved previous results. Therefore SVMs are one of the standard techniques in Bioinformatics. Further applications of SVMs are given under

`http://www.clopinet.com/isabelle/Projects/SVM/applist.html`.

SVM web-sites with tutorials and software are

- `http://www.kernel-machines.org`,
- `http://www.support-vector-machines.org/SVM_stat.html`, and
- `http://kernelsvm.tripod.com`.

4.2 Linearly Separable Problems

First we consider classification tasks which are linearly separable.

As in previous chapter we assume that objects $x \in \mathcal{X}$ from an object set $\mathcal{X}$ are represented or described by feature vectors $\boldsymbol{x} \in \mathbb{R}^d$.

The training set consists of l objects $X = \left\{x^1, \ldots, x^l\right\}$ with a label $y^i \in \{-1, 1\}$ for classification and $y^i \in \mathbb{R}$ for regression. The matrix of feature vectors is $\boldsymbol{X} = \left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\right)$ and the vector of labels $\boldsymbol{y} = \left(y^1, \ldots, y^l\right)^T$.

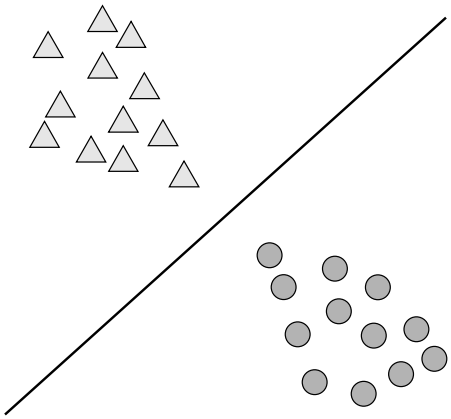


Figure 4.1: A linearly separable problem, where data points of class 1 (the circles) are on one side of the hyperplane (a line in two dimensions) and data points of class 2 (the triangles) are on the other side of the hyperplane.

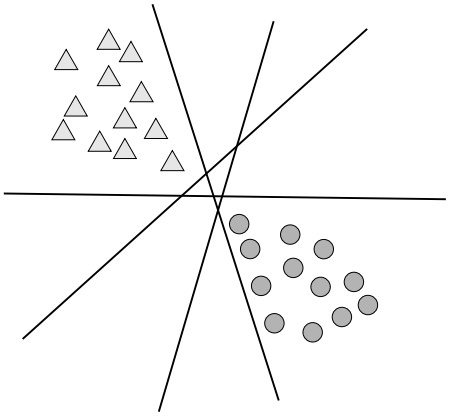


Figure 4.2: Different solutions for linearly separating the the classes.

Linear separable means, for the training data a discriminant function $\boldsymbol{w}^T\boldsymbol{x} + b$ exists which defines a classification function sign $\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$ for which

$$y^i = \text{sign}\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right),\ 1 \leq i \leq l\ . \tag{4.1}$$

That means all positive examples are of one side of the boundary function $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ and all negative examples are on the other side. Fig. 4.1 shows a two-dimensional example for a linear separable problem.

However the data can be separated by different hyperplanes as shown in Fig. 4.2 for the two-dimensional example. Which is the best separating hyperplane?

Our theoretical considerations in last chapter suggest to use the classification function with the lowest complexity. For the complexity measure we will use the margin. Fig. 4.3 gives an intuition why larger margin is desirable as data points can be noise without jumping over the boundary function.

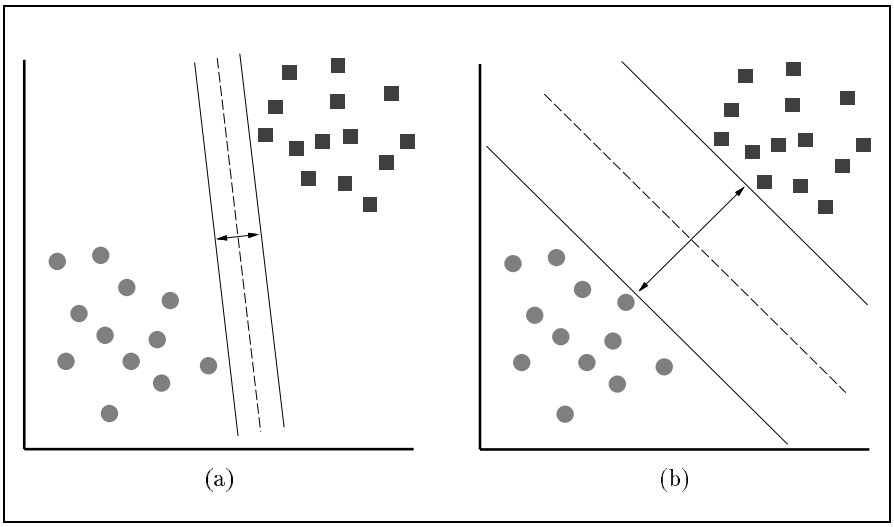


Figure 4.3: Intuitively, better generalization is expected from separation on the right hand side than from the left hand side because larger distance to the data points allows for noisy data. Copyright © 1997 [Osuna et al., 1997].

4.3 Linear SVM

We are looking for the separating hyperplane with the largest margin, because it has the tightest bound on the VC-dimension.

The *margin* was defined in Section 3.6.5 as the minimal distance of the boundary function $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ to all data points. As in Section 3.6.5 we assume that the classification function is described by a discrimination function in the canonical form (remember that scaling $\boldsymbol{w}$ and b with the same factor does not change the classification function), that means

$$\min_{i=1,\ldots,l} \left|\boldsymbol{w}^T\boldsymbol{x}^i\right| = 1\,. \tag{4.2}$$

In Section 3.6.5 we saw that optimizing b in order to obtain the smallest $\boldsymbol{w}$ of vectors having the same directions as $\boldsymbol{w}$ leads to at least one point exists for which $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and at least one point exists for which $\boldsymbol{w}^T\boldsymbol{x} + b = -1$. We then showed in Section 3.6.5 that the margin is given by

$$\gamma = \frac{1}{\|\boldsymbol{w}\|}\,. \tag{4.3}$$

The situation is depicted in Fig. 4.4.

In Section 3.6.5 we optimized b and the length of $\boldsymbol{w}$ but not the direction of $\boldsymbol{w}$. We will to this now.

If we assume correct classification and the canonical form then

$$\boldsymbol{w}^T\boldsymbol{x}^i + b \geq 1 \text{ for } y^i = 1 \text{ and} \tag{4.4}$$
$$\boldsymbol{w}^T\boldsymbol{x}^i + b \leq -1 \text{ for } y^i = -1\,. \tag{4.5}$$

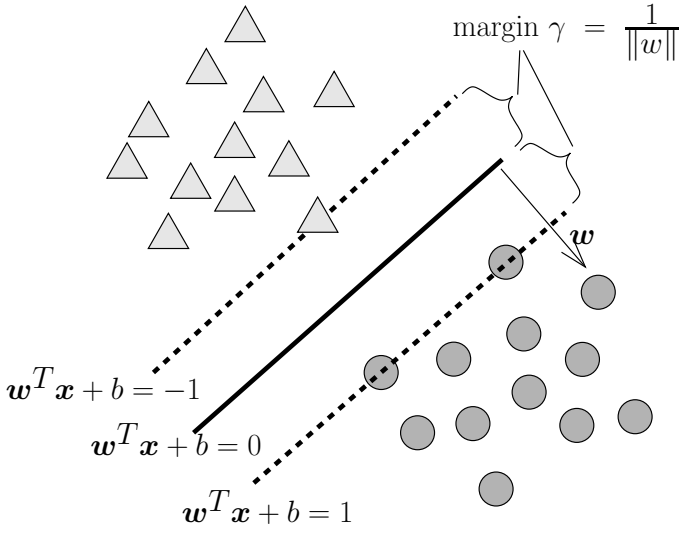


Figure 4.4: For the hyperplane described by the canonical discriminant function and for the optimal offset b (same distance to class 1 and class 2), the margin is $\gamma = \frac{1}{\|\boldsymbol{w}\|}$. At least one point exists for which $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and at least one point exists for which $\boldsymbol{w}^T\boldsymbol{x} + b = -1$.

The values 1 and -1 are due to the canonical form.

To maximize the margin γ, we have to maximize $\frac{1}{\|\boldsymbol{w}\|}$ which means we have to minimize $\|\boldsymbol{w}\|$ or equivalently $\boldsymbol{w}^T\boldsymbol{w} = \|\boldsymbol{w}\|^2$.

To realize the structural risk minimization principle we maximize the margin but ensure correct classification. This leads to the support vector optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & \boldsymbol{w}^T\boldsymbol{x}^i + b \geq 1 \text{ for } y^i = 1 \\ & \boldsymbol{w}^T\boldsymbol{x}^i + b \leq -1 \text{ for } y^i = -1\,. \end{aligned} \tag{4.6}$$

This can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) \geq 1\,. \end{aligned} \tag{4.7}$$

This optimization problem is well defined because $\boldsymbol{w}^T\boldsymbol{w} = \|\boldsymbol{w}\|^2$ is positive definite and the constraints are linear. The problem is a convex quadratic optimization task with has one unique solution.

Above formulation is in the primal space. If we use Lagrange multipliers then we can solve this problem in the dual space.

The Lagrange function is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) \;=\; \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \sum_{i=1}^{l} \alpha_i \;\left(y^i \;\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;-\; 1\right)\;, \tag{4.8}$$

where $\alpha_i \;\geq\; 0$ are Lagrange multipliers which are larger than zero because the constraints are inequalities.

The solution of the optimization problem is a saddle point of the Lagrangian. To find the saddle point the minimization has to be done over $\boldsymbol{w}$ and b and the maximization over $\alpha_i \;\geq\; 0$.

For the minimum the derivatives of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})$ are zero:

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} \;=\; \boldsymbol{w} \;-\; \sum_{i=1}^{l} \alpha_i \; y^i \; \boldsymbol{x}^i \;=\; \mathbf{0} \;\text{ and} \tag{4.9}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} \;=\; \sum_{i=1}^{l} \alpha_i \; y^i \;=\; 0\;. \tag{4.10}$$

We can solve the first equation for $\boldsymbol{w}$ and obtain

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l} \alpha_i \; y^i \; \boldsymbol{x}^i\;. \tag{4.11}$$

And the second equation gives an equality constraint

$$\sum_{i=1}^{l} \alpha_i \; y^i \;=\; 0\;. \tag{4.12}$$

The expression for $\boldsymbol{w}$ in eq. (4.11) can be substituted into the Lagrangian:

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) \;=\; \frac{1}{2}\;\left(\sum_{i=1}^{l} \alpha_i \; y^i \;\left(\boldsymbol{x}^i\right)^T\right)\left(\sum_{i=1}^{l} \alpha_i \; y^i \; \boldsymbol{x}^i\right) \;- \\
&\sum_{i=1}^{l} \alpha_i \; y^i \left(\sum_{j=1}^{l} \alpha_j \; y^j \;\left(\boldsymbol{x}^j\right)^T\right) \boldsymbol{x}^i \;-\; b \sum_{i=1}^{l} \alpha_i \; y^i \;+\; \sum_{i=1}^{l} \alpha_i \;= \\
&\frac{1}{2} \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \;\left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \;-\; \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \;\left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \;+\; \sum_{i=1}^{l} \alpha_i \;= \\
&-\frac{1}{2} \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \;\left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \;+\; \sum_{i=1}^{l} \alpha_i\;.
\end{aligned} \tag{4.13}$$

The Lagrangian has to be maximized with respect to the dual variables, therefore we can minimize the negative Lagrangian

$$\frac{1}{2} \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \;\left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \;-\; \sum_{i=1}^{l} \alpha_i\;. \tag{4.14}$$

The dual formulation is of the optimization problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \, - \, \sum_{i=1}^{l} \alpha_i \\ \text{s.t.} \quad & \alpha_i \, \geq \, 0 \\ & \sum_{i=1}^{l} \alpha_i \, y^i \, = \, 0 \, . \end{aligned} \tag{4.15}$$

The dual formulation has only box constraints and one linear equality constraint and is therefore simpler to solve than the primal formulation.

The dual formulation in vector notation is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \, \boldsymbol{\alpha}^T \boldsymbol{Y}^T \boldsymbol{X}^T \boldsymbol{X} \, \boldsymbol{Y} \, \boldsymbol{\alpha} \, - \, \mathbf{1}^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{\alpha} \, \geq \, \mathbf{0} \\ & \boldsymbol{\alpha}^T \boldsymbol{y} \, = \, 0 \, , \end{aligned} \tag{4.16}$$

where $\boldsymbol{Y} \, = \, \mathrm{diag}(\boldsymbol{y})$ is the diagonal matrix of the labels.

We can now solve the dual formulation to obtain the optimal vector $\boldsymbol{\alpha}$. From the primal $\boldsymbol{\alpha}$ the dual vector $\boldsymbol{w}$ can be computed via

$$\boldsymbol{w} \, = \, \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \, . \tag{4.17}$$

For classification we do not need an explicit representation of $\boldsymbol{w}$ but only the dot products between the new vector $\boldsymbol{x}$ and the vectors $\boldsymbol{x}^i$:

$$\boldsymbol{w}^T \boldsymbol{x} \, + b \, = \, \sum_{i=1}^{l} \alpha_i \, y^i \, \left(\boldsymbol{x}^i\right)^T \boldsymbol{x} \, + \, b \, . \tag{4.18}$$

The Karush-Kuhn-Tucker (KKT) conditions require that the product of Lagrange multipliers and constraint is zero for the optimal $\boldsymbol{w}$ and $\boldsymbol{b}$:

$$\alpha_i \, \left(y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \, + \, b\right) \, - \, 1\right) \, = \, 0 \tag{4.19}$$

It follows that either

$$\alpha_i \, = \, 0 \text{ or} \tag{4.20}$$

$$y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \, + \, b\right) \, = \, 1 \, . \tag{4.21}$$

The $\boldsymbol{x}^i$ for which $\alpha_i \, > \, 0$ are called *support vectors*.

That means the $\boldsymbol{w}$ is only described by support vectors because the terms with $\alpha_i = 0$ vanish.

The value of b can be determined from each support $\boldsymbol{x}^i$ vector because

$$y^i \ \left(\boldsymbol{w}^T \boldsymbol{x}^i \ + \ b\right) \ = \ 1 \tag{4.22}$$

therefore

$$b \ = \ y^i \ - \ \boldsymbol{w}^T \boldsymbol{x}^i \ . \tag{4.23}$$

If the solutions $\boldsymbol{\alpha}$ are not exact then b can be computed by averaging the computed b values over all support vectors.

Note that for the optimal $\boldsymbol{\alpha}$

$$b \sum_{i=1}^{l} \alpha_i \ y^i \ = \ 0 \tag{4.24}$$

holds. We have for the optimal solution

$$\begin{aligned} \boldsymbol{w}^T \boldsymbol{w} \ &= \ \sum_{i=1}^{l} \alpha_i \ y^i \ \boldsymbol{w}^T \boldsymbol{x}^i \ = \\ \sum_{i=1}^{l} \alpha_i \ y^i \ \boldsymbol{w}^T \boldsymbol{x}^i \ &+ \ b \sum_{i=1}^{l} \alpha_i \ y^i \ - \ \sum_{i=1}^{l} \alpha_i \ + \ \sum_{i=1}^{l} \alpha_i \ = \\ \sum_{i=1}^{l} \alpha_i \ \left(y^i \ \left(\boldsymbol{w}^T \boldsymbol{x}^i \ + \ b\right) \ - \ 1\right) \ &+ \ \sum_{i=1}^{l} \alpha_i \ = \\ \sum_{i=1}^{l} \alpha_i \ , \end{aligned} \tag{4.25}$$

where we used the KKT conditions eq. (4.19).

The margin γ can be expressed by support vector weights α_i for the optimal solution:

$$\gamma \ = \ \frac{1}{\sqrt{\sum_{i=1}^{l} \alpha_i}} \ . \tag{4.26}$$

4.4 Linear SVM for Non-Linear Separable Problems

In previous section we assumed that the classification problem is linearly separable. However in many tasks this assumption is not true. See Fig. 4.6 for a non-linear separable task.

In Fig. 4.7 on the top line two problems are depicted which are not linearly separable. However if data points can be moved then these problems are again linear separable.

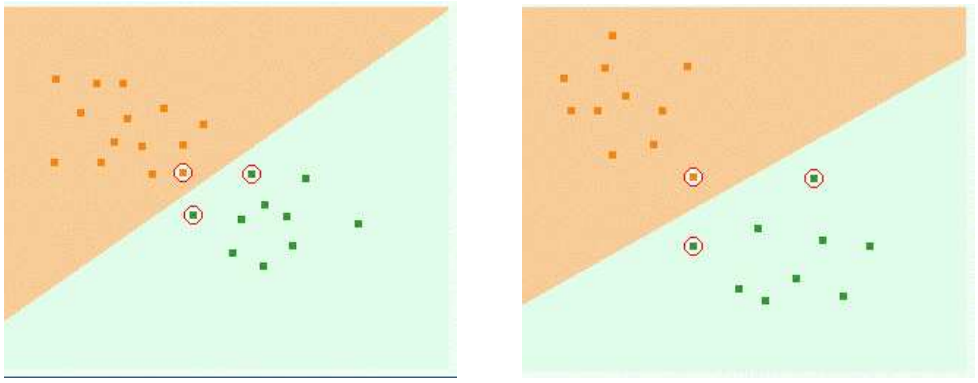


Figure 4.5: Two examples for linear SVMs. The class of points is given by their color. The circled points are support vectors and the color of the regions correspond to the class points in this region would be classified. Screen-shots from `http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSVM.html`.

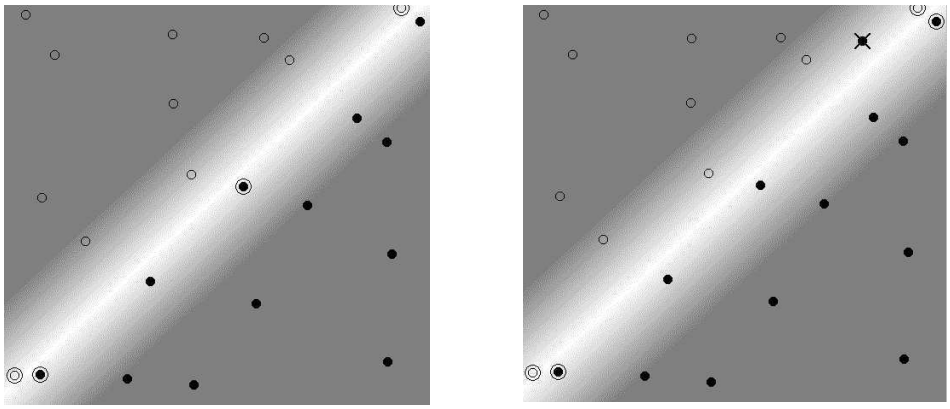


Figure 4.6: Left: linear separable task. Right: a task which is not linearly separable, where the filled point marked with a cross is in the region of the non-filled points. Circled points are support vectors. Copyright © 1998 [Burges, 1998].

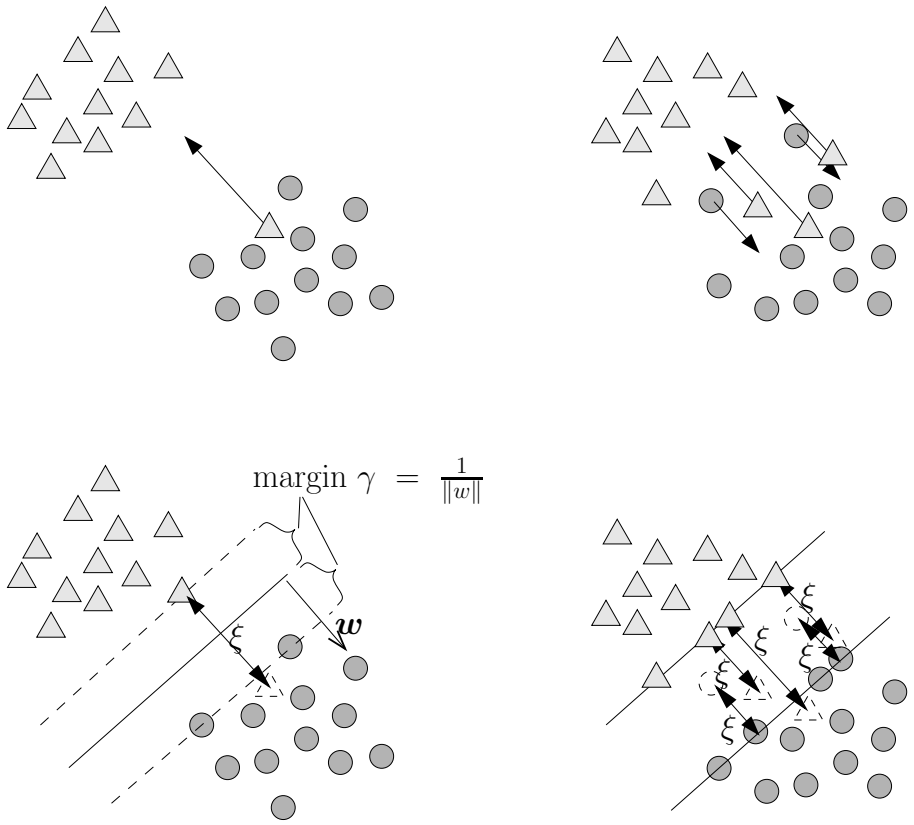


Figure 4.7: Two problems at the top line which are not linearly separable. If data points can be moved then the problem would be linear separable as shown in the bottom line.

What does moving mean? If the data point is moved orthogonal to a separating hyperplane in the direction of its class. With $\delta_i > 0$ we can express that by

$$\left(\boldsymbol{x}^i\right)^{\text{moved}} \;=\; \boldsymbol{x}^i \;+\; y^i\,\delta_i\,\boldsymbol{w} \tag{4.27}$$

$$\boldsymbol{w}^T\left(\boldsymbol{x}^i\right)^{\text{moved}} \;=\; \boldsymbol{w}^T\boldsymbol{x}^i \;+\; y^i\,\delta_i\,\|\boldsymbol{w}\|^2 \tag{4.28}$$

and obtain

$$y^i\left(\boldsymbol{w}^T\left(\boldsymbol{x}^i\right)^{\text{moved}} \;+\; b\right) \;\geq\; 1 \tag{4.29}$$

$$\Leftrightarrow\;\; y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \delta_i\,\|\boldsymbol{w}\|^2\,. \tag{4.30}$$

If we set $\xi_i = \delta_i\,\|\boldsymbol{w}\|^2$ then we obtain

$$y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i\,. \tag{4.31}$$

The new introduced variables ξ_i are called *slack variables*.

The movements expressed by the slack variables should be minimized. How strong movements are penalized is expressed by a new hyper-parameter C.

We obtain a optimization problem with slack variables as

$$\begin{aligned}
\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; C\,\sum_{i=1}^{l}\xi_i \\
\text{s.t.} \quad & y^i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i \\
& \xi_i \;\geq\; 0\,.
\end{aligned} \tag{4.32}$$

The Lagrange function is

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\xi},\boldsymbol{\mu}) \;=\; & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; C\,\sum_{i=1}^{l}\xi_i \;- \\
& \sum_{i=1}^{l}\alpha_i\;\left(y^i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;-\; 1 \;+\; \xi_i\right) \;-\; \sum_{i=1}^{l}\mu_i\,\xi_i\,,
\end{aligned} \tag{4.33}$$

where $\alpha_i, \mu_i \;\geq\; 0$ are Lagrange multipliers.

At the minimum the derivatives of $\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\xi},\boldsymbol{\mu})$ are zero:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} \;=\; \boldsymbol{w} \;-\; \sum_{i=1}^{l}\alpha_i\,y^i\,\boldsymbol{x}^i \;=\; \mathbf{0} \tag{4.34}$$

$$\frac{\partial\mathcal{L}}{\partial b} \;=\; \sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0 \tag{4.35}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} \;=\; \mathbf{1}C \;-\; \boldsymbol{\alpha} \;-\; \boldsymbol{\mu} \;=\; \mathbf{0}\,. \tag{4.36}$$

Again be obtain

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l} \alpha_i \; y^i \; \boldsymbol{x}^i \; . \tag{4.37}$$

and

$$\sum_{i=1}^{l} \alpha_i \; y^i \;=\; 0 \; . \tag{4.38}$$

Additionally, from the last derivative we obtain

$$\alpha_i \;\leq\; C \tag{4.39}$$

because $\mu_i \;\geq\; 0$.

The expression for $\boldsymbol{w}$ can be again substituted into the Lagrangian:

$$\begin{aligned}
\mathcal{L} \;=\; & \frac{1}{2} \left(\sum_{i=1}^{l} \alpha_i \; y^i \; \left(\boldsymbol{x}^i \right)^T \right) \left(\sum_{i=1}^{l} \alpha_i \; y^i \; \boldsymbol{x}^i \right) \;+\; C \; \sum_{i=1}^{l} \xi_i \;- \\
& \sum_{i=1}^{l} \alpha_i \; y^i \left(\sum_{j=1}^{l} \alpha_j \; y^j \; \left(\boldsymbol{x}^j \right)^T \right) \boldsymbol{x}^i \;-\; b \; \sum_{i=1}^{l} \alpha_i \; y^i \;+\; \sum_{i=1}^{l} \alpha_i \;- \\
& \sum_{i=1}^{l} \alpha_i \; \xi_i \;+\; \sum_{i=1}^{l} \alpha_i \;-\; \sum_{i=1}^{l} \mu_i \; \xi_i \;= \\
& -\; \frac{1}{2} \; \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \;+\; \sum_{i=1}^{l} \xi_i \left(C \;-\; \alpha_i \;-\; \mu_i \right) \;+\; \sum_{i=1}^{l} \alpha_i \;= \\
& -\; \frac{1}{2} \; \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \;+\; \sum_{i=1}^{l} \alpha_i \; .
\end{aligned} \tag{4.40}$$

We again minimize the negative Lagrangian

$$\frac{1}{2} \; \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \;-\; \sum_{i=1}^{l} \alpha_i \; . \tag{4.41}$$

The dual formulation is of the optimization problem with slack variables is

$$\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \; \sum_{i,j} \alpha_i \; \alpha_j \; y^i \; y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \;-\; \sum_{i=1}^{l} \alpha_i \\
\text{s.t.} \quad & 0 \;\leq\; \alpha_i \;\leq\; C \\
& \sum_{i=1}^{l} \alpha_i \; y^i \;=\; 0 \; .
\end{aligned} \tag{4.42}$$

The dual formulation with slack variables differs from the dual without slack variables in eq. (4.42) only through the upper bound C on α_i.

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha}} & \frac{1}{2}\,\boldsymbol{\alpha}^T \boldsymbol{Y}^T \boldsymbol{X}^T \boldsymbol{X}\,\boldsymbol{Y}\,\boldsymbol{\alpha} \;-\; \mathbf{1}^T \boldsymbol{\alpha} \\ \text{s.t.} & \mathbf{0} \;\leq\; \boldsymbol{\alpha} \;\leq\; C\mathbf{1} \\ & \boldsymbol{\alpha}^T \boldsymbol{y} \;=\; 0\,, \end{array} \tag{4.43}$$

where $\boldsymbol{Y} \;=\; \mathrm{diag}(\boldsymbol{y})$ is the diagonal matrix of the labels.

The Karush-Kuhn-Tucker (KKT) conditions require:

$$\alpha_i \,\left(y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;-\; 1 \;+\; \xi_i\right) \;=\; 0 \tag{4.44}$$

$$\mu_i \,\xi_i \;=\; 0\,. \tag{4.45}$$

From the last equality it follows that if $\xi_i \;>\; 0$ then $\mu_i \;=\; 0$. Because of the condition $C \;-\; \alpha_i \;-\; \mu_i \;=\; 0$ (derivative of Lagrangian with respect to ξ_i is zero) we obtain $\alpha_i = C$ and therefore $\xi_i \;=\; 1 \;-\; y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right)$.

If $\mu_i \;>\; 0$ then $\xi_i \;=\; 0$ and either $y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;=\; 1$ or $\alpha_i \;=\; 0$.

From $\alpha_i = 0$ follows that $\mu_i = C$ and $\xi_i = 0$ therefore

That means

$$\alpha_i > 0: \tag{4.46}$$

$$\Rightarrow \begin{cases} \xi_i \;=\; y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;-\; 1 \;>\; 0 & \text{and} \quad \alpha_i = C \\ y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;=\; 1 & \text{and} \quad 0 \;<\; \alpha_i \;<\; C \end{cases}$$

$$\alpha_i = 0: \tag{4.47}$$

$$\Rightarrow \xi_i \;=\; 0 \text{ and } y^i \,\left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;>\; 1\,.$$

Data points which are moved ($\xi_i \;>\; 0$) have $\alpha_i \;=\; C$ and data points on the boundary have $0 \;<\; \alpha_i \;<\; C$. The data points classified correctly with absolute discriminant functions value larger than 1 have $\alpha_i \;=\; 0$.

kind of data point	α_i	ξ_i
$\boldsymbol{x}^i$ moved	$\alpha_i \;=\; C$	$\xi_i \;>\; 0$
$\boldsymbol{x}^i$ on the boundary	$0 \;<\; \alpha_i \;<\; C$	$\xi_i \;=\; 0$
$\boldsymbol{x}^i$ classified correctly	$\alpha_i \;=\; 0$	$\xi_i \;=\; 0$

The vectors $\boldsymbol{x}^i$ with $\alpha_i \;>\; 0$ are the *support vectors*. The situations is depicted in Fig. 4.8.

Again only support vectors determine the discriminant function. Data points with $\alpha \;=\; 0$ do not influence the solution – even if they would be removed from the training set, the results remains the same.

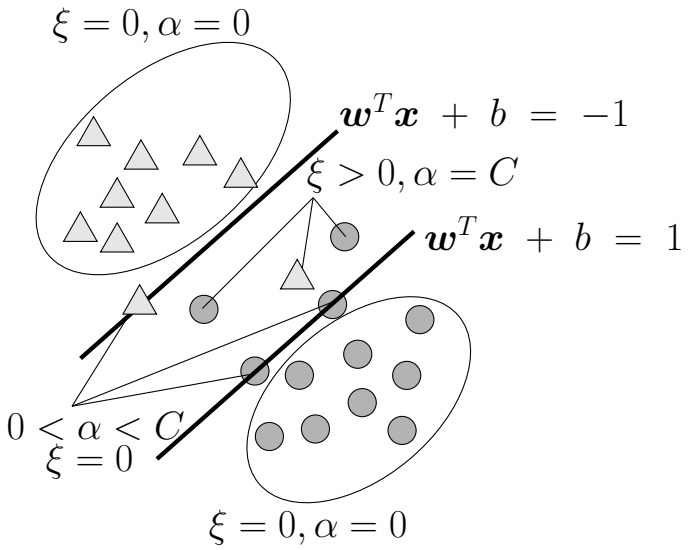


Figure 4.8: Typical situation for the C-SVM. The slack variables for data points within the boundaries $\boldsymbol{w}^T\boldsymbol{x}^i + b = 1$ and $\boldsymbol{w}^T\boldsymbol{x}^i + b = -1$ are $\xi > 0$ with $\alpha = C$, for data points on the boundaries $\xi = 0,\ 0 < \alpha < C$, and for all other data points $\xi = 0,\ alpha = 0$. Support vectors are data points with $\alpha > 0$.

4.5 Average Error Bounds for SVMs

The following theorems are proved by using the Leave-One-Out Cross Validation (LOO CV) estimator which was shown in Subsection 3.2.2.2 to be almost unbiased.

The complexity of the SVM is described by the margin which in turn can be expressed through the support vector weights $\alpha_i > 0$.

Essential support vectors are the support vectors for which the solution changes if they are removed from the training set. Fig. 4.9 shows essential and non-essential support vectors for a two-dimensional problem.

We denote the number of essential support vectors by k_l and r_l the radius of the sphere which contains all essential support vectors.

First we note that $k_l \leq d+1$ (because $(d+1)$ points in general position are enough to define a hyperplane which has equal distance to all points).

Now we can give bounds for the expected risk $ER(g(.;\boldsymbol{w}_l))$, where the expectation is taken over the training set of size l and a test data point.

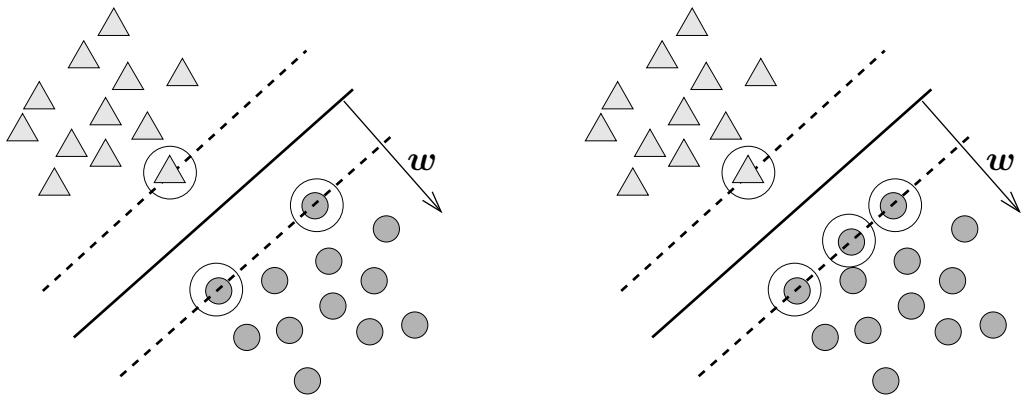


Figure 4.9: Essential vectors. Left: all support vectors (the circled points) are essential. Right: the three support vectors on the right hand side line are not essential because removing one of them does not change the solution.

Theorem 4.1 (Average Bounds for SVMs) *For the expected risk with above definitions*

$$ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{Ek_{l+1}}{l\;+\;1} \tag{4.48}$$

$$ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{d+1}{l\;+\;1} \tag{4.49}$$

$$ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{E\left(\frac{r_{l+1}}{\gamma_{l+1}}\right)^2}{l\;+\;1} \tag{4.50}$$

$$ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{E\min\left\{k_{l+1}\;,\;\left(\frac{r_{l+1}}{\gamma_{l+1}}\right)^2\right\}}{l\;+\;1} \tag{4.51}$$

$$ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{E\left(\left(k^*_{l+1}\right)^2\sum_{i^*}\alpha_{i^*}\;+\;m\right)}{l\;+\;1} \tag{4.52}$$

$$C \leq r_l^{-2}: \;\; ER(g(.;\boldsymbol{w}_l)) \;\leq\; \frac{E\left((k_{l+1})^2\sum_i \alpha_i\right)}{l\;+\;1}\,, \tag{4.53}$$

where i^* are the support vectors with $0 \;<\; \alpha_{i^*} \;<\; C$ and m is the number of support vectors with $\alpha_i \;=\; C$.

All of above bounds are based on the leave-one-out cross validation estimator and its property to be almost unbiased.

It is important to note that we obtain for the expected risk a bound of the order $\frac{1}{l}$ whereas we saw in the theoretical sections bounds (worst case bounds) for the risk of $\frac{1}{\sqrt{l}}$.

Important to know would be the variance of the expected risk.

4.6 ν-SVM

The C-support vector machine can be reformulated in a more convenient form. Instead of the hyper-parameter C another hyper-parameter ν can be introduced which has a more intuitive interpretation.

The hyper-parameter ν will be an upper bound on the fraction of margin errors and a lower bound on the number of support vectors.

The primal formulation of the ν-SVM is

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \nu\,\rho \;+\; \frac{1}{l}\,\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; \rho \;-\; \xi_i \\ & \xi_i \;\geq\; 0 \text{ and } \rho \;\geq\; 0\,. \end{aligned} \tag{4.54}$$

Without the slack variables the margin would be scaled by ρ. Therefore the margin is maximized in the objective because it enters it with a negatively. However the margin can only be maximized as long as the constraints are fulfilled.

To see the properties of this formulation we need the *margin error*, which is the fraction of data points for which

$$y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;<\; \rho\,. \tag{4.55}$$

Theorem 4.2 (ν-SVM Properties)
For the solution of the ν-SVM optimization problem eq. (4.54) holds:
(i) ν is an upper bound on the fraction of margin errors.
(ii) ν is a lower bound on the fraction of support vectors.
(iii) Under mild conditions (see [Schölkopf and Smola, 2002]) ν asymptotically reaches the fraction of support vectors and the fraction of margin errors with probability 1.

The Lagrangian of the ν-SVM formulation is

$$\begin{aligned} \mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\xi},\boldsymbol{\mu},\rho,\delta) \;=\; & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \nu\,\rho \;+\; \frac{1}{l}\,\sum_{i=1}^{l}\xi_i \;- \\ & \sum_{i=1}^{l}\alpha_i\,\left(y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;-\; \rho \;+\; \xi_i\right) \;-\; \sum_{i=1}^{l}\mu_i\,\xi_i \;-\; \delta\,\rho\,. \end{aligned} \tag{4.56}$$

At the minimum the derivatives of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu}, \rho, \delta)$ are zero:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \;=\; \boldsymbol{w} \;-\; \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \;=\; \mathbf{0} \tag{4.57}$$

$$\frac{\partial \mathcal{L}}{\partial b} \;=\; \sum_{i=1}^{l} \alpha_i \, y^i \;=\; 0 \tag{4.58}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} \;=\; \frac{1}{l}\mathbf{1} \;-\; \boldsymbol{\alpha} \;-\; \boldsymbol{\mu} \;=\; \mathbf{0} \tag{4.59}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} \;=\; \sum_{i=1}^{l} \alpha_i \;-\; \delta \;-\; \nu \;=\; 0\,. \tag{4.60}$$

If we substitute these values into the Lagrangian we obtain

$$\mathcal{L} \;=\; -\,\frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i\,. \tag{4.61}$$

The dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \\ \text{s.t.} \quad & 0 \;\leq\; \alpha_i \;\leq\; \frac{1}{l} \\ & \sum_{i=1}^{l} \alpha_i \, y^i \;=\; 0 \\ & \sum_{i=1}^{l} \alpha_i \;\geq\; \nu\,. \end{aligned} \tag{4.62}$$

In contrast to the C-SVM formulation the term $\sum_{i=1}^{l} \alpha_i$ does not appear in the objective function but it is lower bounded by the constant ν. This also avoids the trivial solution $\boldsymbol{\alpha} \;=\; \mathbf{0}$.

The Karush-Kuhn-Tucker (KKT) conditions require:

$$\alpha_i \, \left(y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;-\; \rho \;+\; \xi_i\right) \;=\; 0 \tag{4.63}$$

$$\mu_i \, \xi_i \;=\; 0 \tag{4.64}$$

$$\delta \, \rho \;=\; 0\,. \tag{4.65}$$

The last equation together with eq. (4.60) immediately leads to the fact that for solutions with margin larger than zero $\sum_{i=1}^{l} \alpha_i \;=\; \nu$.

b and ρ can be computed from support vectors with $0 \;<\; \alpha_i \;<\; \frac{1}{l}$ because then $\mu_i > 0$ and $\xi_i = 0$ therefore $y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b\right) \;=\; \rho$. For two such support vectors $\boldsymbol{x}^1$ with $y^1 = 1$ and $\boldsymbol{x}^2$

with $y^2 = -1$ we obtain

$$\rho = \frac{1}{2}\left(\boldsymbol{w}^T\boldsymbol{x}^1 - \boldsymbol{w}^T\boldsymbol{x}^2\right) \tag{4.66}$$

$$b = -\frac{1}{2}\left(\boldsymbol{w}^T\boldsymbol{x}^1 + \boldsymbol{w}^T\boldsymbol{x}^2\right) \tag{4.67}$$

The first equation is obtained by adding the two equations and the second by subtracting the two equations. Analogously the values can be computed for more than two support vectors with $0 < \alpha_i < \frac{1}{l}$.

The constraints $\alpha_i \leq \frac{1}{l}$, $\sum_{i=1}^{l} \alpha_i = \nu$, and $\sum_{i=1}^{l} \alpha_i\, y^i = 0$ impose constraints on the ν.

First it is clear that $0 \leq \nu \leq 1$.

Let us assume that we have $k < \frac{l}{2}$ examples with $y^i = 1$ and $(l-k)$ examples with $y^i = -1$, i.e. we have fewer positive examples than negative. We are now looking for the maximal ν. It is optimal if all k positive examples have maximal value of $\alpha_i = \frac{1}{l}$ giving in sum $\frac{k}{l}$. The negative examples sum also up to $\frac{k}{l}$ in order to fulfill $\sum_{i=1}^{l} \alpha_i\, y^i = 0$. Therefore $\sum_{i=1}^{l} \alpha_i = \frac{2\,k}{l}$. We obtain that

$$\nu \leq \frac{2\,k}{l}\,. \tag{4.68}$$

If $k \geq \frac{l}{2}$, we can apply the above argumentation and obtain

$$\nu \leq \frac{2\,(l-k)}{l}\,. \tag{4.69}$$

Merging these two cases, we obtain

$$\nu \leq \frac{2\,\min\{k, (l-k)\}}{l}\,. \tag{4.70}$$

The same holds if the number of negative examples are smaller than the positives.

Because ν is an upper bound on the fraction of margin errors, for very unbalanced data sets (small k) the ν-SVM does not allow many errors of the larger set. However for some tasks many errors of the larger set is the best solution.

ν-SVM may have problems with unbalanced data sets.

Next we will investigate the connection between the C-SVM and the ν-SVM.

Let us assume, that we have a ν-SVM solution with a specific $\rho > 0$ ($\rho = 0$ is excluded) and the variables $\boldsymbol{w}$, b, and $\boldsymbol{\xi}$. We define new variables as

$$\boldsymbol{w}^* = \boldsymbol{w}/\rho \tag{4.71}$$

$$b^* = b/\rho \tag{4.72}$$

$$\boldsymbol{\xi}^* = \boldsymbol{\xi}/\rho \tag{4.73}$$

and we obtain as primal for the ν-SVM with the new variables:

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*,\rho} \quad & \frac{1}{2}\,\rho^2\,\|\boldsymbol{w}^*\|^2 \;-\; \nu\,\rho \;+\; \rho\,\frac{1}{l}\,\sum_{i=1}^{l}\xi_i^* \\ \text{s.t.} \quad & \rho\,y^i\,\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; \rho \;-\; \rho\,\xi_i^* \\ & \rho\,\xi_i^* \;\geq\; 0 \text{ and } \rho \;\geq\; 0 \end{aligned} \tag{4.74}$$

that is

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*,\rho} \quad & \rho^2\,\left(\frac{1}{2}\,\|\boldsymbol{w}^*\|^2 \;-\; \frac{\nu}{\rho} \;+\; \frac{1}{l\,\rho}\,\sum_{i=1}^{l}\xi_i^*\right) \\ \text{s.t.} \quad & y^i\,\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; 1 \;-\; \xi_i^* \\ & \xi_i^* \;\geq\; 0 \text{ and } \rho \;\geq\; 0\,. \end{aligned} \tag{4.75}$$

Because the variables are only rescaled, this formulation is equivalent to the original one.

If we now fix the optimal ρ of the ν-SVM solution and define $C \;=\; \frac{1}{l\,\rho}$ then the factor ρ^2 is constant as is the additive term $\frac{\nu}{\rho}$ and we obtain the C-SVM formulation:

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*} \quad & \frac{1}{2}\,\|\boldsymbol{w}^*\|^2 \;+\; C\,\sum_{i=1}^{l}\xi_i^* \\ \text{s.t.} \quad & y^i\,\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; 1 \;-\; \xi_i^* \\ & \xi_i^* \;\geq\; 0\,. \end{aligned} \tag{4.76}$$

That means for each ν-SVM solution with $\rho > 0$ (depending on ν) there exists a C which gives the same solution in the C-SVM formulation.

Basically the ν-SVM is more convenient for hyper-parameter selection, i.e. the selection of ν compared to the selection of C in the C-SVM. The hyper-parameter selection is also more robust.

However for unbalanced data sets the C-SVM is to prefer because only small values of ν are possible.

4.7 Non-Linear SVM and the Kernel Trick

Until now we only considered linear models but for some problems nonlinear models are more appropriate.

We want to apply the nice results obtained from the linear theory. The idea is to map the feature vectors $\boldsymbol{x}$ by a nonlinear function $\boldsymbol{\Phi}$ into a feature space:

$$\boldsymbol{\Phi}:\;\; \mathbb{R}^d \to \mathbb{R}^{d_\Phi},\; \boldsymbol{x} \;\mapsto\; \boldsymbol{x}_\phi,\; \boldsymbol{x}_\phi \;=\; \boldsymbol{\Phi}(\boldsymbol{x})\,. \tag{4.77}$$

In this feature space we can apply the linear theory of SVMs. Afterwards we can project the results back into the original space. Fig. 4.10 depicts the feature mapping. In the feature space the

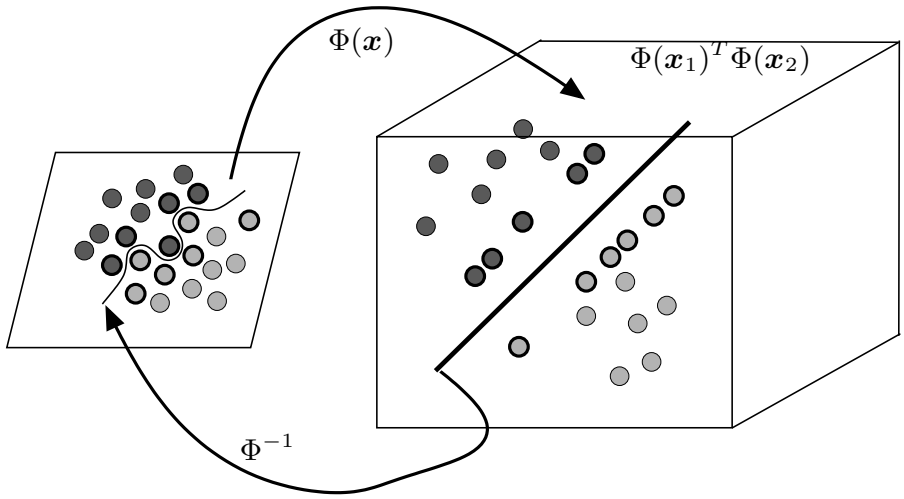


Figure 4.10: Nonlinearly separable data is mapped into a feature space where the data is linear separable. The support vectors in feature space are marked by thicker borders. These vectors as well as the boundary function are shown in the original space where the linear boundary function becomes a nonlinear boundary function.

data is assumed to be linear separable (the VC-dimension of linear functions increases with the dimensionality). The result can be transferred back into the original space.

For example consider

$$\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \left(x_1^2 \,,\; x_2^2 \,,\; \sqrt{2}\, x_1 \, x_2\right) \,. \tag{4.78}$$

This is a mapping from a two-dimensional space into a three-dimensional space.

The four data points $\boldsymbol{x}^1 \;=\; (1,1), \boldsymbol{x}^2 \;=\; (1,-1), \boldsymbol{x}^3 \;=\; (-1,1), \boldsymbol{x}^4 \;=\; (-1,-1)$ with labels $y^1 = -1, y^2 = 1, y^3 = 1, y^4 = -1$ are not separable in the two-dimensional space.

Their images are

$$\begin{aligned}
\boldsymbol{\Phi}(\boldsymbol{x}^1) \;&=\; (1,1,\sqrt{2}) \\
\boldsymbol{\Phi}(\boldsymbol{x}^2) \;&=\; (1,1,-\sqrt{2}) \\
\boldsymbol{\Phi}(\boldsymbol{x}^3) \;&=\; (1,1,-\sqrt{2}) \\
\boldsymbol{\Phi}(\boldsymbol{x}^4) \;&=\; (1,1,\sqrt{2}) \,,
\end{aligned}$$

which are linearly separable in the three-dimensional space. Fig. 4.11 shows the mapping into the three-dimensional space.

We write in the following x_{mn} for $(\boldsymbol{x}^m)_n$, the j-th component of the vector $\boldsymbol{x}^i$.

The dot product in the three-dimensional space is

$$\begin{aligned}
&\boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}^j) \;=\; x_{i1}^2 \, x_{j1}^2 \;+\; x_{i2}^2 \, x_{j2}^2 \;+\; 2 \, x_{i1} \, x_{i2} \, x_{j1} \, x_{j2} \;= \\
&(x_{i1} \, x_{j1} \;+\; x_{i2} \, x_{j2})^2 \;=\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j\right)^2 \,.
\end{aligned}$$

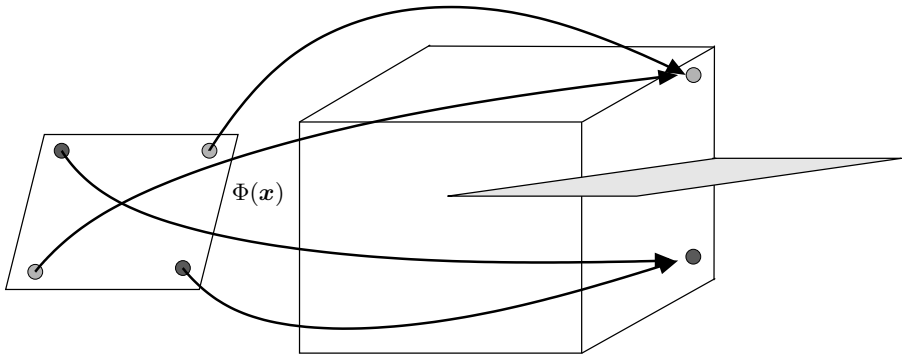


Figure 4.11: An example of a mapping from the two-dimensional space into the three-dimensional space. The data points are not linearly separable in the two-dimensional space but in the three-dimensional space.

In another example we can map the two-dimensional vectors into a 9-dimensional space by

$$\begin{aligned} \boldsymbol{\Phi}(\boldsymbol{x}) \;\; = \;\; & \Big(x_1^3 \, , \; x_2^3 \, , \; \sqrt{3} \, x_1^2 \, x_2 \, , \; \sqrt{3} \, x_2^2 \, x_1 \, , \\ & \sqrt{3} \, x_1^2 \, , \; \sqrt{3} \, x_2^2 \, , \; \sqrt{6} \, x_1 \, x_2 \, , \; \sqrt{3} \, x_1 \, , \; \sqrt{3} \, x_2 \Big) \; . \end{aligned} \tag{4.79}$$

The dot product in the 9-dimensional space is

$$\begin{aligned} & \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^j) \; = \\ & x_{i1}^3 \, x_{j1}^3 \; + \; x_{i2}^3 \, x_{j2}^3 \; + \\ & 3 \, x_{i1}^2 \, x_{i2} \, x_{j1}^2 \, x_{j2} \; + \; 3 \, x_{i2}^2 \, x_{i1} \, x_{j2}^2 \, x_{j1} \; + \\ & 3 \, x_{i1}^2 \, x_{j1}^2 \; + \; 3 \, x_{i2}^2 \, x_{j2}^2 \; + \\ & 6 \, x_{i1} \, x_{i2} \, x_{j1} \, x_{j2} \; + \; 3 \, x_{i1} \, x_{j1} \; + \; 3 \, x_{i2} x_{j2} \; = \\ & = \; \Big(\big(\boldsymbol{x}^i \big)^T \boldsymbol{x}^j \; + \; 1 \Big)^3 \; - 1 \, . \end{aligned}$$

Note that the discriminant function is

$$\begin{aligned} & \boldsymbol{w}^T \boldsymbol{x} \; = \; \sum_{i=1}^{l} \alpha_i \, y^i \; \big(\boldsymbol{x}^i \big)^T \boldsymbol{x} \; = \\ & \sum_{i=1}^{l} \alpha_i \, y^i \; \Big(\big(\boldsymbol{x}^i \big)^T \boldsymbol{x} \; + c \Big) \; , \end{aligned} \tag{4.80}$$

for a constant c, because we have $\sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0$.

Fig. 4.12 shows the SVM architecture which maps into a feature space.

Therefore mapping into the feature space and dot product in this space can be unified by $\Big(\big(\boldsymbol{x}^i \big)^T \boldsymbol{x}^j \; + \; 1 \Big)^3$.

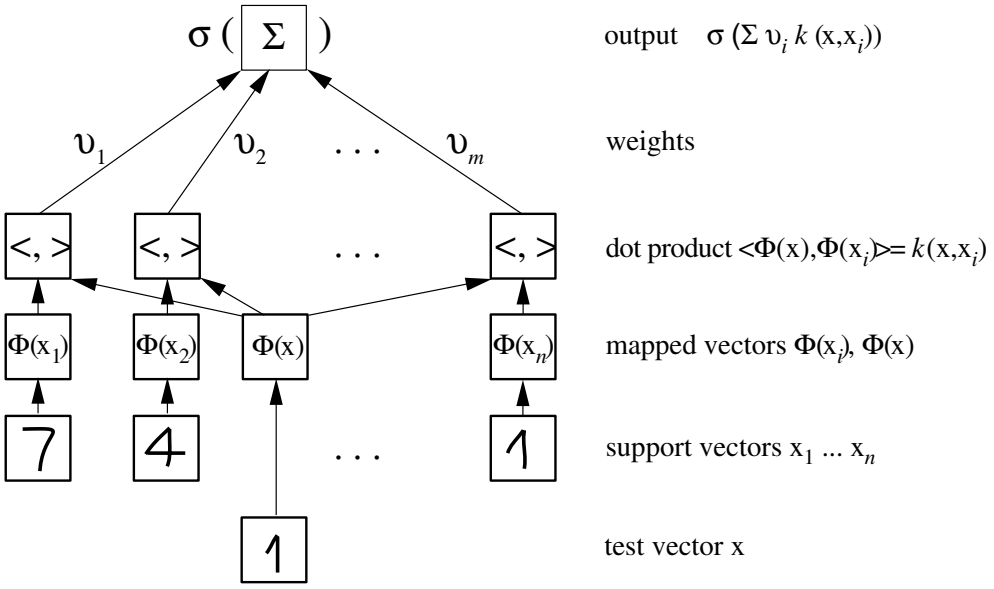


Figure 4.12: The support vector machine with mapping into a feature space is depicted. Images of digits are represented by vectors $\boldsymbol{x}$. These vectors are mapped by $\boldsymbol{\Phi}$ into a feature space in which a dot product is performed. Certain digits are support vectors $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n$. A new digit $\boldsymbol{x}$ is compared to the support vectors by mapping it into the feature space and building the dot product with all support vectors. These dot products are combined as a weighted sum. Finally a function σ (for two classes the sign function) predicts the class label from the weighted sum. Copyright © 2002 [Schölkopf and Smola, 2002] MIT Press.

A function which produces a scalar out of two vectors is called *kernel* k. In our example we have $k\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) \;=\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \;+\; 1\right)^3$.

Certain kernels represent the mapping of vectors into a feature space and a dot product in this space.

The following theorem characterizes functions which build a dot product in some space.

Theorem 4.3 (Mercer)
Let the kernel k be symmetric and from $L_2(X \times X)$ defining a Hilbert-Schmidt operator

$$T_k(f)(\boldsymbol{x}) \;=\; \int_X k(\boldsymbol{x}, \boldsymbol{x}')\; f(\boldsymbol{x}')\; d\boldsymbol{x}' \;. \tag{4.81}$$

If T_k is positive semi-definite, i.e. for all $f \in L_2(X)$

$$\int_{X\times X} k(\boldsymbol{x}, \boldsymbol{x}')\; f(\boldsymbol{x})\; f(\boldsymbol{x}')\; d\boldsymbol{x}\; d\boldsymbol{x}' \;\geq\; 0 \;, \tag{4.82}$$

then T_k has eigenvalues $\lambda_j \;\geq\; 0$ with associated eigenfunctions $\psi_j \in L_2(X)$. Further

$$(\lambda_1, \lambda_2, \ldots) \in \ell_1 \tag{4.83}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') \;=\; \sum_j \lambda_j\; \psi_j(\boldsymbol{x})\; \psi_j(\boldsymbol{x}') \;, \tag{4.84}$$

where ℓ_1 is the space of vectors with finite one-norm and the last sum converges absolutely and uniformly for almost all $\boldsymbol{x}$ and $\boldsymbol{x}'$.

The sum may be an infinite sum for which the eigenvalues converge to zero. In this case the feature space is an infinite dimensional space.

Here "for almost all" means "except for a set with zero measure", i.e. single points may lead to an absolute and uniform convergence. That the convergence is "absolutely and uniformly" is important because the sum can be resorted and derivative and sum can be exchanged.

Note that if X is a compact interval $[a, b]$ and k is continuous then eq. (4.82) is equivalent to positive definiteness of k. A kernel k is *positive semi-definite* if for all l, all $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l$, and all $\alpha_i, 1 \leq i \leq l$

$$\sum_{i,j=1,1}^{l,l} \alpha_i\; \alpha_j\; k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;\geq\; 0 \;. \tag{4.85}$$

We define the *Gram matrix* $\boldsymbol{K}$ as $K_{ij} \;=\; k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ the above inequality is

$$\boldsymbol{\alpha}^T \boldsymbol{K}\; \boldsymbol{\alpha} \;\geq\; 0 \;, \tag{4.86}$$

which means that the Gram matrix has non-negative eigenvalues, i.e. is positive semi-definite.

The mapping $\mathbf{\Phi}$ can be explicitly given as

$$\mathbf{\Phi}(\boldsymbol{x}) \;=\; \left(\sqrt{\lambda_1}\,\psi_1(\boldsymbol{x}), \sqrt{\lambda_2}\,\psi_2(\boldsymbol{x}), \ldots\right) \tag{4.87}$$

Note that $\mathbf{\Phi}(\boldsymbol{x}) \in \ell_2$, where ℓ_2 is the set of all vectors which have finite Euclidean norm.

From the fact that the sum in the theorem converges uniformly it follows that for every ϵ there exists a $N(\epsilon)$ so that the first $N(\epsilon)$ components of $\mathbf{\Phi}(\boldsymbol{x})$ are sufficient to approximate the kernel with error smaller than ϵ. That means infinite dimensional feature spaces can always be approximated by finite feature space of accuracy ϵ.

We express the parameter vector $\boldsymbol{w}$ by a linear weighed sum of support vectors. That means $\boldsymbol{w}$ lives in a subspace of dimension equal or smaller than l, the number of training examples.

If new vectors contain directions orthogonal to the space spanned by the training vectors then these directions are ignored by the classification function.

Note that in general the points in feature space are on a manifold, like a folded sheet of paper in the three-dimensional space. On the other hand the margin in computed in the full space. Here the margin as complexity measure may be questionable because functions are distinguished at regions in space where no data point can appear. The same holds if the original space because features may have dependencies between each other.

The C-SVM with slack variables in its dual form from eq. (4.42) is only expressed by dot products. It can be written in with kernels in the feature space by just replacing $\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j$ by $k(\boldsymbol{x}^i, \boldsymbol{x}^j)$. This is called *the kernel trick*. In any algorithm which is based only on dot products a non-linear algorithm can be constructed by introducing the kernel at the places where the dot products where before.

Applying the kernel trick, the problem in eq. (4.42 is now

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i,j} \alpha_i\, y^i\, \alpha_j\, y^j\, k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;-\; \sum_{i=1}^{l} \alpha_i \\ \text{s.t.} \quad & 0 \;\leq\; \alpha_i \;\leq\; C \\ & \sum_{i=1}^{l} \alpha_i\, y^i \;=\; 0\,. \end{aligned} \tag{4.88}$$

The discriminant function from eq. (4.18) can be formulated using a kernel as

$$\boldsymbol{w}^T \boldsymbol{x} \;+ b \;=\; \sum_{i=1}^{l} \alpha_i\, y^i\, k(\boldsymbol{x}^i, \boldsymbol{x}) \;+\; b\,. \tag{4.89}$$

The most popular kernels are the following:

$$\text{linear kernel:} \quad k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;=\; \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \tag{4.90}$$

$$\text{RBF kernel:} \quad k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;=\; \exp\left(-\frac{\|\boldsymbol{x}^i \;-\; \boldsymbol{x}^j\|^2}{2\,\sigma^2}\right) \tag{4.91}$$

$$\text{polynomial kernel:} \quad k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;=\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \;+\; \beta\right)^{\alpha} \tag{4.92}$$

$$\text{sigmoid kernel:} \quad k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;=\; \tanh\left(\alpha\;\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \;+\; \beta\right) \tag{4.93}$$

Comments:

- The sigmoid kernel is not positive semi-definite for all α and β and is not a very popular choice.
- The RBF kernel is the most popular choice.
- The RBF kernel maps to feature vectors of length 1, because $k(\boldsymbol{x}, \boldsymbol{x}) \;=\; \boldsymbol{\Phi}^T(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; 1$. Therefore the RBF kernel maps onto a hyper-sphere. Because $k(\boldsymbol{x}^1, \boldsymbol{x}^2) > 0$ all vectors are in the same octant in the feature space. The kernel basically measures the angle between the vectors.
- The Gram matrix of the RBF kernel has full rank if training examples are distinct from each other. Thus SVMs with RBF kernels have *infinite VC-dimension*. Fig. 4.13 depicts the situation that an RBF-kernel is sitting on top of each data point, therefore all training points can be separated.
- The space of an RBF kernel is infinite. However the eigenvalues of the Hilbert-Schmidt operator decay exponentially with their number $\lambda_{n+1} \;\leq\; C \exp\left(-\,c\,\sigma\,n^{1/d}\right)$ (Schaback & Wendland, 2002 – C and c are constants independent of n) for a bounded domain in $\mathbb{R}^d$. Thus, σ controls the decay of the eigenvalues and therefore the dimensions of the finite dimensional space for which the kernel can be approximated ϵ-exact with a dot product. Basically σ controls the dimension of the feature space approximation.
- Sums and limits of positive semi-definite (p.d.) kernels are also positive semi-definite kernels. If k_1 and k_2 are p.d. kernels then also $k(\boldsymbol{x}^1, \boldsymbol{x}^2) \;=\; \int_X k_1(\boldsymbol{x}^1, \boldsymbol{x})\; k_1(\boldsymbol{x}^2, \boldsymbol{x})\; \boldsymbol{dx}$ and $k(\boldsymbol{x}^1, \boldsymbol{x}^2) \;=\; k_1(\boldsymbol{x}^1, \boldsymbol{x}^2)\; k_2(\boldsymbol{x}^1, \boldsymbol{x}^2)$.

Fig. 4.16 shows again the example from Fig. 4.6 but now with a polynomial kernel of degree 3. The non-linear separable problem can be separated.

The figures 4.18, 4.19, and 4.20 give SVM examples for the RBF kernel and for the polynomial kernels.

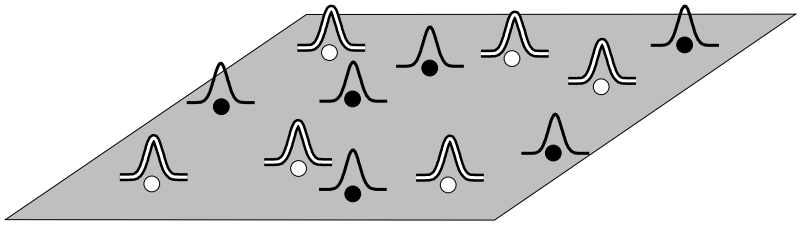


Figure 4.13: An SVM example with RBF kernels. If the width of the kernel is small enough an arbitrary number of training examples can be separated, therefore the VC-dimension is infinity.

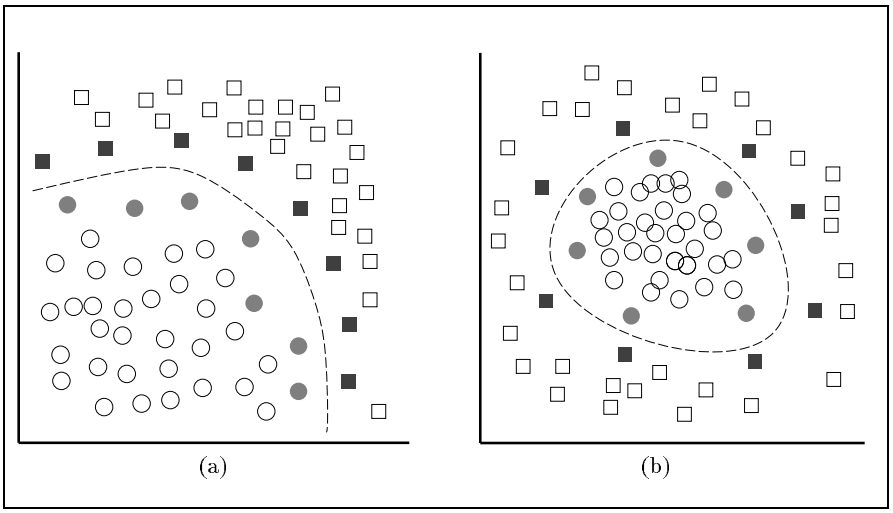


Figure 4.14: Left: An SVM with a polynomial kernel. Right: An SVM with an RBF kernel. Filled points are support vectors. Copyright © 1997 [Osuna et al., 1997].

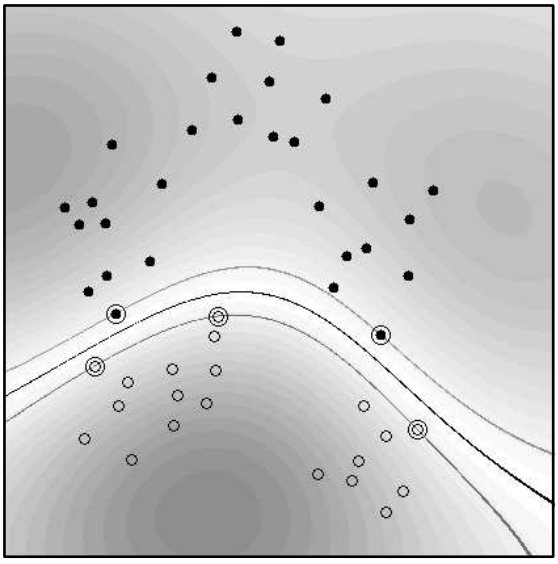


Figure 4.15: SVM classification with an RBF kernel. Support vectors are circled.Copyright ©2002 [Schölkopf and Smola, 2002] MIT Press.

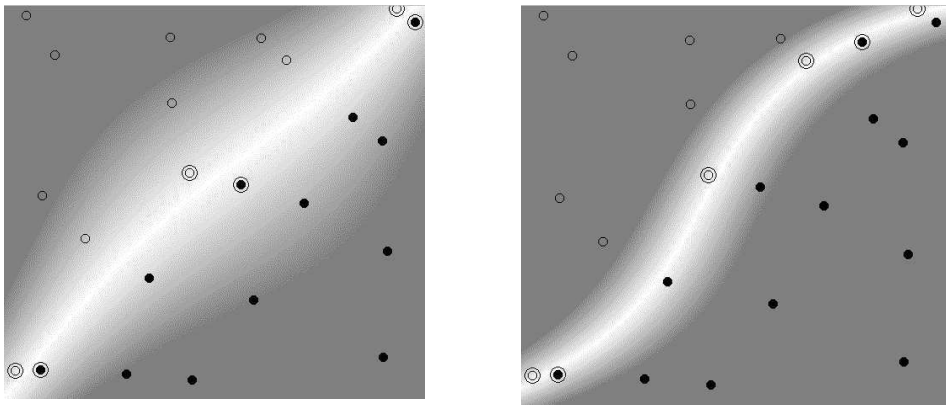


Figure 4.16: The example from Fig. 4.6 but now with polynomial kernel of degree 3. The right task can now be separated. Copyright © 1998 [Burges, 1998].

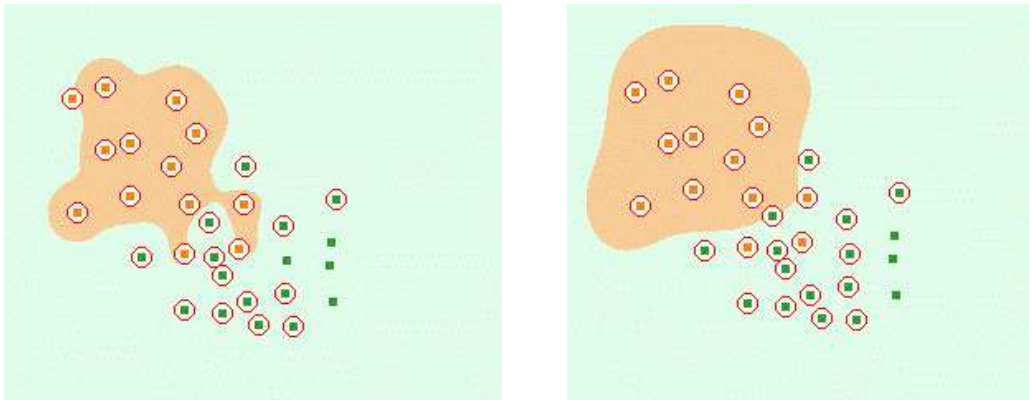


Figure 4.17: SVM with RBF-kernel. Left: small σ. Right: larger σ. The class of points is given by their color. The circled points are support vectors and the color of the regions correspond to the class points in this region would be classified. Screen-shots from `http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSVM.html`.

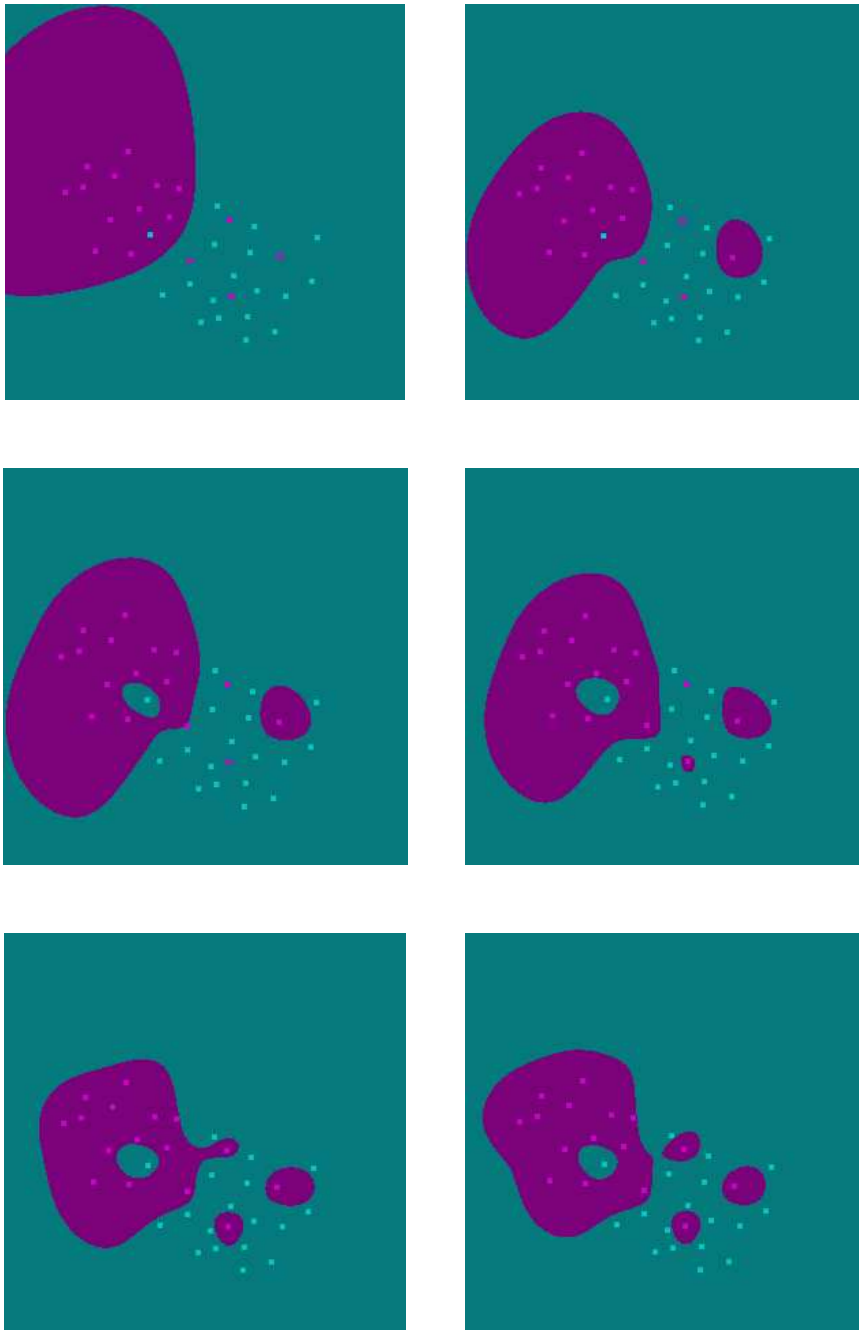


Figure 4.18: SVM with RBF kernel with different σ. From the upper left to the lower right $\sigma = 0.3, 0.18, 0.16, 0.14, 0.12, 0.1$. Screen shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

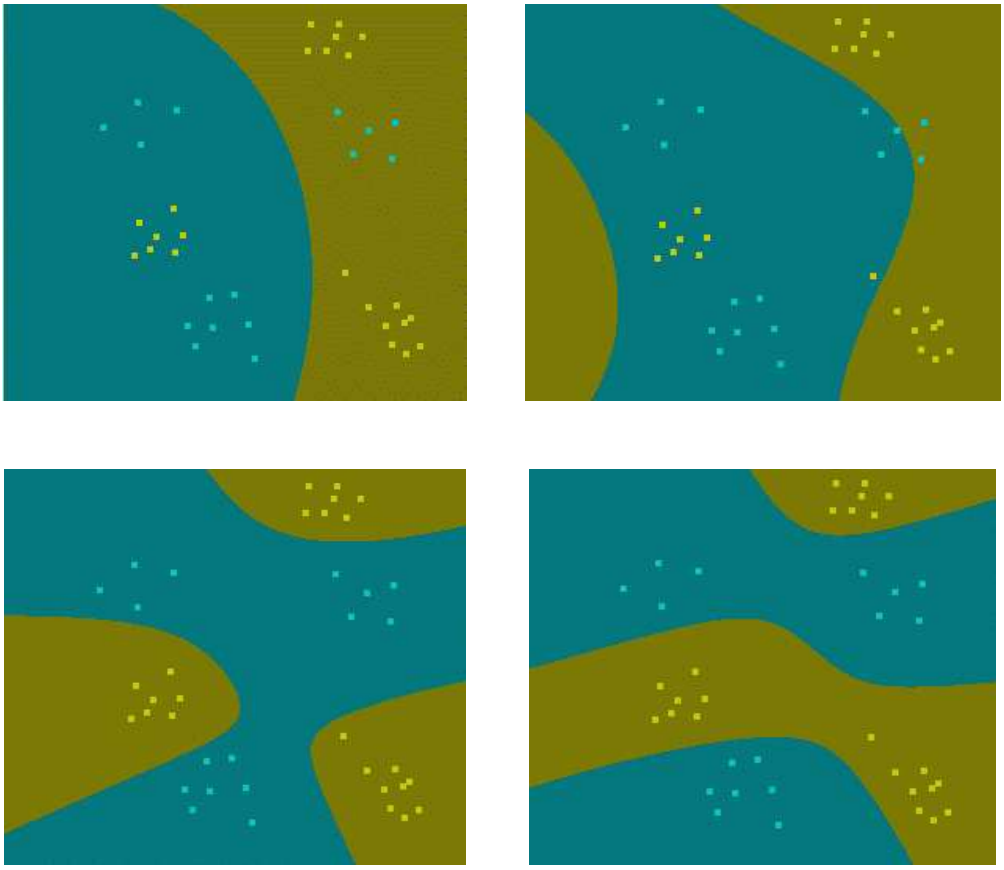


Figure 4.19: SVM with polynomial kernel with different degrees α. From the upper left to the lower right $\alpha = 2, 3, 4, 8$. Screen-shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

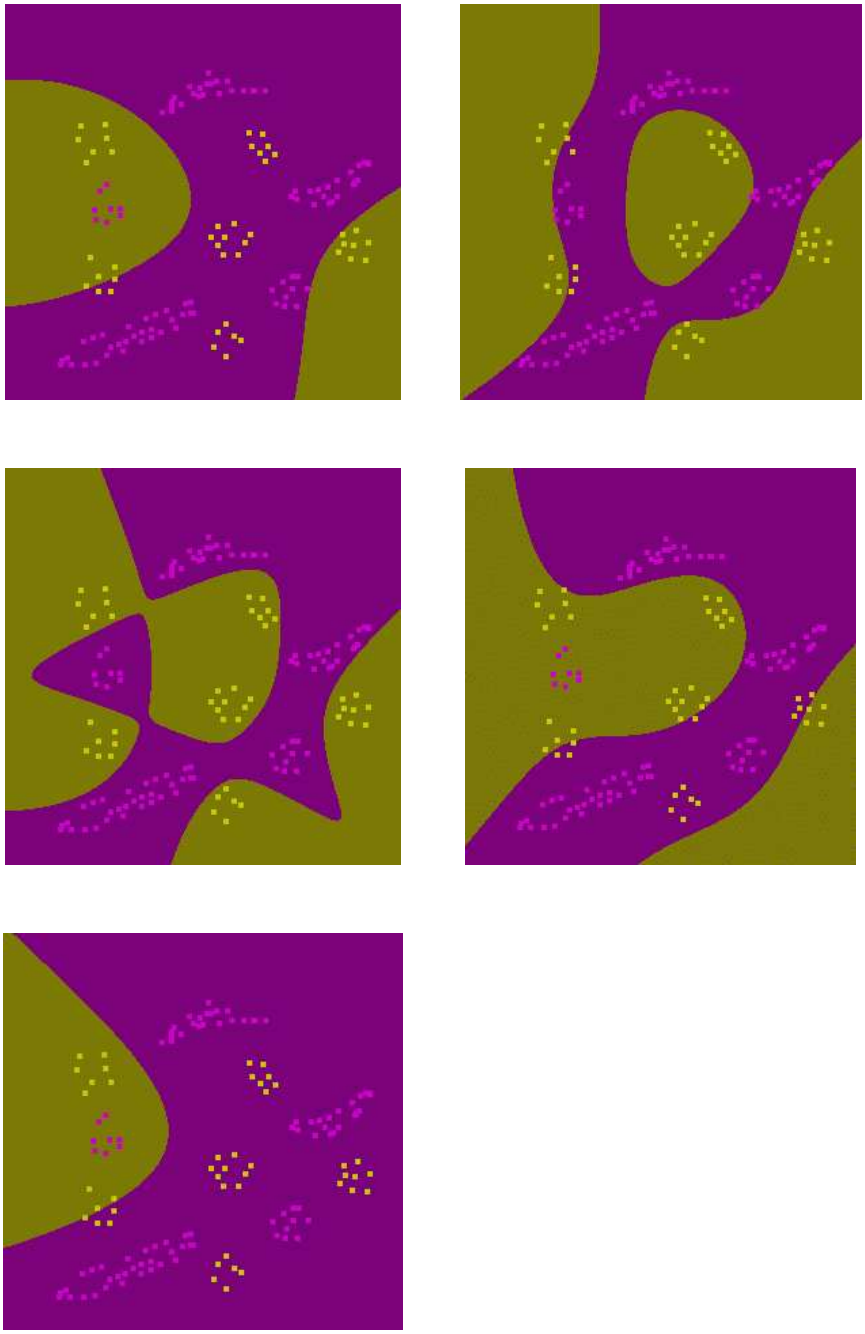


Figure 4.20: SVM with polynomial kernel with degrees $\alpha = 4$ (upper left) and $\alpha = 8$ (upper right) and with RBF kernel with $\sigma = 0.3, 0.6, 1.0$ (from left middle to the bottom). Screen-shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

4.8 Other Interpretation of the Kernel: Reproducing Kernel Hilbert Space

Basicly, this section is from Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Reproducing_kernel_Hilbert_space`.

Hilbert spaces can be defined by reproducing kernels (Nachman Aronszajn and, independently, Stefan Bergman in 1950).

Let X be an arbitrary set and H a Hilbert space of complex-valued functions on X. We say that H is a reproducing kernel Hilbert space if every linear map of the form

$$L_{\boldsymbol{x}} : f \mapsto f(\boldsymbol{x}) \tag{4.94}$$

from H to the complex numbers is continuous for any $\boldsymbol{x}$ in X. That is the evaluation at $\boldsymbol{x}$.

Theorem 4.4 (Riesz representation theorem) *Let H^* denote HÂt's dual space, consisting of all continuous linear functionals from H into the field $\mathbb{C}$. If $\boldsymbol{x}$ is an element of H, then the function ϕ_x defined by*

$$\phi_x(\boldsymbol{y}) = \langle \boldsymbol{y}, \boldsymbol{x} \rangle \quad \forall \boldsymbol{y} \in H \ , \tag{4.95}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product of the Hilbert space, is an element of H^*.

Every element of H^* can be written uniquely in this form, that is the mapping

$$\Phi : H \to H^*, \quad \Phi(x) = \phi_x \tag{4.96}$$

is an isometric (anti-) isomorphism.

This theorem implies that for every $\boldsymbol{x}$ in X there exists an element K_x of H with the property that:

$$f(\boldsymbol{x}) = \langle f, \ K_x \rangle \quad \forall f \in H \quad (*). \tag{4.97}$$

The function K_x is called the point-evaluation functional at the point x.

Since H is a space of functions, the element K_x is itself a function and can therefore be evaluated at every point. We define the function $K : X \times X \to \mathbb{C}$ by

$$K(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} K_x(\boldsymbol{y}). \tag{4.98}$$

This function is called the reproducing kernel for the Hilbert space H and it is determined entirely by H because the Riesz representation theorem guarantees, for every $\boldsymbol{x}$ in X, that the element K_x satisfying (*) is unique.

Properties:

- The reproducing property

  It holds that

$$\langle K(x,\cdot), K(y,\cdot)\rangle \;=\; K(\boldsymbol{x},\boldsymbol{y}). \tag{4.99}$$

- Orthonormal sequences

  If $\{\phi_k\}_{k=1}^{\infty}$ is an orthonormal sequence such that the closure of its span is equal to H, then

$$K\left(\boldsymbol{x},\boldsymbol{y}\right) = \sum_{k=1}^{\infty} \phi_k\left(\boldsymbol{x}\right)\phi_k\left(\boldsymbol{y}\right). \tag{4.100}$$

The next theorem from Moore-Aronszajn states that every symmetric, positive definite kernel defines a unique reproducing kernel Hilbert space.

Theorem 4.5 (Moore-Aronszajn) *Suppose K is a symmetric, positive definite kernel on a set E. Then there is a unique Hilbert space of functions on E for which K is a reproducing kernel.*

Proof. Define, for all $\boldsymbol{x}$ in E, $K_x = K(\boldsymbol{x},\cdot)$. Let H_0 be the linear span of $\{K_x : \boldsymbol{x} \in E\}$. Define an inner product on H_0 by

$$\left\langle \sum_{j=1}^{n} b_j\, K_{\boldsymbol{y}^j}, \sum_{i=1}^{l} \alpha_i\, K_{\boldsymbol{x}^i} \right\rangle \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} \overline{\alpha_i}\,\beta_j\, K(\boldsymbol{y}^j,\boldsymbol{x}^i). \tag{4.101}$$

The symmetry of this inner product follows from the symmetry of K and the non-degeneracy follows from the fact that K is positive definite.

Let H be the completion of H_0 with respect to this inner product. Then H consists of functions of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{\infty} \alpha_i\, K_{x^i}(\boldsymbol{x}) \tag{4.102}$$

where $\sum_{i=1}^{\infty} \alpha_i^2\, K(\boldsymbol{x}^i,\boldsymbol{x}^i) < \infty$. The fact that the above sum converges for every $\boldsymbol{x}$ follows from the Cauchy-Schwartz inequality.

Now we can check the RKHS property, (*):

$$\langle f, K_x\rangle \;=\; \left\langle \sum_{i=1}^{\infty} \alpha_i\, K_{x^i}, K_x \right\rangle = \sum_{i=1}^{\infty} \alpha_i\, K(\boldsymbol{x}^i,\boldsymbol{x}) \;=\; f(\boldsymbol{x}). \tag{4.103}$$

To prove uniqueness, let G be another Hilbert space of functions for which K is a reproducing kernel. For any x and y in E, (*) implies that $\langle K_x, K_y \rangle_H = K(\boldsymbol{x}, \boldsymbol{y}) = \langle K_x, K_y \rangle_G$. By linearity, $\langle \cdot, \cdot \rangle_H = \langle \cdot, \cdot \rangle_G$ on the span of $\{K_x : \boldsymbol{x} \in E\}$. Then $G = H$ by the uniqueness of the completion.

References:

- Nachman Aronszajn, Theory of Reproducing Kernels, Transactions of the American Mathematical Society, volume 68, number 3, pages 337ÂŰ404, 1950.
- Alain Berlinet and Christine Thomas, Reproducing kernel Hilbert spaces in Probability and Statistics, Kluwer Academic Publishers, 2004.
- George Kimeldorf and Grace Wahba, Some results on Tchebycheffian Spline Functions, J. Mathematical Analysis and Applications, 33, 1 (1971) 82ÂŰ95.
- Grace Wahba, Spline Models for Observational Data, SIAM, 1990.
- Felipe Cucker and Steve Smale, On the Mathematical Foundations of Learning, Bulletin of the American Mathematical Society, volume 39, number 1, pages 1ÂŰ49, 2002.

4.9 Example: Face Recognition

The following example is extracted from [Osuna et al., 1997].

An SVM is trained on a database of face and non-face images of size 19×19 pixel. As kernel a polynomial kernel with $\alpha = 2$ and for the SVM parameter $C = 200$ are used.

The data is preprocessed:

- Pixels close to the boundary of the window are removed to reduce background patterns.
- To correct of an illumination gradient, a best-fit brightness plane is subtracted from the window pixel values (reduction of light and heavy shadows).
- A histogram equalization is performed over the patterns in order to compensate for differences in illumination brightness, different camera response curves, etc.

Negative pattern (images without a face) were images of landscapes, trees, buildings, rocks, etc.

The system was used in a bootstrapping setting: misclassifications of the first system were used for a second system as negative examples. In this was difficult examples were filtered out to be recognized by a more specialized system.

The following steps were used to produce an output:

- Re-scaling of the input image several times.
- Cutting of 19×19 window patterns out of the scaled image.
- Preprocessing of the window using a mask, light correction and histogram equalization.

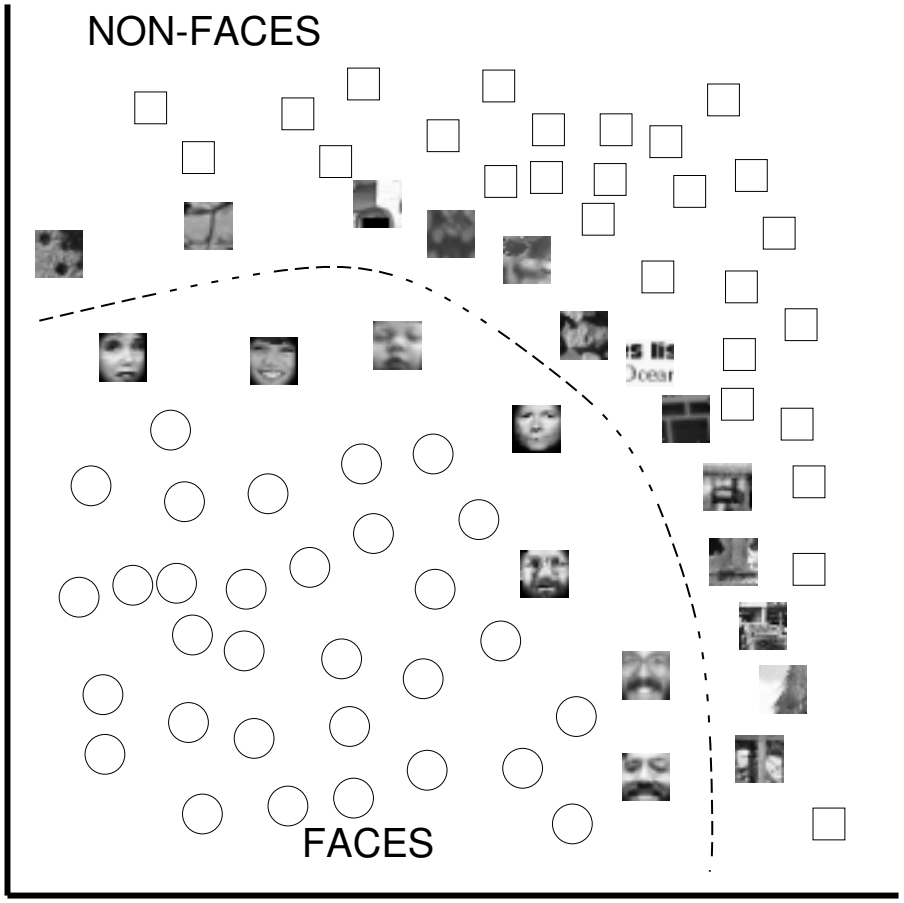


Figure 4.21: Face recognition example. A visualization how the SVM separates faces from non-faces. Support vectors are images of certain faces and images of which are close to face images. The fact that most support vectors are non-faces is depicted in the figure. Copyright © 1997 [Osuna et al., 1997].

- Classification of the pattern using the SVM.
- If the SVM predicts a face, drawing a rectangle around the face in the output image.

Fig. 4.21 depicts how faces should be separated from non-faces using a SVM. More non-faces are used as support vectors because they form the majority class.

Figures 4.22, 4.23, 4.24, 4.25, 4.26, and 4.27 show the result of the face extraction experiment. Figures 4.26 and 4.27 show more difficult tasks where people are not looking into the camera and false positives and false negatives are present.

4.10 Multi-Class SVM

The theory of SVMs has been developed for two classes, i.e. for binary classification. However in real world applications more than one class is available. For example in classifying of proteins

Figure 4.22: Face recognition example. Faces extracted from an image of the Argentina soccer team, an image of a scientist, and the images of a Star Trek crew (even the Klingon face was found!). Copyright © 1997 [Osuna et al., 1997].

Figure 4.23: Face recognition example. Faces are extracted from an image of the German soccer team and two lab images. Copyright © 1997 [Osuna et al., 1997].

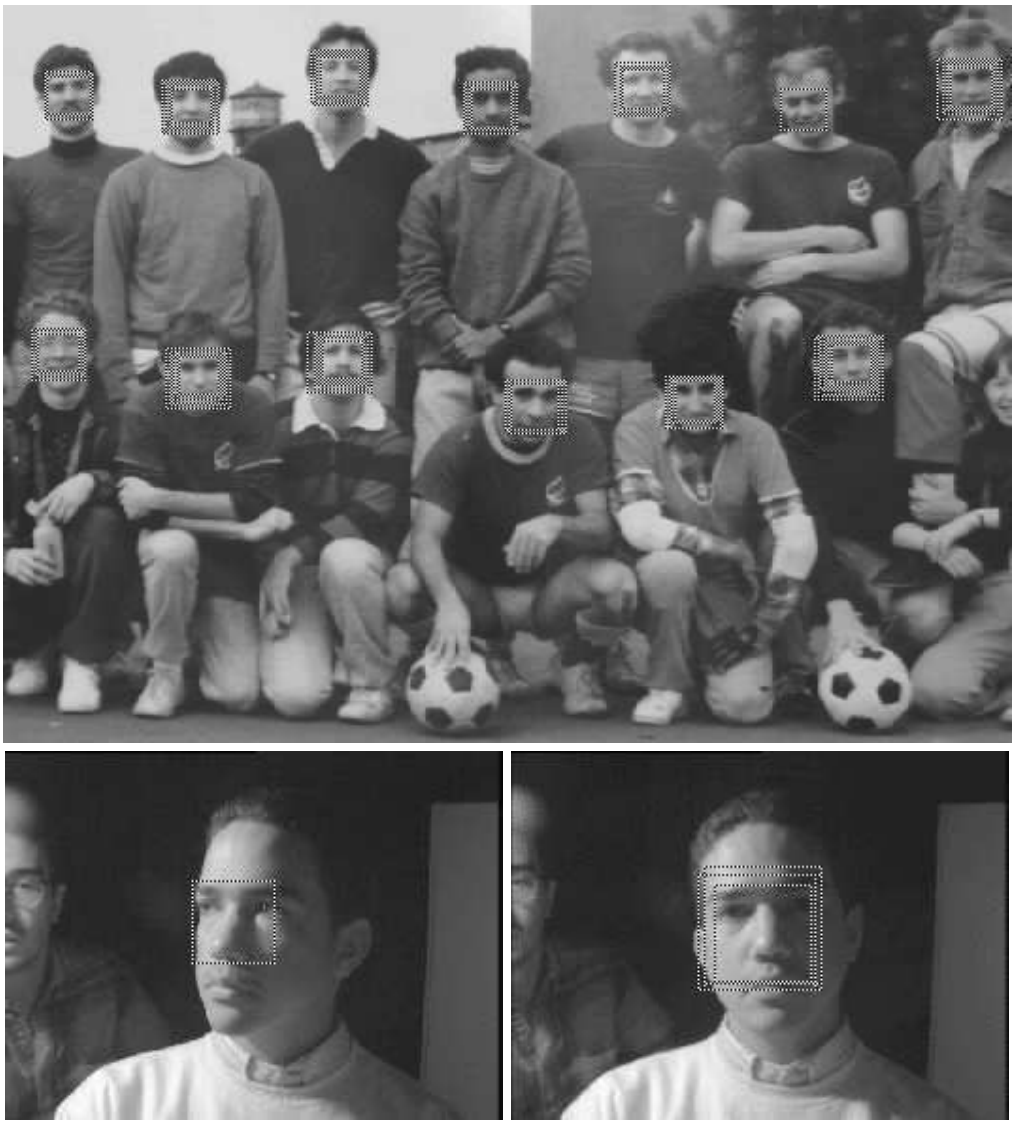


Figure 4.24: Face recognition example. Faces are extracted from another image of a soccer team and two images with lab members. Copyright ©1997 [Osuna et al., 1997].

Figure 4.25: Face recognition example. Faces are extracted from different view and different expressions. Copyright © 1997 [Osuna et al., 1997].

Figure 4.26: Face recognition example. Again faces are extracted from an image of a soccer team but the players are moving during taking the image. Clearly false positives and false negatives are present. Copyright © 1997 [Osuna et al., 1997].

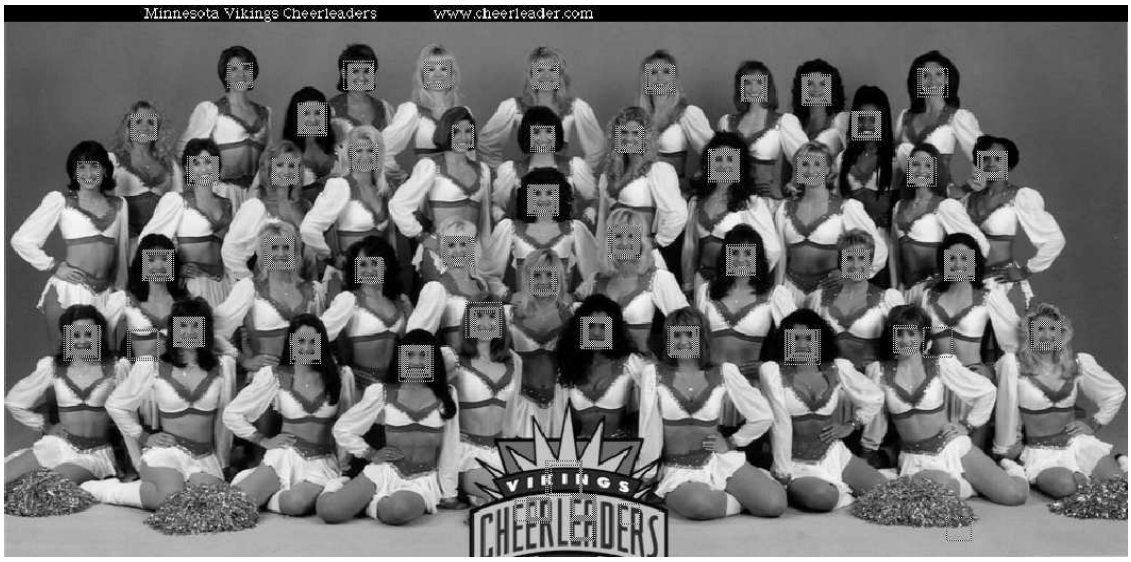


Figure 4.27: Face recognition example. Faces are extracted from a photo of cheerleaders. Here false positives are present. Copyright © 1997 [Osuna et al., 1997].

into structural classes (e.g. according to SCOP) there are many structural classes.

One-against the rest. The first approach to using SVMs to multi-class problems of M classes is to construct for every class j a binary classifier which separates the class from the rest. We assume that a data point belongs exactly to one class.

Given the discriminant functions g_j of these classifiers we can chose the

$$\arg \max_{j=1,\ldots,M} g_j(\boldsymbol{x}) \;=\; \arg \max_{j=1,\ldots,M} \sum_{i=1}^{l} y_j^i \, \alpha_{ij} \, k\left(\boldsymbol{x}, \boldsymbol{x}^i\right) \;+\; b_j \;, \tag{4.104}$$

where $y_j^i = 1$ if $\boldsymbol{x}^i$ belongs to class j and otherwise $y_j^i = -1$, α_{ij} and b_j are the optimal parameters of the "j-th class against the rest" SVM classification problem. This is a "winner-takes-all" approach.

For this approach confusion can exist if two classes have a boundary region where only data of these two classes are located. The classifier is chosen to optimize all data and therefore special parameters for discriminating these two classes are not optimal. In the next approach all pairwise classifiers are considered.

Pairwise Classifiers. Another approach to multi-class SVM is to train (optimize) an SVM for every pair of classes which separates only these two classes. This gives $\frac{M\ (M-1)}{2}$ classification tasks.

Note that the single tasks are computationally not as expensive as the one-against-the-rest because the training set is reduced.

A new data point is assigned to the class which obtained the highest number of votes. A class obtains a vote if a pairwise classifier predicts the class. For large M the procedure can be made

more efficient because not all classifiers must be evaluated: the classes which cannot obtain more or equal as many votes as the best class can be excluded.

Intuitively, for very similar classes a new pattern belonging to these classes results in $(M-2)$ votes for each these classes and the decision is determined by the pairwise classifier of these two classes.

This is the our preferred multi-class approach because in practical problems it often outperformed the other approaches. Let now $y_i \in \{1, \ldots, M\}$ give directly the class $\boldsymbol{x}^i$ belongs to.

Multi-class Objectives. The direct optimization of the multi-class classification problem is possible via

$$\begin{aligned} \min_{\boldsymbol{w}_j, b_j, \boldsymbol{\xi}_j} \quad & \frac{1}{2} \sum_{j=1}^{M} \|\boldsymbol{w}_j\|^2 \; + \; C \sum_{i=1}^{l} \sum_{j \neq y_i} \xi_{ij} \\ \text{s.t.} \quad & \boldsymbol{w}_{y_i}^T \boldsymbol{x}^i \; + \; b_{y_i} \; \geq \; \boldsymbol{w}_j^T \boldsymbol{x}^i \; + \; b_j \; + \; 2 \; - \; \xi_{ij} \\ & \xi_{ij} \; \geq \; 0 \, , \end{aligned} \tag{4.105}$$

for all $i = 1, \ldots, l$ and all $j = 1, \ldots, M$ such that $j \neq y_i$.

The training is more complex here because all variables and data points are treated in one optimization problem. The performance is about the performance of the one-against-the-rest approach.

Comments. Sometimes classification can be made more robust by prediction additional features of the data points. For example proteins can be classified into pure helical or pure beta-sheet structures, or the surface accessibility, polarity, atomic weight, hydrophobicity etc. can be predicted. For discrimination of similar classes these additional information might be helpful. It is similar to "error-correcting output coding", where additional bits help to correct the binary string.

4.11 Support Vector Regression

Now we want to apply the support vector technique to regression, i.e. the task is not to classify the example $\boldsymbol{x}$ by assigning a binary label but to assign a real value. That means y is now a real number $y \in \mathbb{R}$.

But how can regression derived from classification? Ideas from statistical learning theory carry over to regression: a function class is the class which approximates functions with precision ϵ. The class complexity depends now also on ϵ. Also leave-one-out estimates are possible is only few vectors (the support vectors) are used to define the regression function.

If $y \in \boldsymbol{I}$ and $\boldsymbol{I}$ is a compact subset of $\mathbb{R}$ – for simplicity we assume that $\boldsymbol{I}$ is connected, i.e. a interval in $\mathbb{R}$ – then $\boldsymbol{I}$ can be covered by finite many compact intervals of length $\epsilon > 0$. Each of these intervals defines a class. Even more convenient would be that we use only the interval boundaries and classify whether a data point is larger or smaller than the boundary. In this case a linear regression function can be represented through a linear classification task with ϵ precision.

Therefore regression can be transformed into a classification task with $\epsilon > 0$ precision. Reducing the precision allows to described the regression function with few vectors because all zero-loss

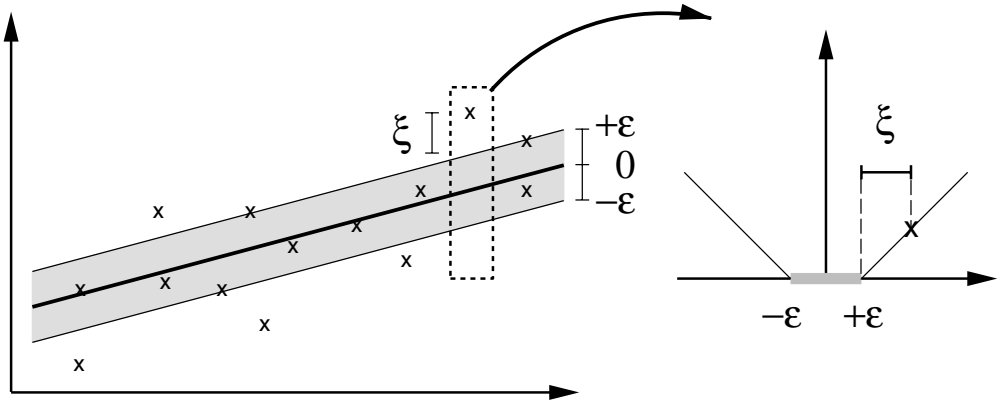


Figure 4.28: Support vector regression. A regression function is found such that a tube with radius ϵ around the regression function contains most data. Points outside the tube are penalized by ξ. Those penalties are traded off against the complexity given by $\|\boldsymbol{w}\|$, which is represented by the slope of a linear function. Copyright © 2002 [Schölkopf and Smola, 2002] MIT Press.

data points do not contribute to the optimal regression function. Support vectors would be at the border of the intervals and outliers, which are data points differing more than ϵ from their neighbors which are equal to one another. However this formulation would be dependent on the choice of the intervals. If the intervals are shifted then permanent support vectors are only the outliers.

To define these permanent support vectors we need the notion of ϵ-insensitive loss function

$$|y - g(\boldsymbol{x})|_\epsilon = \max\{0\,,\, |y - g(x)| - \epsilon\}\,, \tag{4.106}$$

where $\epsilon > 0$. This loss function was introduced by [Vapnik, 1995].

If we now consider again linear functions $\boldsymbol{w}^T\boldsymbol{x} + b$ then we obtain the situation depicted in Fig. 4.28. The errors must be distinguished as being more than ϵ below the target y or more than ϵ above the target y and are defined as

$$\begin{aligned} \xi^- &= \max\{0\,,\, g(x) - y - \epsilon\} \\ \xi^+ &= \max\{0\,,\, y - g(x) - \epsilon\} \\ \xi &= \left(\xi^- + \xi^+\right) \end{aligned} \tag{4.107}$$

where again only $\boldsymbol{x}^i$ with $\xi_i > 0$ are support vectors.

The complexity is again defined through $\|\boldsymbol{w}\|^2$. *However the motivation through a maximal margin is no longer valid.*

Small $\|\boldsymbol{w}\|^2$ means a flat function, e.g. in the linear case a small slope.

More interesting small $\|\boldsymbol{w}\|^2$ means *small variation* around the constant b of the regression function on a compact set of feature vectors $\boldsymbol{x}$. On the compact set $\|\boldsymbol{x}\| \leq R$ holds. We obtain with the Cauchy-Schwarz inequality

$$\left\|\boldsymbol{w}^T\boldsymbol{x}\right\| \leq \|\boldsymbol{w}\|\,\|\boldsymbol{x}\| \leq \|\boldsymbol{w}\|\,R\,. \tag{4.108}$$

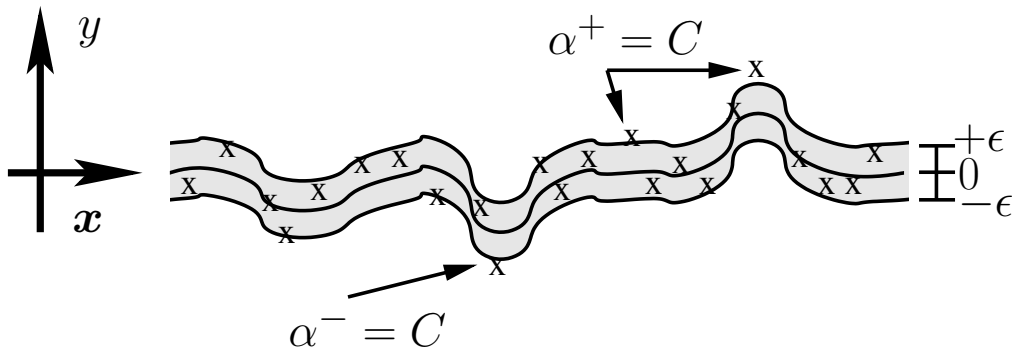


Figure 4.29: Nonlinear support vector regression is depicted. A regression function is found such that a tube with radius ϵ around the regression function contains most data.

This holds also for the kernel version which means for $g(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i \, y^i \, k(\boldsymbol{x}, \boldsymbol{x}^i) + b$:

$$\|g(\boldsymbol{x}) - b\| \leq \|\boldsymbol{w}\| \, R \leq \sqrt{\sum_{i,j} \alpha_i \, \alpha_j \, k(\boldsymbol{x}^i, \boldsymbol{x}^j)} \, R \leq \|\boldsymbol{\alpha}\| \, \sqrt{e_{\max}} \, R \,, \tag{4.109}$$

where $e_{\max}$ is the maximal eigenvalue of the kernel matrix, i.e. the Gram matrix.

We obtain as optimization problem

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-} \quad & \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{l} \sum_{i=1}^{l} \left(\xi_i^+ + \xi_i^-\right) \\ \text{s.t.} \quad & y^i - \left(\boldsymbol{w}^T \boldsymbol{x}^i + b\right) \leq \epsilon + \xi_i^+ \\ & \left(\boldsymbol{w}^T \boldsymbol{x}^i + b\right) - y^i \leq \epsilon + \xi_i^- \\ & \xi_i^+ \geq 0 \text{ and } \xi_i^- \geq 0 \,. \end{aligned} \tag{4.110}$$

The Lagrangian is

$$\begin{aligned} & L\left(\boldsymbol{w}, b, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^-\right) = \\ & \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{l} \sum_{i=1}^{l} \left(\xi_i^+ + \xi_i^-\right) - \sum_{i=1}^{l} \left(\mu_i^+ \, \xi_i^+ + \mu_i^- \, \xi_i^-\right) - \\ & \sum_{i=1}^{l} \alpha_i^+ \left(\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T \boldsymbol{x}^i + b\right) - \\ & \sum_{i=1}^{l} \alpha_i^- \left(\epsilon + \xi_i^- + y^i - \boldsymbol{w}^T \boldsymbol{x}^i - b\right) \,, \end{aligned} \tag{4.111}$$

where $\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^- \geq \mathbf{0}$.

The saddle point conditions require

$$\begin{aligned}
\frac{\partial L}{\partial b} &= \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) = 0 \\
\frac{\partial L}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) \boldsymbol{x}^i = \mathbf{0} \\
\frac{\partial L}{\partial \boldsymbol{\xi}^+} &= \frac{C}{l}\mathbf{1} - \boldsymbol{\alpha}^+ - \boldsymbol{\mu}^+ = \mathbf{0} \\
\frac{\partial L}{\partial \boldsymbol{\xi}^-} &= \frac{C}{l}\mathbf{1} - \boldsymbol{\alpha}^- - \boldsymbol{\mu}^- = \mathbf{0}
\end{aligned} \tag{4.112}$$

The last two equations constrain the α_i^+ and α_i^- whereas the first two equations can be inserted into the Lagrangian to eliminate the primal variables.

We obtain as dual formulation

$$\begin{aligned}
\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \quad & \frac{1}{2} \sum_{i,j=(1,1)}^{(l,l)} \left(\alpha_i^+ - \alpha_i^-\right) \left(\alpha_j^+ - \alpha_j^-\right) \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j + \\
& \epsilon \sum_{i=1}^{l} \left(\alpha_i^+ + \alpha_i^-\right) - \sum_{i=1}^{l} y^i \left(\alpha_i^+ - \alpha_i^-\right) \\
\text{s.t.} \quad & 0 \leq \alpha_i^+ \leq \frac{C}{l} \\
& 0 \leq \alpha_i^- \leq \frac{C}{l} \\
& \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) = 0 \,.
\end{aligned} \tag{4.113}$$

The regression function can be written as

$$g(\boldsymbol{x}) = \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) \left(\boldsymbol{x}^i\right)^T \boldsymbol{x} + b \,. \tag{4.114}$$

The KKT conditions state for the optimal parameters:

$$\begin{aligned}
& \alpha_i^+ \left(\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T \boldsymbol{x}^i + b\right) = 0 \\
& \alpha_i^- \left(\epsilon + \xi_i^- + y^i - \boldsymbol{w}^T \boldsymbol{x}^i - b\right) = 0 \\
& \xi_i^+ \, \mu_i^+ = 0 \\
& \xi_i^- \, \mu_i^- = 0 \,.
\end{aligned} \tag{4.115}$$

From $\xi_i^+ > 0$ follows that $\mu_i^+ = 0$ and therefore $\alpha_i^+ = \frac{C}{l}$. The analog holds for ξ_i^-, μ_i^- and α_i^-. We obtain as conditions

$$\begin{aligned} \xi_i^+ \left(\frac{C}{l} - \alpha_i^+\right) &= 0 \\ \xi_i^- \left(\frac{C}{l} - \alpha_i^-\right) &= 0\,. \end{aligned} \tag{4.116}$$

The data points $\boldsymbol{x}^i$ with $\alpha_i^+ = \frac{C}{l}$ are outside the ϵ-tube.

Data points with $0 < \alpha_i^+ < \frac{C}{l}$ are on the ϵ-tube border (same for $0 < \alpha_i^- < \frac{C}{l}$). Because $\alpha_i^+ < \frac{C}{l}$ implies $\mu_i^+ > 0$ and therefore $\xi_i^+ = 0$. But $0 < \alpha_i^+$ implies $\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T\boldsymbol{x}^i + b = 0$ that is $\epsilon + -y^i + \boldsymbol{w}^T\boldsymbol{x}^i + b = 0$ from which b can be computed as

$$b = y^i - \boldsymbol{w}^T\boldsymbol{x}^i - \epsilon\,. \tag{4.117}$$

And for $0 < \alpha_i^- < \frac{C}{l}$ the value b can be computed through

$$b = y^i - \boldsymbol{w}^T\boldsymbol{x}^i + \epsilon\,. \tag{4.118}$$

If $\alpha_i^+ > 0$ and $\alpha_i^- > 0$, then

$$\begin{aligned} &\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T\boldsymbol{x}^i + b = 0 \text{ and} \\ &\epsilon + \xi_i^- + y^i - \boldsymbol{w}^T\boldsymbol{x}^i - b = 0\,. \end{aligned} \tag{4.119}$$

and adding both equations gives

$$2\,\epsilon + \xi_i^+ + \xi_i^- = 0 \tag{4.120}$$

which contradicts $\epsilon > 0$.

Therefore we have

$$\alpha_i^+\,\alpha_i^- = 0\,. \tag{4.121}$$

We can set $\alpha = (\alpha^+ - \alpha^-)$ then α^+ is the positive alpha part and α^- the negative part, where only one part exists. Note that $\|\boldsymbol{\alpha}\|_1 = \sum_{i=1}^{l}\left(\alpha_i^+ + \alpha_i^-\right)$. Therefore the dual can be expressed compactly as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i,j=(1,1)}^{(l,l)} \alpha_i\,\alpha_j\,\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j - \\ & \epsilon\,\|\boldsymbol{\alpha}\|_1 + \sum_{i=1}^{l} y^i\,\alpha_i \\ \text{s.t.} \quad & -\frac{C}{l} \le \alpha_i \le \frac{C}{l} \\ & \sum_{i=1}^{l}\alpha_i = 0\,. \end{aligned} \tag{4.122}$$

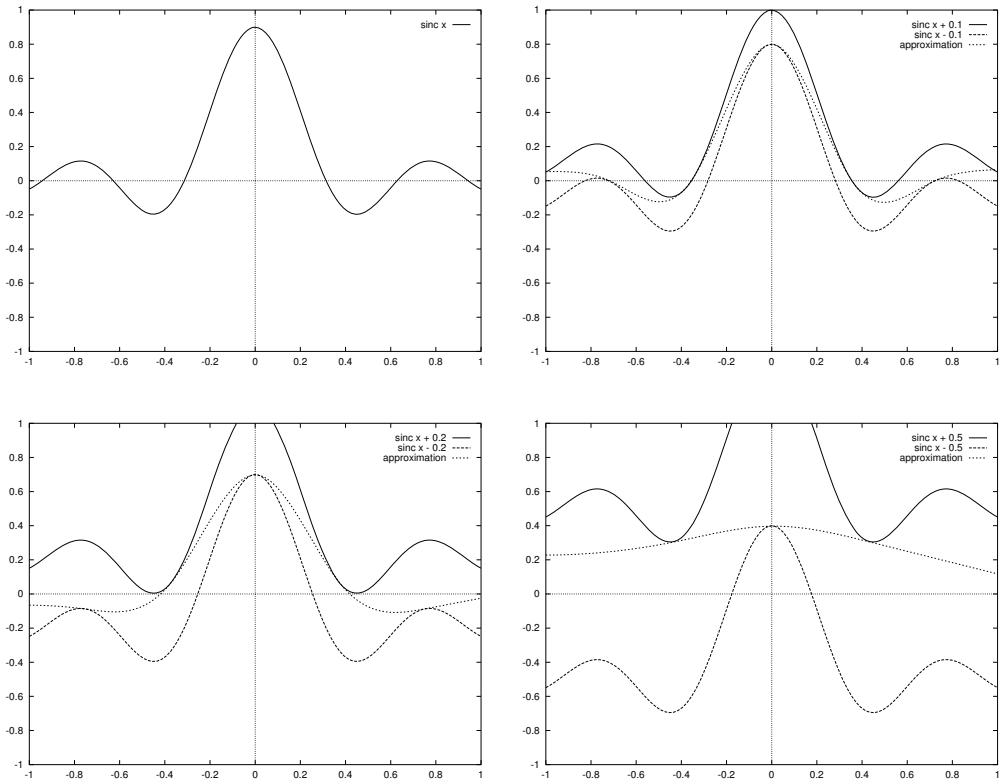


Figure 4.30: Example of SV regression: smoothness effect of different ϵ. Upper left: the target function $\mathrm{sinc}x$, upper right: SV regression with $\epsilon = 0.1$, lower left: $\epsilon = 0.2$, and lower right: $\epsilon = 0.5$. Larger ϵ leads to a smoother approximation function which fits into the broader ϵ-tube. Copyright © 2002 [Smola and Schölkopf, 2002].

or in vector notation

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\,\boldsymbol{\alpha}\boldsymbol{X}^T\boldsymbol{X}\,\boldsymbol{\alpha} \;-\; \epsilon\,\|\boldsymbol{\alpha}\|_1 \;+\; \boldsymbol{y}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & -\frac{C}{l}\mathbf{1} \;\leq\; \boldsymbol{\alpha} \;\leq\; \frac{C}{l}\mathbf{1},\;\; 1^T\boldsymbol{\alpha} \;=\; 0\,. \end{aligned} \tag{4.123}$$

An example for support vector regression is given in Fig. 4.30. Larger ϵ leads to a smoother approximation function which fits into the broader ϵ-tube. Fig. 4.31 shows that decreasing ϵ results in an increase of the number of support vectors. As depicted in Fig. 4.32 the support vectors pull the approximation/regression function into the ϵ-tube.

ν-SVM Regression.

Analog to classification also a ν-support vector regression (ν-SVR) approach can be formulated. Large ϵ is penalized and traded against complexity, i.e. smoothness.

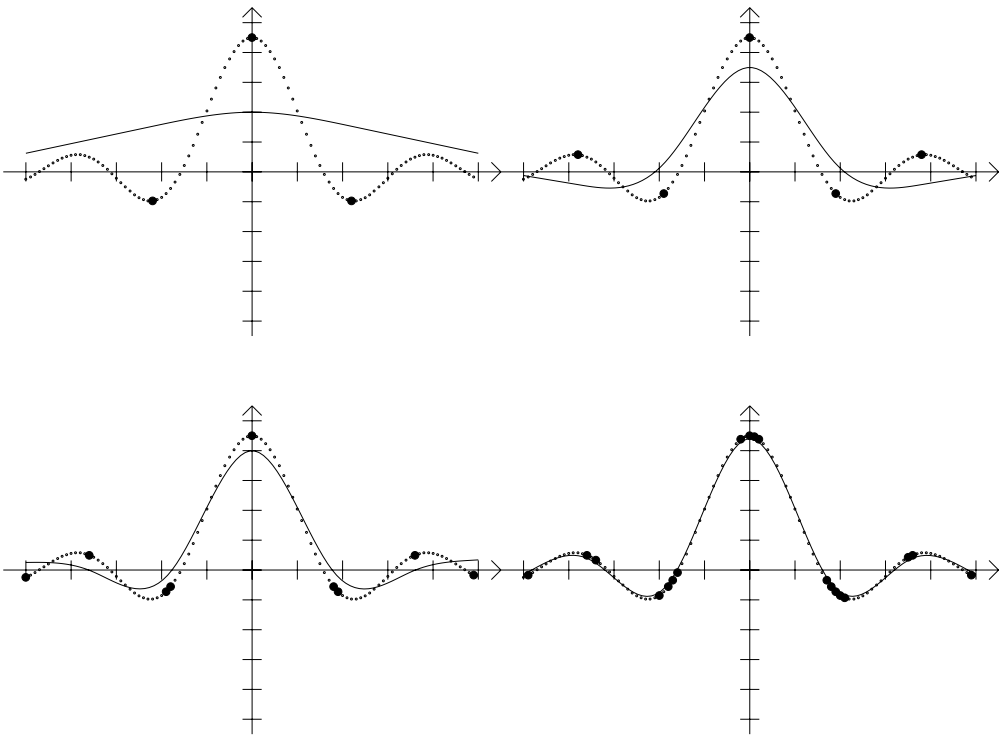


Figure 4.31: Example of SV regression: support vectors for different ϵ. Example from Fig. 4.30. The tiny dots are the data points and the big dots are support vectors. Upper left: $\epsilon = 0.5$, upper right: $\epsilon = 0.2$, lower left: $\epsilon = 0.1$, and lower right: $\epsilon = 0.002$. The number of support vectors increases with smaller ϵ. The support vectors can be viewed as applying a force on a horizontal rubber band. Copyright © 2002 [Smola and Schölkopf, 2002].

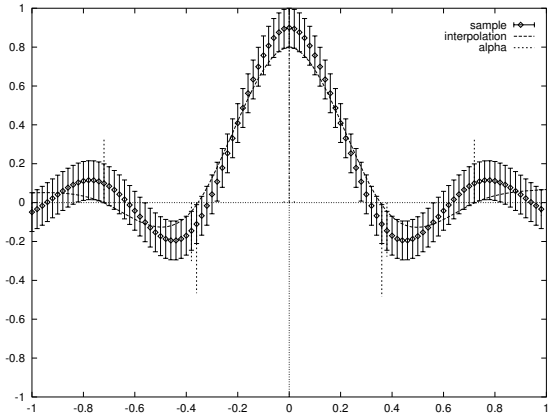


Figure 4.32: Example of SV regression: support vectors pull the approximation curve inside the ϵ-tube. Example from Fig. 4.30. Dash-dotted vertical lines indicate where the approximation curve is pulled inside the ϵ-tube. At these positions support vectors are located. Copyright © 2002 [Smola and Schölkopf, 2002].

We obtain as optimization problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},b,\boldsymbol{\xi}^+,\boldsymbol{\xi}^-,\epsilon} & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \,+\, C\,\left(\nu\,\epsilon \,+\, \frac{1}{l}\,\sum\limits_{i=1}^{l}\left(\xi_i^+ \,+\, \xi_i^-\right)\right) \\ \text{s.t.} & y^i \,-\, \left(\boldsymbol{w}^T\boldsymbol{x}^i \,+\, b\right) \,\leq\, \epsilon \,+\, \xi_i^+ \\ & \left(\boldsymbol{w}^T\boldsymbol{x}^i \,+\, b\right) \,-\, y^i \,\leq\, \epsilon \,+\, \xi_i^- \\ & \xi_i^+ \,\geq\, 0,\, \xi_i^- \,\geq\, 0 \text{ and } \epsilon \,\geq\, 0\,. \end{array} \tag{4.124}$$

The Lagrangian is

$$\begin{array}{l} L\left(\boldsymbol{w},b,\boldsymbol{\xi}^+,\boldsymbol{\xi}^-,\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-,\boldsymbol{\mu}^+,\boldsymbol{\mu}^-,\epsilon,\beta\right) \,= \\ \frac{1}{2}\,\|\boldsymbol{w}\|^2 \,+\, C\,\nu\,\epsilon \,+\, \frac{C}{l}\,\sum\limits_{i=1}^{l}\left(\xi_i^+ \,+\, \xi_i^-\right) \,-\, \beta\,\epsilon \,- \\ \sum\limits_{i=1}^{l}\left(\boldsymbol{\mu}_i^+\,\xi_i^+ \,+\, \boldsymbol{\mu}_i^-\,\xi_i^-\right) \,- \\ \sum\limits_{i=1}^{l}\alpha_i^+\left(\epsilon \,+\, \xi_i^+ \,-\, y^i \,+\, \boldsymbol{w}^T\boldsymbol{x}^i \,+\, b\right) \,- \\ \sum\limits_{i=1}^{l}\alpha_i^-\left(\epsilon \,+\, \xi_i^- \,+\, y^i \,-\, \boldsymbol{w}^T\boldsymbol{x}^i \,-\, b\right)\,, \end{array} \tag{4.125}$$

where $\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^- \geq \mathbf{0}$.

The saddle point conditions require

$$\begin{array}{l} \frac{\partial L}{\partial b} \,=\, \sum\limits_{i=1}^{l}\left(\alpha_i^+ \,+\, \alpha_i^-\right) \,=\, 0 \\ \frac{\partial L}{\partial \boldsymbol{w}} \,=\, \boldsymbol{w} \,-\, \sum\limits_{i=1}^{l}\left(\alpha_i^+ \,-\, \alpha_i^-\right)\,\boldsymbol{x}^i \,=\, \mathbf{0} \\ \frac{\partial L}{\partial \boldsymbol{\xi}^+} \,=\, \frac{C}{l}\,\mathbf{1} \,-\, \boldsymbol{\alpha}^+ \,-\, \boldsymbol{\mu}^+ \,=\, \mathbf{0} \\ \frac{\partial L}{\partial \boldsymbol{\xi}^-} \,=\, \frac{C}{l}\,\mathbf{1} \,-\, \boldsymbol{\alpha}^- \,-\, \boldsymbol{\mu}^- \,=\, \mathbf{0} \\ \frac{\partial L}{\partial \epsilon} \,=\, C\,\nu \,-\, \sum\limits_{i=1}^{l}\left(\boldsymbol{\alpha}^+ \,+\, \boldsymbol{\alpha}^-\right) \,-\, \beta \,=\, 0\,. \end{array} \tag{4.126}$$

We obtain as dual formulation

$$\begin{aligned}
\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \quad & \frac{1}{2}\sum_{i,j=(1,1)}^{(l,l)} \left(\alpha_i^+ - \alpha_i^-\right)\left(\alpha_j^+ - \alpha_j^-\right)\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j - \\
& \sum_{i=1}^{l} y^i \left(\alpha_i^+ - \alpha_i^-\right) \\
\text{s.t.} \quad & 0 \le \alpha_i^+ \le \frac{C}{l} \\
& 0 \le \alpha_i^- \le \frac{C}{l} \\
& \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) = 0 \\
& \sum_{i=1}^{l} \left(\alpha_i^+ + \alpha_i^-\right) \le C\,\nu\,.
\end{aligned} \tag{4.127}$$

The normal vector of the regression function is given by $\boldsymbol{w} = \sum_{i=1}^{l}\left(\alpha_i^+ - \alpha_i^-\right)\boldsymbol{x}^i$.

Again b and ϵ can be computed for a pair of α_i^+ and α_j^- with $0 < \alpha_i^+, \alpha_j^- < \frac{C}{l}$ for which

$$\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T\boldsymbol{x}^i + b = 0 \tag{4.128}$$

$$\epsilon + \xi_j^- + y^j - \boldsymbol{w}^T\boldsymbol{x}^j - b = 0\,. \tag{4.129}$$

Again we obtain similar statements as for the classification problem:

- ν is an upper bound on the fraction of errors (points outside the ϵ-tube).
- ν is a lower bound on the fraction of support vectors.
- Under mild conditions: with probability 1, asymptotically, ν equals both the fraction of support vector and the fraction of errors.

Figures 4.33, 4.34, and 4.35 show examples for ν-SV regression, where different ϵ, ν, and kernel width σ are tried out.

Comments. In general much more support vectors are produced for the regression case if compared to a classification problem. Also fast optimizers need much more time to find the solution of a regression problem.

A robust loss function is also known from statistics to ensure robustness [Huber, 1981] who defines the loss as

$$\begin{cases} \frac{1}{2\,\sigma}\left(y - g(x)\right)^2 & \text{if} \quad |y - g(x)| < \sigma \\ |y - g(x)| - \frac{\sigma}{2} & \text{otherwise} \end{cases} \tag{4.130}$$

which is quadratic around zero and linear (Laplacian) elsewhere.

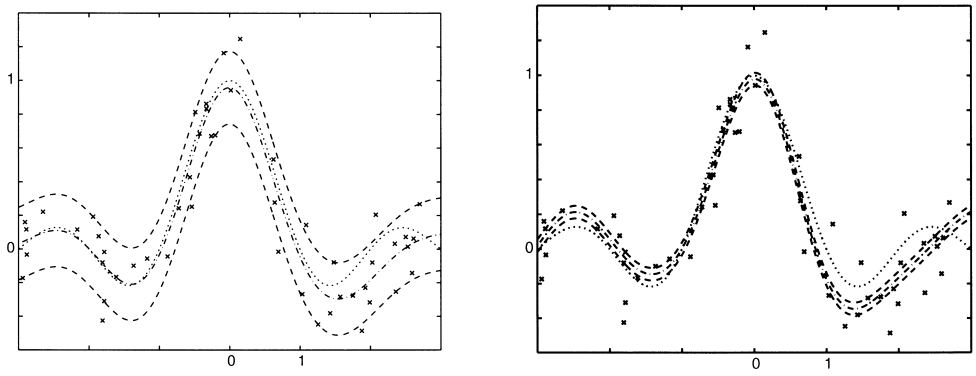


Figure 4.33: ν-SV regression with $\nu = 0.2$ (left) and $\nu = 0.8$ (right). ϵ is automatically adjusted to $\epsilon = 0.22$ (left) and $\epsilon = 0.04$ (right). Large ν allows more points to be outside the tube. Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

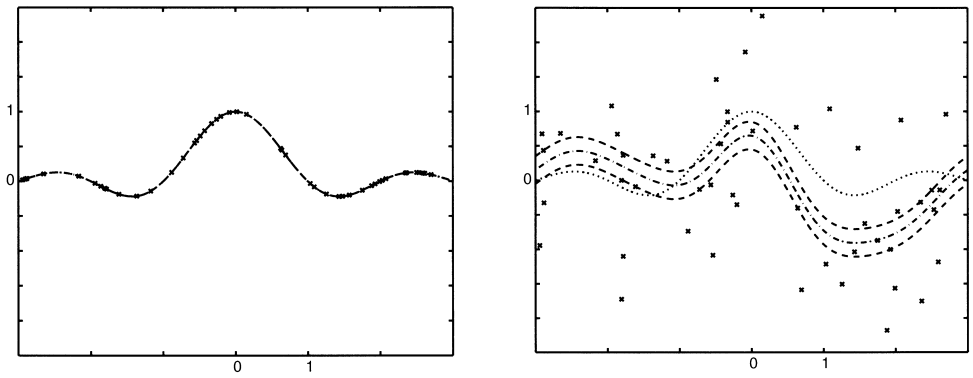


Figure 4.34: ν-SV regression where ϵ is automatically adjusted to the noise level. Noise level $\sigma = 0$ (left) and $\sigma = 1.0$ lead to $\epsilon = 0$ (left) and $\epsilon = 1.19$ (right), respectively. In both cases the same parameter $\nu = 0.2$ was used. Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

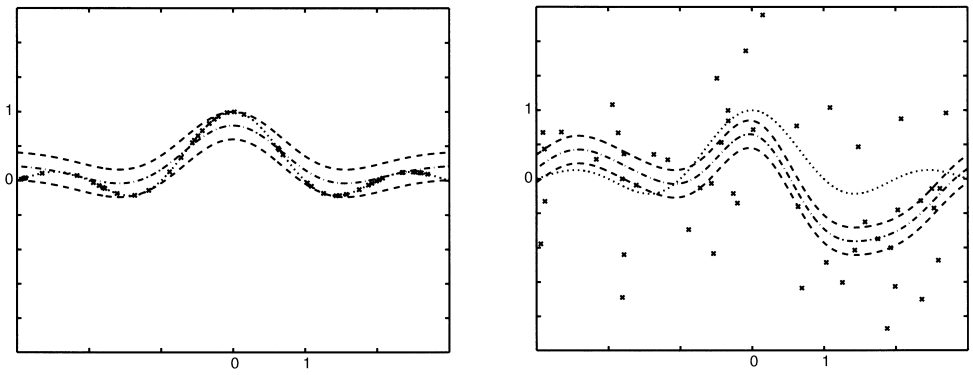


Figure 4.35: Standard SV regression with the example from Fig. 4.34 (noise levels: left $\sigma = 0$ and right $\sigma = 1.0$). $\epsilon = 0.2$ was fixed which is neither optimal for the low noise case (left) nor for the high noise case (right). Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

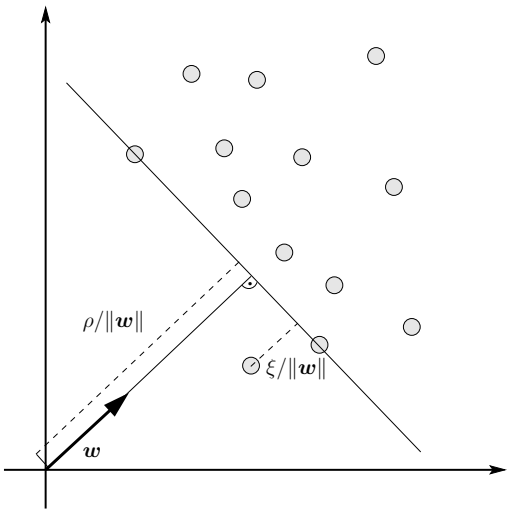


Figure 4.36: The idea of the one-class SVM is depicted. The data points are separated from the origin with the hyperplane which has the largest margin.

4.12 One Class SVM

Can the support vector technique be used for unsupervised learning? Yes!

An unsupervised SVM method called "one-class support vector machine" has been developed [Schölkopf et al., 2001] which can be used for novelty detection and quantile estimation. The later can be in turn used to filter data, i.e. select all candidates which are on quantiles with very large margin.

The one-class support vector machine found its application already in bioinformatics. In bioinformatics tasks sometimes only one class can be determined with certainty, which makes the one-class SVM a natural choice. For example protein-protein interactions are experimentally verified however falsified data is not available. Protein-protein interactions were characterized by one-class SVM [Alashwal et al., 2006]. In another application the goal was to detect new classes in leukemia and lymphoma data set where also the one-class SVM was applied [Spinosa and de Carvalho, 2005].

One-class SVMs can be used as a filtering method similar to the approach in [Vollgraf et al., 2004]. If the negative class is huge then an one-class SVM may filter out all positives even if a lot of false positives are present. The true positives can be separated from the false positives in a binary classification task. The one-class SVM reduced the data set as a preprocessing step to a binary classifier.

The idea of the one-class SVM is to separate the positive class from the origin, which is considered as a single point of the negative class. In the context of SVMs the separation should be performed with maximal margin. The idea is depicted in Fig. 4.36.

We now apply the ν-SVM formulation from eq. (4.54) to this problem:

$$\begin{aligned} \min_{\tilde{\boldsymbol{w}},\tilde{b},\tilde{\boldsymbol{\xi}},\tilde{\rho}} \quad & \frac{1}{2}\,\|\tilde{\boldsymbol{w}}\|^2 \;-\; \nu\,\tilde{\rho} \;+\; \frac{1}{l}\,\sum_{i=1}^{l}\tilde{\xi}_i \\ \text{s.t.} \quad & y^i\,\left(\tilde{\boldsymbol{w}}^T\boldsymbol{x}^i \;+\; \tilde{b}\right) \;\geq\; \tilde{\rho} \;-\; \tilde{\xi}_i \\ & \tilde{\xi}_i \;\geq\; 0 \;\text{ and }\; \tilde{\rho} \;\geq\; 0\,. \end{aligned} \tag{4.131}$$

First we re-scale the values because the hyperplane is not supposed to be in its canonical form:

$$\tilde{b} \;=\; \nu\,b \tag{4.132}$$

$$\tilde{\boldsymbol{w}} \;=\; \nu\,\boldsymbol{w} \tag{4.133}$$

$$\tilde{\rho} \;=\; \nu\,\rho \tag{4.134}$$

$$\tilde{\xi} \;=\; \nu\,\xi \tag{4.135}$$

and obtain after scaling

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \rho \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; \rho \;-\; \xi_i \\ & \xi_i \;\geq\; 0 \;\text{ and }\; \rho \;\geq\; 0\,. \end{aligned} \tag{4.136}$$

The origin $\mathbf{0}$ is the negative class point with $y = -1$ all other data points have $y^i = 1$.

We observed at the ν-SVM formulation that unbalanced classes are not suited for this task. The remedy for this problem is to up-weight the error ξ^- for the origin by factor of $(l\,\nu)$, which is equivalent to place $(l\,\nu)$ negative class points at the origin.

The constraints say that $\boldsymbol{w}^T\mathbf{0} \;+\; b \;\leq\; -\,\rho \;+\; \xi^-$. The optimal value (reducing ξ^- or equivalently increasing b) is $b \;=\; -\,\rho \;+\; \xi^-$.

We obtain for $(l-1)$ positive examples

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l-1}\xi_i \;+\; b \\ \text{s.t.} \quad & \boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;\geq\; -\,\xi_i,\;\; \xi_i \;\geq\; 0\,. \end{aligned} \tag{4.137}$$

which gives for l positive points and re-scaling of ν the one class support vector optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \;+\; b \\ \text{s.t.} \quad & \boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;\geq\; -\,\xi_i,\;\; \xi_i \;\geq\; 0\,. \end{aligned} \tag{4.138}$$

The Lagrangian of the one class SVM formulation is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) \;=\; \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \;+\; b \;- \tag{4.139}$$
$$\sum_{i=1}^{l}\alpha_i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;+\; \xi_i\right) \;-\; \sum_{i=1}^{l}\mu_i\,\xi_i\;.$$

Setting the derivatives of L with respect to $\boldsymbol{w}$,$\boldsymbol{\xi}$, and b equal to zero gives:

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l}\alpha_i\,\boldsymbol{x}^i \tag{4.140}$$
$$\alpha_i \;=\; \frac{1}{l\,\nu} \;-\; \mu_i \;\leq\; \frac{1}{l\,\nu}$$
$$\sum_{i=1}^{l}\alpha_i \;=\; 1\;.$$

The dual problem for the one-class SVM is

$$\min_{\boldsymbol{\alpha}} \qquad \frac{1}{2}\,\sum_{i,j}\alpha_i\,\alpha_j\;\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \tag{4.141}$$
$$\text{s.t.} \qquad 0 \;\leq\; \alpha_i \;\leq\; \frac{1}{l\,\nu}$$
$$\sum_{i=1}^{l}\alpha_i \;=\; 1\;.$$

The value for b can again be determined from $0 \;<\; \alpha_j \;<\; \frac{1}{l\,\nu}$ as

$$b \;=\; -\;\sum_{i=1}^{l}\alpha_i\;\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \tag{4.142}$$

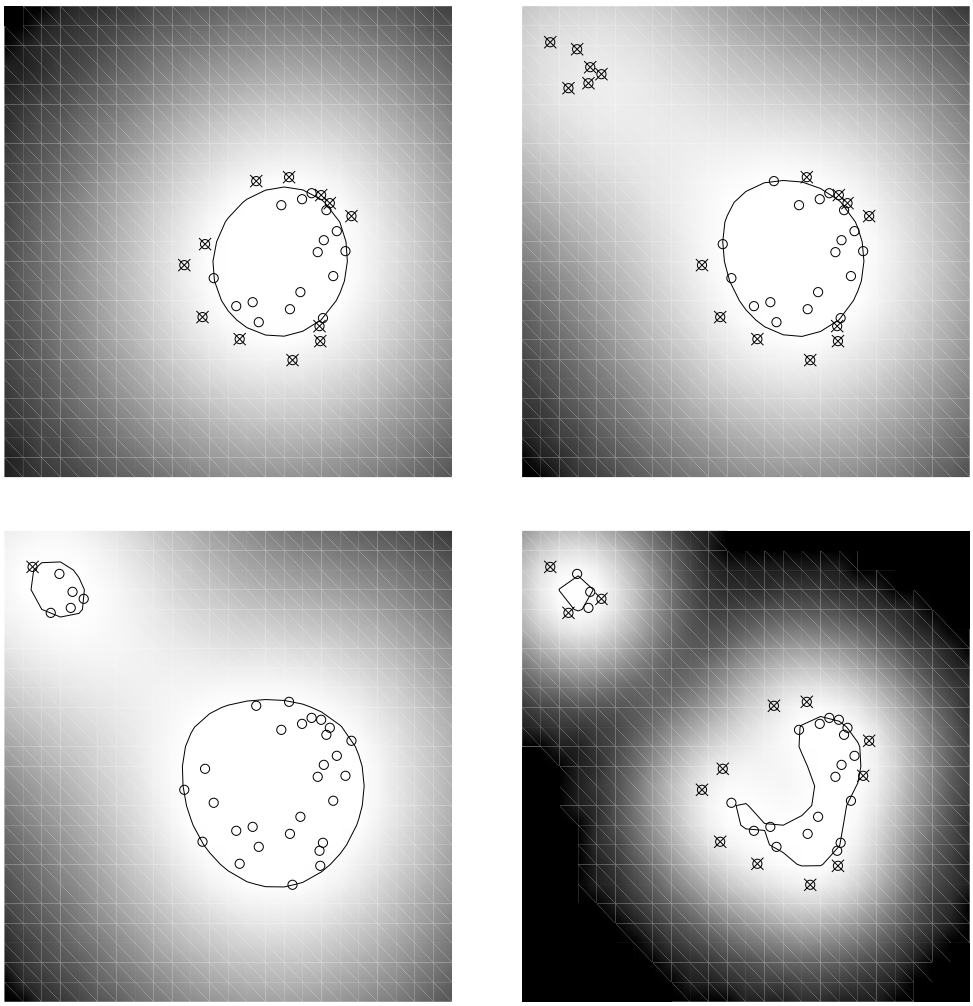
Note for $\nu = 1$ a kernel density estimator is obtained because $\alpha_i = \alpha_j = \frac{1}{l}$ (especially $\nu \leq 1$ must hold).

Again we obtain similar ν-statements:

- ν is an upper bound on the fraction of outliers (points which are negatively labeled).
- ν is a lower bound on the fraction of support vectors.
- Under mild conditions: with probability 1, asymptotically, ν equals both the fraction of support vector and the fraction of outliers.

Worst case error bounds can be given similar to the standard support vector machines.

Figures 4.37 and 4.38 show the single-class SVM applied to toy problems where both ν and the width σ of the Gaussian kernel is varied.



ν, kernel width σ	0.5, 0.5	0.5, 0.5	0.1, 0.5	0.5, 0.1
frac. SVs/OLs	0.54, 0.43	0.59, 0.47	0.24, 0.03	0.65, 0.38
margin $\frac{\rho}{\|\boldsymbol{w}\|}$	0.84	0.70	0.62	0.48

Figure 4.37: A single-class SVM applied to two toy problems (upper left and upper right). The second panel is the second problem with different values for ν and σ, the width of the Gaussian kernel. In the second panel now the data points in the upper left corner are considered. "OLs" means outliers and "SVs" support vectors. [Schölkopf et al., 2001].

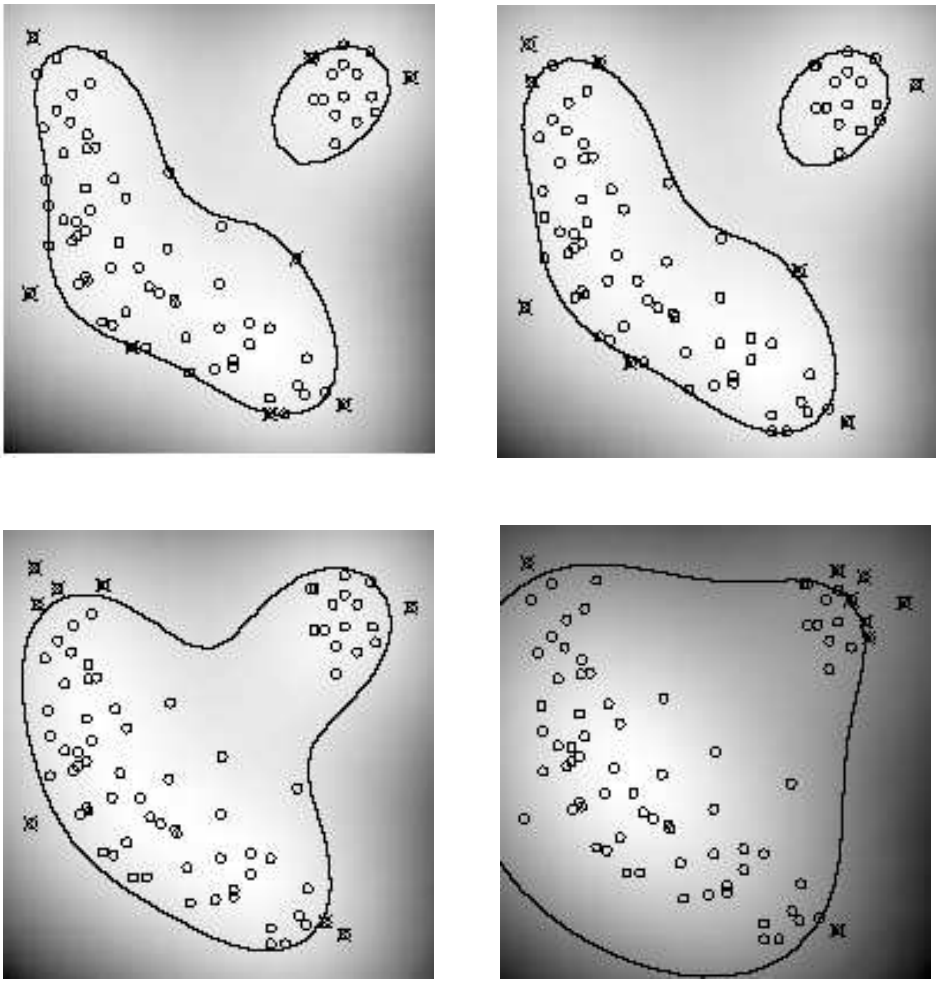


Figure 4.38: A single-class SVM applied to another toy problem with $\sigma = 0.5$ and $\nu \in \{0.1, 0.2, 0.4, 1.0\}$ from upper left to lower right. Note, that $\nu = 1$ (lower right) leads to a kernel density estimate. Copyright © 2001 MIT Press Journals [Schölkopf et al., 2001].

4.13 Least Squares SVM

The C-SVM from eq. (4.32) was

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},b,\boldsymbol{\xi}} & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; C\,\sum\limits_{i=1}^{l}\xi_i \\ \text{s.t.} & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i \\ & \xi_i \;\geq\; 0\,. \end{array} \tag{4.143}$$

The slack variable ξ_i denoted the error. The error is computed according to the 1-norm. However the 2-norm may also be possible.

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},b,\boldsymbol{\xi}} & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; C\,\sum\limits_{i=1}^{l}\xi_i^2 \\ \text{s.t.} & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i\,, \end{array} \tag{4.144}$$

where $\xi_i \;\geq\; 0$ is dropped and the dual variable $\boldsymbol{\mu}$ disappears.

The derivative of the Lagrangian with respect to $\boldsymbol{\xi}$ is now

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} \;=\; 2\,C\,\boldsymbol{\xi} \;-\; \boldsymbol{\alpha} \;=\; \mathbf{0}\,. \tag{4.145}$$

Therefore

$$\xi_i \;=\; \frac{1}{2\,C}\,\alpha_i \tag{4.146}$$

which gives the terms $-\alpha_i\,\xi_i \;=\; -\,\frac{1}{2\,C}\,\alpha_i^2$ and $C\,\xi_i^2 \;=\; \frac{1}{4\,C}\,\alpha_i^2$ summing up to $-\,\frac{1}{4\,C}\,\alpha_i^2$.

The dual is now

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha}} & \frac{1}{2}\,\sum\limits_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j\,\left(\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \;+\; \frac{1}{2\,C}\delta_{ij}\right) \;-\; \sum\limits_{i=1}^{l}\alpha_i \\ \text{s.t.} & 0 \;\leq\; \alpha_i \\ & \sum\limits_{i=1}^{l}\alpha_i\,y^i \;=\; 0\,, \end{array} \tag{4.147}$$

where δ_{ij} is the Kronecker delta which is 1 for $i = j$ and 0 otherwise.

Again be obtain known facts like

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l}\alpha_i\,y^i\,\boldsymbol{x}^i\,. \tag{4.148}$$

Also the offset b can be regularized which removes the quality constraint from the dual. However the problem is no longer translation invariant.

The primal problem formulation is

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{2}\,b^2 \;+\; C\,\sum_{i=1}^{l}\xi_i^2 \\ \text{s.t.} \quad & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i\;. \end{aligned} \tag{4.149}$$

The derivative with respect to b is now

$$\frac{\partial\mathcal{L}}{\partial b} \;=\; b \;-\; \sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0 \tag{4.150}$$

$$\tag{4.151}$$

We have now the terms $\frac{1}{2}\,b^2 \;=\; \frac{1}{2}\sum_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j$ and $-b\,\sum_i y^i\,\alpha_i \;=\; -\sum_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j$ in the Lagrangian which gives together $-\frac{1}{2}\sum_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j$.

The dual is in this case

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\,\sum_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j\,\left(\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \;+\; 1 \;+\; \frac{1}{2\,C}\delta_{ij}\right) \;-\; \sum_{i=1}^{l}\alpha_i \\ \text{s.t.} \quad & 0 \;\leq\; \alpha_i\;. \end{aligned} \tag{4.152}$$

The value b can be computed through

$$b \;=\; \sum_{i=1}^{l}\alpha_i\,y^i\;. \tag{4.153}$$

From the KKT conditions it can be inferred that the optimal solution satisfies

$$\boldsymbol{\alpha}^T\,(\boldsymbol{Q}\,\boldsymbol{\alpha}\,\mathbf{1}) \;=\; 0\;, \tag{4.154}$$

where

$$Q_{ij} \;=\; y^i\,y^j\,\left(\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \;+\; 1 \;+\; \frac{1}{2\,C}\delta_{ij}\right)\;. \tag{4.155}$$

A linear to the solution converging iteration scheme is

$$\boldsymbol{\alpha}^{t+1} \;=\; \boldsymbol{Q}^{-1}\left(\left(\left(\boldsymbol{Q}\,\boldsymbol{\alpha}^t \;-\; \mathbf{1}\right) \;-\; \gamma\,\boldsymbol{\alpha}^t\right)_+ \;+\; \mathbf{1}\right)\;, \tag{4.156}$$

where $(\ldots)_+$ sets all negative components of a vector to zero.

However this algorithm needs the inversion of a matrix which is computational expensive. But the inversion must be made only once. An SMO-like algorithm might be faster.

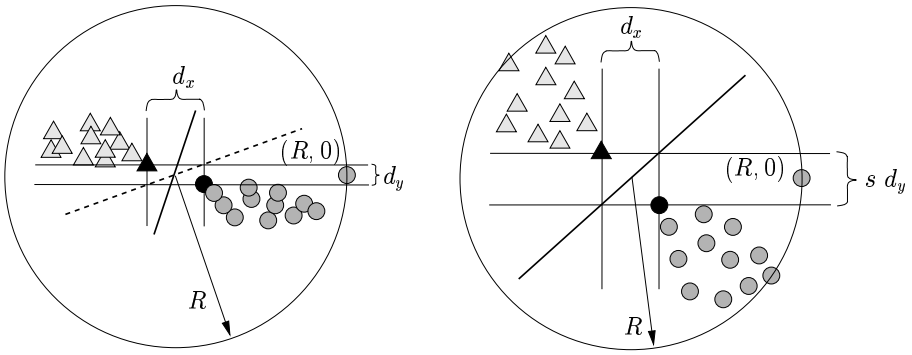


Figure 4.39: LEFT: Data points from two classes (triangles and circles) are separated by the SVM solution. The two support vectors are separated by d_x along the horizontal and by d_y along the vertical axis, from which we obtain the margin $\gamma = \frac{1}{2}\sqrt{d_x^2 + d_y^2}$ and $\frac{R^2}{\gamma^2} = \frac{4\ R^2}{d_x^2 + d_y^2}$. The dashed line indicates the classification boundary of the classifier shown on the right, scaled back by a factor of $\frac{1}{s}$. RIGHT: The same data but scaled along the vertical axis by the factor s. The data points still lie within the sphere of radius R. The solid line denotes the support vector hyperplane, whose back-scaled version is also shown on the left (dashed line). We obtain $\gamma = \frac{1}{2}\sqrt{d_x^2 + s^2\ d_y^2}$ and $\frac{R^2}{\gamma^2} = \frac{4\ R^2}{d_x^2 + s^2\ d_y^2}$. For $s \neq 1$ and $d_y \neq 0$ both the margin γ and the term $\frac{R^2}{\gamma^2}$ change with scaling.

4.14 Potential Support Vector Machine

Both the SVM solution and the bounds on the generalization from the last chapter are not invariant under linear transformations like scaling.

The dependence on scaling is illustrated in Fig. 4.39, where the optimal hyperplane is not invariant under scaling, hence predictions of class labels may change if the data is re-scaled before learning. We found in previous chapter in Subsection 3.6.5 in theorem 3.10 that the VC-dimension was bounded by

$$d_{\mathrm{VC}} \leq \frac{R^2}{\gamma^2}\ . \tag{4.157}$$

This bound is also not scale invariant as shown in Fig. 4.39.

However if we given the length in meters, centimeters, or millimeters scales the data. Often the output of measurement devices must be scaled to make different devices compatible. For many measurements the outputs are scaled for convenient use and do not have any meaning. If the classifier depends on then scale factors: which scale factors should be used? Therefore scaling invariance seems to be desirable for obtain robust classification.

For the Potential Support Vector Machine (P-SVM) the training data is scaled such that the margin γ remains constant while the new radius $R_{\min}$ of the sphere containing all training data becomes as small as possible. In principle all data points are projected onto a line in direction $\boldsymbol{w}$. All directions orthogonal to the normal vector $\boldsymbol{w}$ of the hyperplane with maximal margin γ are scaled to zero.

For simplicity we assume that the data is centered at the origin. We then obtain for the new radius:

$$R_{\min} \;=\; \max_i \left| \frac{\boldsymbol{w}^T}{\|\boldsymbol{w}\|} \boldsymbol{x}^i \right| \;. \tag{4.158}$$

The new complexity measure instead of the margin would be

$$\frac{R_{\min}^2}{\gamma^2} \;=\; \frac{1}{\|\boldsymbol{w}\|^2} \left(\max_i \left| \boldsymbol{w}^T \boldsymbol{x}^i \right| \right)^2 \|\boldsymbol{w}\|^2 \;=\; \max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;. \tag{4.159}$$

Note that

$$\max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;\leq\; \frac{R^2}{\gamma^2} \tag{4.160}$$

for $R = \max_i \left\| \boldsymbol{x}^i \right\|$ follows from the Cauchy-Schwarz inequality. Therefore the new bond is tighter than the margin bound.

Because the objective $\max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2$ is inconvenient to optimize, an upper bound on these objective is optimized instead:

$$\max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;\leq\; \sum_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;=\; \boldsymbol{w}^T \boldsymbol{X}\, \boldsymbol{X}^T \boldsymbol{w} \;. \tag{4.161}$$

Using the objective

$$\boldsymbol{w}^T \boldsymbol{X}\, \boldsymbol{X}^T \boldsymbol{w} \;=\; \left\| \boldsymbol{X}^T \boldsymbol{w} \right\|^2 \;. \tag{4.162}$$

instead of $\boldsymbol{w}^T \boldsymbol{w}$ is equivalent to first sphering the data and then applying a standard SVM. This approach is called "sphered support vector machine" (S-SVM).

Minimizing the new objective leads to normal vectors which tend to point in directions of low variance of the data. For sphered data the covariance matrix is given by $\frac{1}{l}\, \boldsymbol{X}\, \boldsymbol{X}^T \;=\; \boldsymbol{I}$ and we recover the classical SVM because scaling of the objective does not matter. The new objective leads to separating hyperplanes which are invariant to linear transformations of the data. Consequently, the bounds and the performance of the derived classifier no longer depend on scale factors. Note, that the kernel trick carries over to the S-SVM.

The S-SVM is computational expensive to optimize, because in various expressions $\left(\boldsymbol{X}\, \boldsymbol{X}^T \right)^{-1}$ appear.

Let us assume we have a dot product matrix

$$K_{ij} \;=\; \left(\boldsymbol{x}^i \right)^T \boldsymbol{z}^j \tag{4.163}$$

that is

$$\boldsymbol{K} \;=\; \boldsymbol{X}^T\, \boldsymbol{Z} \;. \tag{4.164}$$

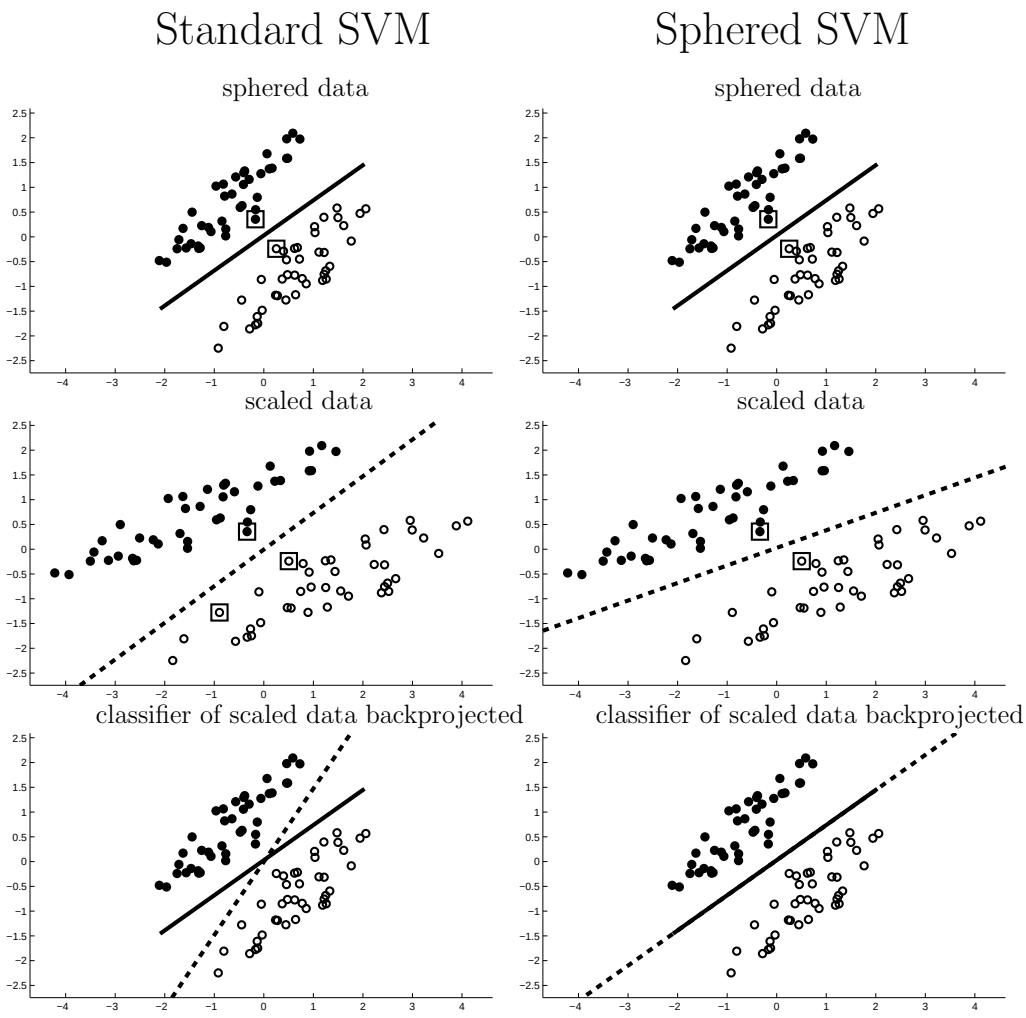


Figure 4.40: The standard SVM (left) in contrast to the sphered SVM (right). Scaling the data and back-scaling gives the original S-SVM solution in contrast to the standard SVM where the solutions differ.

The vectors $\left(\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^N\right)$ are called *complex features* summarized in the matrix $\boldsymbol{Z}$ and are used to describe the separating hyperplane.

Note that the separating hyperplane can be described by any set of points and not only by the data points which must be classified.

Analog to the least square SVM, the residual error r_i for $\boldsymbol{x}^i$ is defined as

$$r_i \;=\; \boldsymbol{w}^T \boldsymbol{x}^i \;+\; b \;-\; y^i \,. \tag{4.165}$$

If we assume

$$\boldsymbol{w} \;=\; \sum_{j=1}^{N} \alpha_j \boldsymbol{z}^j \tag{4.166}$$

then

$$r_i \;=\; \sum_{j=1}^{N} \alpha_j K_{ij} \;+\; b \;-\; y^i \,. \tag{4.167}$$

that is r_i is linear in K_{ij}.

The empirical error, the mean square error, is defined as

$$R_{\text{emp}} \;=\; \frac{1}{l} \sum_{i=1}^{l} r_i^2 \tag{4.168}$$

and it is minimal if

$$\frac{\partial R_{\text{emp}}}{\partial \alpha_j} \;=\; 2\,\frac{1}{l} \sum_{i=1}^{l} r_i \, K_{ij} \;=\; 0 \tag{4.169}$$

for every $1 \;\leq\; j \;\leq\; N$.

This motivates a new set of constraints

$$\boldsymbol{K}^T \boldsymbol{r} \;=\; \boldsymbol{K}^T \left(\boldsymbol{X}^T \, \boldsymbol{w} \;+\; b\mathbf{1} \;-\; \boldsymbol{y}\right) \;=\; \mathbf{0} \,, \tag{4.170}$$

which an optimal classifier must fulfill.

Because the error is quadratic in $\boldsymbol{\alpha}$ the absolute value of $\frac{\partial R_{\text{emp}}}{\partial \alpha_j}$ gives the distance to the optimum. A threshold ϵ may threshold the difference to the optimum and the constraints are

$$\begin{aligned} &\boldsymbol{K}^T \left(\boldsymbol{X}^T \, \boldsymbol{w} \;+ b\mathbf{1} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\, \mathbf{1} \;\leq\; \mathbf{0} \,, \\ &\boldsymbol{K}^T \left(\boldsymbol{X}^T \, \boldsymbol{w} \;+ b\mathbf{1} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\, \mathbf{1} \;\geq\; \mathbf{0} \,. \end{aligned} \tag{4.171}$$

Because different $\boldsymbol{z}^j$ may have different length and therefore the $K_{\cdot j}$ have different variance. That means, the α_j scale differently and are not comparable and in terms of absolute values. To

make the α_j comparable to one another, the vectors $K_{.j}$ are normalized to zero mean and unit variance:

$$\sum_{i=1}^{m} K_{ij} \;=\; 0 \tag{4.172}$$

$$\frac{1}{m}\sum_{i=1}^{m} K_{ij}^2 \;=\; 1\,. \tag{4.173}$$

in a preprocessing step. That means

$$\boldsymbol{K}^T\mathbf{1} \;=\; 0\,. \tag{4.174}$$

The constraints simplify to

$$\begin{aligned} &\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;\leq\; \mathbf{0}\,, \\ &\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;\geq\; \mathbf{0}\,. \end{aligned} \tag{4.175}$$

The value for ϵ should be adapted to the level of measurement noise, and should increase if the noise becomes larger.

After the introduction of the slack variables we obtain the primal optimization problem of the Potential Support Vector Machine (P-SVM),

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}^+,\boldsymbol{\xi}^-} \quad & \frac{1}{2}\,\|\boldsymbol{X}^T\,\boldsymbol{w}\|_2^2 \;+\; C\,\mathbf{1}^T\left(\boldsymbol{\xi}^+ \;+\; \boldsymbol{\xi}^-\right) \\ \text{s.t.} \quad & \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;+\; \boldsymbol{\xi}^+ \;\geq\; \mathbf{0} \\ & \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;-\; \boldsymbol{\xi}^- \;\leq\; \mathbf{0} \\ & \boldsymbol{\xi}^+ \;\geq\; \mathbf{0},\;\; \boldsymbol{\xi}^- \;\geq\; \mathbf{0}\;. \end{aligned} \tag{4.176}$$

The Lagrangian L is given by

$$\begin{aligned} L \;=\; & \frac{1}{2}\,\boldsymbol{w}^T\,\boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;+\; C\,\mathbf{1}^T\left(\boldsymbol{\xi}^+ \;+\; \boldsymbol{\xi}^-\right) \;- \\ & \left(\boldsymbol{\alpha}^+\right)^T\,\left(\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;+\; \boldsymbol{\xi}^+\right) \;+ \\ & \left(\boldsymbol{\alpha}^-\right)^T\,\left(\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;-\; \boldsymbol{\xi}^-\right) \;- \\ & \left(\boldsymbol{\mu}^+\right)^T\,\boldsymbol{\xi}^+ \;-\; \left(\boldsymbol{\mu}^-\right)^T\,\boldsymbol{\xi}^-\;. \end{aligned} \tag{4.177}$$

The optimality conditions require that the following derivatives with respect to the primal variables of the Lagrangian L are zero:

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} \;&=\; \boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{X}\,\boldsymbol{K}\,\boldsymbol{\alpha} \;=\; \mathbf{0}\,, \\ \frac{\partial L}{\partial \boldsymbol{\xi}^+} \;&=\; C\mathbf{1} \;-\; \boldsymbol{\alpha}^+ \;-\; \boldsymbol{\mu}^+ \;=\; \mathbf{0}\,, \\ \frac{\partial L}{\partial \boldsymbol{\xi}^+} \;&=\; C\mathbf{1} \;-\; \boldsymbol{\alpha}^- \;-\; \boldsymbol{\mu}^- \;=\; \mathbf{0}\;. \end{aligned} \tag{4.178}$$

In order to ensure the first condition

$$\boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;=\; \boldsymbol{X}\,\boldsymbol{K}\,\boldsymbol{\alpha} \;=\; \boldsymbol{X}\,\boldsymbol{X}^T\boldsymbol{Z}\,\boldsymbol{\alpha} \tag{4.179}$$

one solution is

$$\boldsymbol{w} \;=\; \boldsymbol{Z}\,\boldsymbol{\alpha} \;=\; \sum_{j=1}^{N} \alpha_j\,\boldsymbol{z}^j\;. \tag{4.180}$$

In contrast to the standard SVM expansion of $\boldsymbol{w}$ by its support vectors $\boldsymbol{x}$, the weight vector $\boldsymbol{w}$ is now expanded using the complex features $\boldsymbol{z}$ which serve as the support vectors in this case.

The last two conditions are fulfilled if

$$\boldsymbol{\alpha}^+ \;\leq\; C\mathbf{1} \quad\text{and}\quad \boldsymbol{\alpha}^- \;\leq\; C\mathbf{1}\;. \tag{4.181}$$

The dual optimization problem of the P-SVM is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \quad & \frac{1}{2}\left(\boldsymbol{\alpha}^+ \,-\, \boldsymbol{\alpha}^-\right)^T \boldsymbol{K}^T\,\boldsymbol{K}\,\left(\boldsymbol{\alpha}^+ \,-\, \boldsymbol{\alpha}^-\right) \,- \\ & \boldsymbol{y}^T\,\boldsymbol{K}\,\left(\boldsymbol{\alpha}^+ \,-\, \boldsymbol{\alpha}^-\right) \,+\, \epsilon\,\mathbf{1}^T\left(\boldsymbol{\alpha}^+ \,+\, \boldsymbol{\alpha}^-\right) \\ \text{s.t.} \quad & \mathbf{0} \;\leq\; \boldsymbol{\alpha}^+ \;\leq\; C\mathbf{1}\,, \\ & \mathbf{0} \;\leq\; \boldsymbol{\alpha}^- \;\leq\; C\mathbf{1}\,. \end{aligned} \tag{4.182}$$

To determine b we find its optimal value by setting the derivative of the empirical error to zero:

$$\frac{\partial R_{\text{emp}}}{\partial b} \;=\; 2\,\frac{1}{l}\sum_{i=1}^{l} r_i \;=\; 0\,, \tag{4.183}$$

therefore

$$\begin{aligned} & b \;=\; \frac{1}{l}\sum_{i=1}^{l} y^i \;-\; \boldsymbol{w}^T\boldsymbol{x}^i \;= \\ & \frac{1}{l}\sum_{i=1}^{l} y^i \;-\; \frac{1}{l}\boldsymbol{w}^T\boldsymbol{X}\mathbf{1} \;= \\ & \frac{1}{l}\sum_{i=1}^{l} y^i \;-\; \frac{1}{l}\boldsymbol{\alpha}^T\boldsymbol{K}^T\mathbf{1} \;= \\ & \frac{1}{l}\sum_{i=1}^{l} y^i\,. \end{aligned} \tag{4.184}$$

The discriminant function is

$$\sum_{j=1}^{N} \alpha_j\,(\boldsymbol{z}^j)^T\boldsymbol{x} \;+\; b \;=\; \sum_{j=1}^{N} \alpha_j\,K_{xj} \;+\; b\,. \tag{4.185}$$

At the optimum, the value of the objective is

$$\boldsymbol{w}^T \boldsymbol{X}\, \boldsymbol{X}^T \boldsymbol{w} \;=\; \left(\boldsymbol{\alpha}^+ \,-\, \boldsymbol{\alpha}^-\right)^T \boldsymbol{K}^T \boldsymbol{y} \;- \\ \epsilon\, \mathbf{1}^T \left(\boldsymbol{\alpha}^+ \,+\, \boldsymbol{\alpha}^-\right) \,-\, \boldsymbol{\alpha}^+\, \boldsymbol{\xi}^+ \,+\, \boldsymbol{\alpha}^- \boldsymbol{\xi}^- \,. \tag{4.186}$$

It can be seen that increasing ϵ or allowing larger values of $\boldsymbol{\xi}^+$ or $\boldsymbol{\xi}^-$ reduces the complexity term.

Note, that the $\boldsymbol{K}$ is not required to be positive semi-definite or even square, because only the quadratic part $\boldsymbol{K}^T\, \boldsymbol{K}$, which is always positive definite, appears in the objective function.

The matrix $\boldsymbol{K} \;=\; \boldsymbol{X}^T \boldsymbol{X}$, where $\boldsymbol{z}^j \;= \boldsymbol{x}^j$ and $l = N$, or $\boldsymbol{K} \;=\; \boldsymbol{X}$, where $\boldsymbol{z}^j \;= \boldsymbol{e}^j$ ($\boldsymbol{Z} \;= \boldsymbol{I}$), or a Gram matrix, where $K_{ij} \;=\; k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ ($\boldsymbol{x}^i_\phi \;=\; \phi \boldsymbol{x}^i$ and $\boldsymbol{z}^j_\phi \;=\; \phi \boldsymbol{x}^j$). The Gram matrix must not be positive definite. *The P-SVM can also be applied for kernels which are not positive definite.*

Most importantly, the P-SVM can be used for **feature selection** because the α_j weight the features $\boldsymbol{z}^j$. The optimal separating hyperplane is expressed through support features. The number of features which are selected can be controlled by the hyperparameter ϵ.

The matrix $\boldsymbol{K}$ can also be a measurement matrix because measurements can be expressed as dot products. For example, the matrix $\boldsymbol{K}$ is identified with the matrix obtained by a micro array experiment and $\boldsymbol{x}^i$ are tissue samples and $\boldsymbol{z}^j$ are genes. In this case the value $K_{ij} \;=\; \left(\boldsymbol{x}^i\right)^T \boldsymbol{z}^j$ is the expression of the j-th gene in the i-th tissue sample.

Therefore the P-SVM is an ideal tool for gene selection.

4.15 SVM Optimization and SMO

4.15.1 Convex Optimization

Convex Problems.

The optimization problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & f(\boldsymbol{x}) \\ \text{s.t.} & \forall_i : \;\; c_i(\boldsymbol{x}) \;\leq\; 0 \\ & \forall_j : \;\; e_j(\boldsymbol{x}) \;=\; 0 \,, \end{array} \tag{4.187}$$

where f, c_i, and e_j are convex functions has as solution a convex set and if f is strictly convex then the solution is unique. This problem class is call "constraint convex minimization".

Note, that all SVM optimization problems we encountered so far are constraint convex minimization problems.

The Lagrangian is

$$L(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\mu}) \;=\; f(\boldsymbol{x}) \;+\; \sum_i \alpha_i \; c_i(\boldsymbol{x}) \;+\; \sum_j \mu_j \; e_j(\boldsymbol{x}) \,, \tag{4.188}$$

where $\alpha_i \;\geq\; 0$. The variables $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ are called "Lagrange multipliers". Note, that the Lagrangian can be build also for non-convex functions.

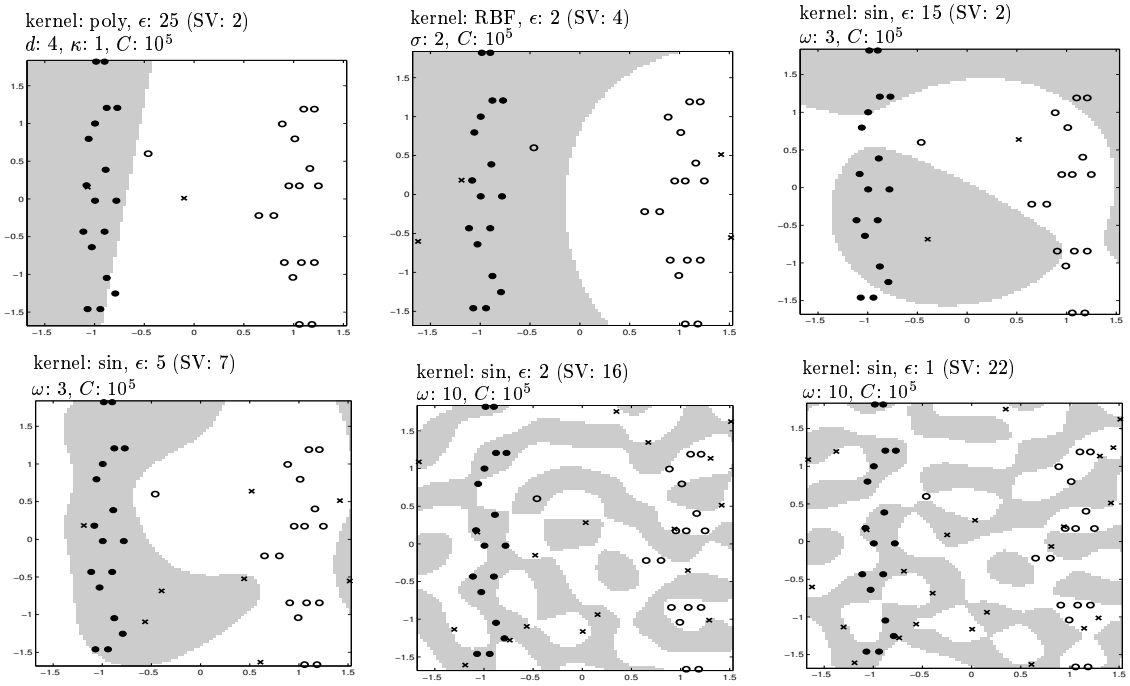


Figure 4.41: Application of the P-SVM method to a toy classification problem. Objects x and complex features z are in a two-dimensional space. The feature vectors $\boldsymbol{x}$ for 34 objects, 17 from each class, were chosen as shown in the figure (open and solid circles), and 50 feature vectors (complex features) were generated randomly and uniformly from the interval $[-2, 2] \times [-2, 2]$. The figures show the resulting P-SVM classifier for the polynomial kernel $k\left(\boldsymbol{x}^i, \boldsymbol{z}^j\right) = \left(\left\langle \boldsymbol{x}^i, \boldsymbol{z}^j \right\rangle + \kappa\right)^d$ (poly), the RBF kernel $k\left(\boldsymbol{x}^i, \boldsymbol{z}^j\right) = \exp(-\frac{1}{\sigma^2} \left\|\boldsymbol{x}^i - \boldsymbol{z}^j\right\|^2$ (RBF), and the sine-kernel $k\left(\boldsymbol{x}^i, \boldsymbol{z}^j\right) = \sin(\theta \left\|\boldsymbol{x}^i - \boldsymbol{z}^j\right\|)$ (sine). Gray and white regions indicate areas of class 1 and class 2 membership as predicted by the selected classifiers and crosses indicate support features. Parameters are given in the figure.

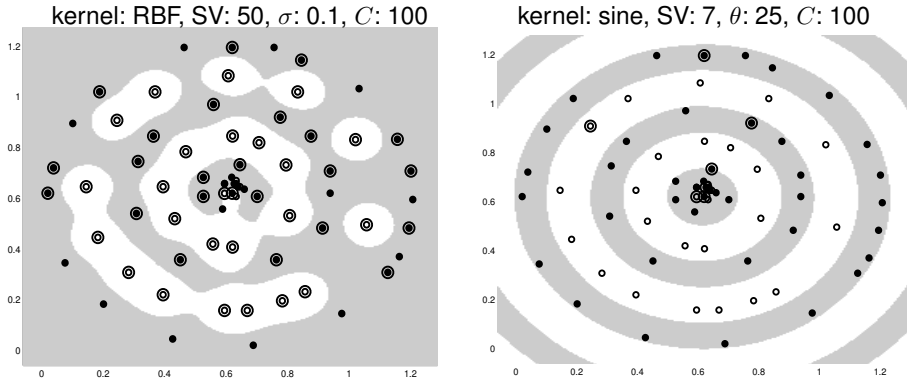


Figure 4.42: Application of the P-SVM method to another toy classification problem. Objects are described by two-dimensional feature vectors $\boldsymbol{x}$, and 70 feature vectors were generated of which 28 belong to class 1 (open circles) and 42 belong to class 2 (solid circles). A Gram matrix was constructed using the positive definite RBF kernel (left) and the indefinite sine-kernel $k\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) = \sin\left(\theta \left\|\boldsymbol{x}^i - \boldsymbol{x}^j\right\|\right)$ (right). White and gray indicate regions of class 1 and class 2 membership. Circled data indicate support vectors. Parameters are given in the figure.

Assume that a feasible solution exists then the following statements are equivalent, where $\boldsymbol{x}$ denotes a feasible solution:

(a) an $\boldsymbol{x}$ exists with $c_i(\boldsymbol{x}) < 0$ for all i (Slater's condition),

(b) an $\boldsymbol{x}$ and $\alpha_i \geq 0$ exist such that $\sum_i \alpha_i \, c_i(\boldsymbol{x}) \leq 0$ (Karlin's condition).

Above statements (a) or (b) follow from the following statement

(c) there exist at least two feasible solutions and a feasible $\boldsymbol{x}$ such that all c_i are strictly convex at $\boldsymbol{x}$ w.r.t. the feasible set (strict constraint qualification).

The saddle point condition of Kuhn-Tucker:
If one of (a) - (c) holds then

$$L\left(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}, \boldsymbol{\mu}\right) \leq L\left(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}}\right) \leq L\left(\boldsymbol{x}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}}\right) . \tag{4.189}$$

is necessary and sufficient for $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})$ being a solution to the optimization problem. Note, that "sufficient" also holds for non-convex functions.

The optimal Lagrange multipliers $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\mu}}$ maximize L with $\boldsymbol{x}$ fixed to the optimal solution $\hat{\boldsymbol{x}}$. The optimal $\hat{\boldsymbol{x}}$ minimize L with Lagrange multipliers fixed to their optimal solution $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\mu}}$.

All $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})$ which fulfill the saddle point condition eq. (4.189) are a solution to the optimization problem.

To see that assume that $(\hat{x}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})$ satisfy the saddle point condition eq. (4.189). From $L\left(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}, \boldsymbol{\mu}\right) \leq L\left(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}}\right)$ it follows that

$$\sum_i \left(\alpha_i - \hat{\alpha}_i\right) c_i(\hat{\boldsymbol{x}}) + \sum_j \left(\mu_j - \hat{\mu}_j\right) e_j(\hat{\boldsymbol{x}}) \leq 0 . \tag{4.190}$$

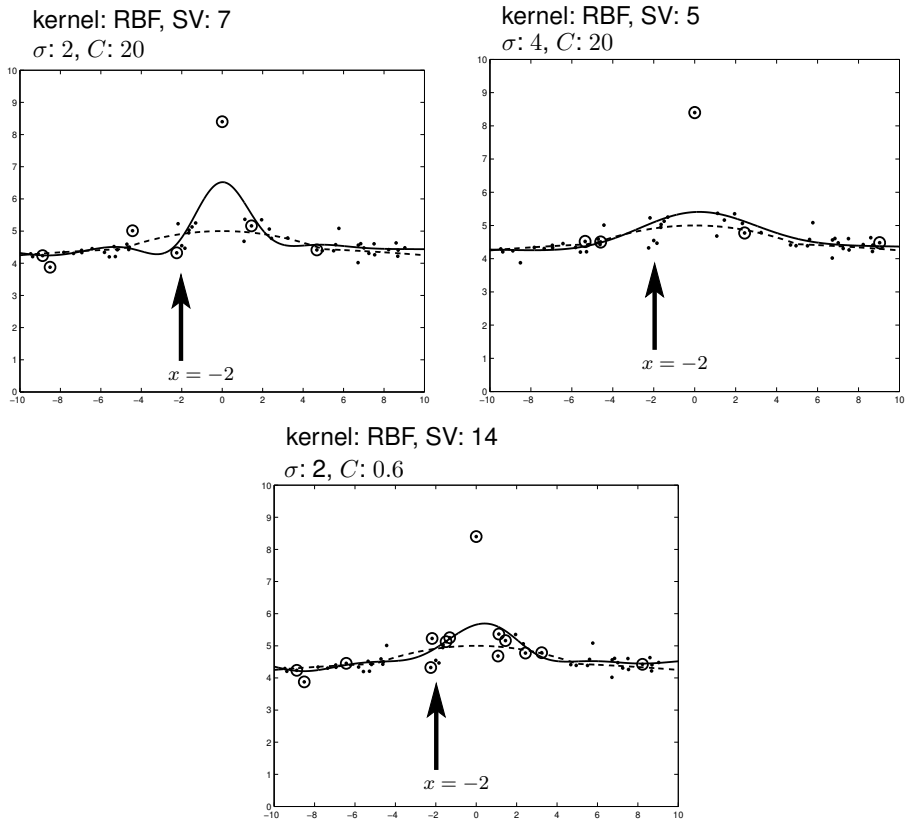


Figure 4.43: Application of the P-SVM method to a toy regression problem. Objects (small dots), described by the x-coordinate, were generated by randomly choosing points from the true function (dashed line) and adding Gaussian noise with mean 0 and standard deviation 0.2 to the y-component of each data point. One outlier was added by hand at $x = 0$. A Gram matrix was then generated using an RBF-kernel with width σ. The solid lines show the regression result. Circled dots indicate support vectors. Parameters are given in the figure. The arrows in the figures mark $x = -2$, where the effect of local vs. global smoothing can be seen.

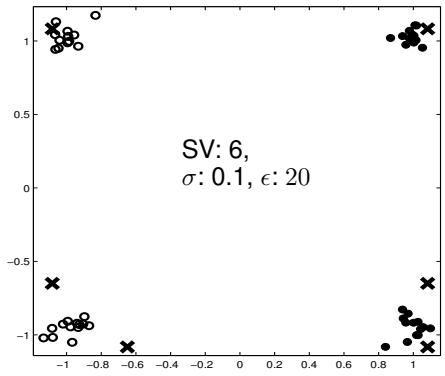


Figure 4.44: Application of the P-SVM method to a toy feature selection problem for a classification task. Objects are described by two-dimensional vectors $\boldsymbol{x}$ and complex features by $\boldsymbol{z}$. Vectors $\boldsymbol{x}$ for 50 objects, 25 from each class (open and solid circles), were generated randomly by choosing a center from $\{(1,1),(1,-1),(-1,1),(-1,-1)\}$ with equal probability, then adding to each coordinate of the center a random value, which stems from a Gaussian with mean 0 and standard deviation 0.1. Feature vectors $\boldsymbol{z}$ (complex features) were generated randomly and uniformly from the interval $[-1.2,1.2]\times[-1.2,1.2]$. An RBF-kernel $\exp\left(-\frac{1}{2\,\sigma^2}\,\|\boldsymbol{x}^i\,-\,\boldsymbol{z}^j\|^2\right)$ with width $\sigma\,=\,0.2$ is applied to each pair $\left(\boldsymbol{x}^i,\boldsymbol{z}^j\right)$ of object and complex feature in order to construct the data matrix $\boldsymbol{K}$. Black crosses indicate the location of features selected by the P-SVM method.

If we set all $\mu_j=\hat{\mu}_j$ and $\alpha_i=\hat{\alpha}_i$ except $\alpha_k=\hat{\alpha}_k\,+\,1$ then we obtain $c_k(\hat{\boldsymbol{x}})\,\leq\,0$ which shows the $\hat{\boldsymbol{x}}$ fulfills the constraints. The equality constraint $e_i(\boldsymbol{x})\,=\,0$ can be replaced by constraints $e_i(\boldsymbol{x})\,\leq\,0$ and $e_i(\boldsymbol{x})\,\geq\,0$. From both constraints follows $0\,\leq\,e_k(\hat{\boldsymbol{x}})\,\leq\,0$, therefore, $e_k(\hat{\boldsymbol{x}})\,=\,0$ (here we can introduce μ^+ and μ^- and set $\mu_k=\mu_k^+\,-\,\mu_k^-$).

If we set all $\mu_j=\hat{\mu}_j$ and $\alpha_i=\hat{\alpha}_i$ except $\alpha_k=0$ then we obtain $\hat{\alpha}_k\;c_k(\hat{\boldsymbol{x}})\,\geq\,0$. Because $\hat{\alpha}_k\,\geq\,0$ and from above $c_k(\hat{\boldsymbol{x}})\,\leq\,0$ we have $\hat{\alpha}_k\;c_k(\hat{\boldsymbol{x}})\,\leq\,0$. It follows that

$$\hat{\alpha}_i\;c_i(\hat{\boldsymbol{x}})\,=\,0 \tag{4.191}$$

and analog

$$\hat{\mu}_j\;e_j(\hat{\boldsymbol{x}})\,=\,0\,. \tag{4.192}$$

These conditions are called "Karush-Kuhn-Tucker" conditions or KKT conditions.

For differentiable problems the minima and maxima can be determined.

Theorem 4.6 (KKT and Differentiable Convex Problems)
A solution to the problem eq. (4.187) with convex, differentiable f, c_i, and e_j is given by $\hat{\boldsymbol{x}}$ if

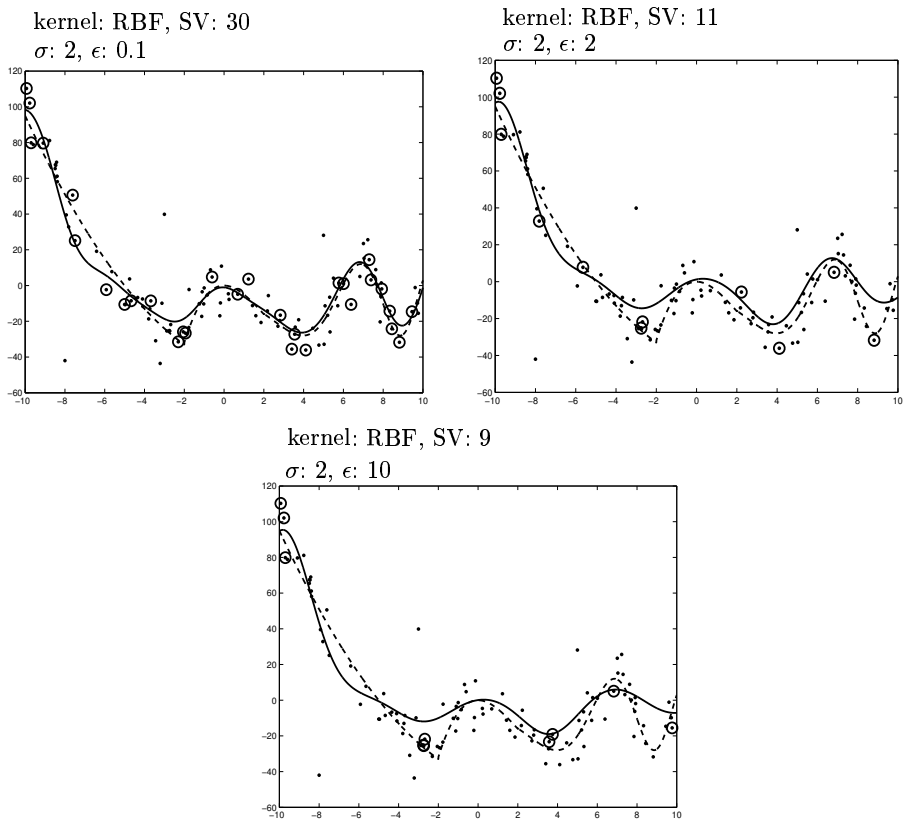


Figure 4.45: Application of the P-SVM to a toy feature selection problem for a regression task. 100 data points are generated from the true function (dashed line) by randomly and uniformly choosing data points from the true function and adding Gaussian noise with mean 0 and standard deviation 10 to the function value. A Gram matrix was constructed using an RBF-kernel with width $\sigma = 2$. The figure shows the P-SVM regression functions (solid lines) and the selected support vectors (circled dots). Parameters are given in the figure.

$\hat{\alpha}_i \geq 0$ *and* $\hat{\mu}_j$ *exist which satisfy:*

$$\frac{\partial L(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})}{\partial \boldsymbol{x}} = \frac{\partial f(\hat{\boldsymbol{x}})}{\partial \boldsymbol{x}} + \sum_i \hat{\alpha}_i \frac{\partial c_i(\hat{\boldsymbol{x}})}{\partial \boldsymbol{x}} + \sum_j \hat{\mu}_j \frac{\partial e_j(\hat{\boldsymbol{x}})}{\partial \boldsymbol{x}} = 0 \tag{4.193}$$

$$\frac{\partial L(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})}{\partial \alpha_i} = c_i(\hat{\boldsymbol{x}}) \leq 0 \tag{4.194}$$

$$\frac{\partial L(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\mu}})}{\partial \mu_j} = e_j(\hat{\boldsymbol{x}}) = 0 \tag{4.195}$$

$$\forall_i : \ \hat{\alpha}_i \, c_i(\hat{\boldsymbol{x}}) = 0 \tag{4.196}$$

$$\forall_j : \ \hat{\mu}_j \, e_j(\hat{\boldsymbol{x}}) = 0 \tag{4.197}$$

For all $\boldsymbol{x}$,$\boldsymbol{\alpha}$ and μ for which eq. (4.193) to eq. (4.195) are fulfilled we have

$$f(\boldsymbol{x}) \geq f(\hat{\boldsymbol{x}}) \geq f(\boldsymbol{x}) + \sum_i \alpha_i \, c_i(\boldsymbol{x}) \, , \tag{4.198}$$

note that $e_j(\boldsymbol{x}) = 0$.

The dual optimization problem (Wolfe's dual) to the optimization problem eq. (4.187) is

$$\begin{aligned} \max_{\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\mu}} \quad & f(\boldsymbol{x}) + \sum_i \alpha_i \, c_i(\boldsymbol{x}) + \sum_j \mu_j \, e_j(\boldsymbol{x}) \\ \text{s.t.} \quad & \forall_i : \ \alpha_i \geq 0 \\ & \frac{\partial L(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial \boldsymbol{x}} = 0 \, . \end{aligned} \tag{4.199}$$

The solutions of the dual eq. (4.199) are the solutions of the primal eq. (4.187). If $\frac{\partial L(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial \boldsymbol{x}} = 0$ can be solved for $\boldsymbol{x}$ and inserted into the dual, then we obtain a maximization problem in $\boldsymbol{\alpha}$.

Linear Programs.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A} \, \boldsymbol{x} + \boldsymbol{d} \leq \boldsymbol{0} \, , \end{aligned} \tag{4.200}$$

where $\boldsymbol{A} \, \boldsymbol{x} + \boldsymbol{d} \leq \boldsymbol{0}$ means that for all i: $\sum_{j=1}^{l} A_{ij} \, x^j + d_j \leq 0$.

The Lagrangian is

$$L = \boldsymbol{c}^T \boldsymbol{x} + \boldsymbol{\alpha}^T (\boldsymbol{A} \, \boldsymbol{x} + \boldsymbol{d}) \, . \tag{4.201}$$

The optimality conditions are

$$\frac{\partial L}{\partial \boldsymbol{x}} = \boldsymbol{A}^T\boldsymbol{\alpha} + \boldsymbol{c} = 0 \tag{4.202}$$
$$\frac{\partial L}{\partial \boldsymbol{\alpha}} = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d} \leq \boldsymbol{0} \tag{4.203}$$
$$\boldsymbol{\alpha}^T(\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d}) = \boldsymbol{0} \tag{4.204}$$
$$\boldsymbol{\alpha} \geq \boldsymbol{0}\,. \tag{4.205}$$

The dual formulation is after inserting $\boldsymbol{A}^T\boldsymbol{\alpha} + \boldsymbol{c} = \boldsymbol{0}$ into the Lagrangian:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \boldsymbol{d}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{A}^T\,\boldsymbol{\alpha} + \boldsymbol{c} = \boldsymbol{0} \\ & \boldsymbol{\alpha} \geq \boldsymbol{0}\,. \end{aligned} \tag{4.206}$$

We compute the dual of the dual. We first make a minimization problem by using $-\,\boldsymbol{d}^T\boldsymbol{\alpha}$ as objective and also use $-\,\boldsymbol{\alpha} \leq 0$ as well as $-\,\boldsymbol{A}^T\,\boldsymbol{\alpha} - \boldsymbol{c} = \boldsymbol{0}$ is with Lagrange multiplier $\boldsymbol{x}'$ for the equality constraints $-\,\boldsymbol{A}^T\,\boldsymbol{\alpha} - \boldsymbol{c} = \boldsymbol{0}$ and $\boldsymbol{\mu}$ for $-\boldsymbol{\alpha} \leq 0$. The dual of the dual is after again transforming it into a minimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\mu}} \quad & \boldsymbol{c}^T\boldsymbol{x}' \\ \text{s.t.} \quad & \boldsymbol{A}\,\boldsymbol{x}' + \boldsymbol{d} + \boldsymbol{\mu} = \boldsymbol{0} \\ & \boldsymbol{\mu} \geq \boldsymbol{0}\,. \end{aligned} \tag{4.207}$$

Because $\boldsymbol{\mu}$ do not influence the objective, we can chose them free. Therefore we obtain again the primal eq. (4.187) because we only have to ensure $\boldsymbol{A}\,\boldsymbol{x}' + \boldsymbol{d} \leq \boldsymbol{0}$.

Quadratic Programs.

The primal quadratic problem is

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d} \leq \boldsymbol{0}\,, \end{aligned} \tag{4.208}$$

where $\boldsymbol{K}$ is strictly positive definite (implying that $\boldsymbol{K}^{-1}$ exists).

The Lagrangian is

$$L(\boldsymbol{x},\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{\alpha}^T(\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d})\,. \tag{4.209}$$

The optimality conditions are

$$\frac{\partial L}{\partial \boldsymbol{x}} = \boldsymbol{K}\,\boldsymbol{x} + \boldsymbol{A}^T\boldsymbol{\alpha} + \boldsymbol{c} = 0 \tag{4.210}$$
$$\frac{\partial L}{\partial \boldsymbol{\alpha}} = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d} \leq \boldsymbol{0} \tag{4.211}$$
$$\boldsymbol{\alpha}^T(\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d}) = \boldsymbol{0} \tag{4.212}$$
$$\boldsymbol{\alpha} \geq \boldsymbol{0}\,. \tag{4.213}$$

The first equation is used to substitute $\boldsymbol{x}$ in the Lagrangian:

$$
\begin{aligned}
L(\boldsymbol{x}, \boldsymbol{\alpha}) \;&=\; \frac{1}{2}\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \;+\; \boldsymbol{c}^T\boldsymbol{x} \;+\; \boldsymbol{\alpha}^T\left(\boldsymbol{A}\,\boldsymbol{x} \;+\; \boldsymbol{d}\right) \;= \\
&-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \;+\; \left(\boldsymbol{x}^T\boldsymbol{K} \;+\; \boldsymbol{c}^T \;+\; \boldsymbol{\alpha}^T\boldsymbol{A}\right)\,\boldsymbol{x} \;+\; \boldsymbol{\alpha}^T\boldsymbol{d} \;= \\
&-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \;+\; \boldsymbol{\alpha}^T\boldsymbol{d} \;= \\
&-\frac{1}{2}\left(-\boldsymbol{K}^{-1}\left(\boldsymbol{c} \;+\; \boldsymbol{A}^T\boldsymbol{\alpha}\right)\right)^T\boldsymbol{K}\,\left(-\boldsymbol{K}^{-1}\left(\boldsymbol{c} \;+\; \boldsymbol{A}^T\boldsymbol{\alpha}\right)\right) \;+\; \boldsymbol{\alpha}^T\boldsymbol{d} \;= \\
&-\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{A}\,\boldsymbol{K}^{-1}\boldsymbol{A}^T\boldsymbol{\alpha} \;+\; \left(\boldsymbol{d}^T \;-\; \boldsymbol{c}^T\boldsymbol{K}^{-1}\boldsymbol{A}^T\right)\,\boldsymbol{\alpha} \;-\; \frac{1}{2}\boldsymbol{c}^T\boldsymbol{K}^{-1}\boldsymbol{c}\,.
\end{aligned}
\tag{4.214}
$$

Note that $\frac{1}{2}\boldsymbol{c}^T\boldsymbol{K}^{-1}\boldsymbol{c}$ is constant in $\boldsymbol{\alpha}$ and $\boldsymbol{x}$. We transform the maximization again into minimization by using the negative objective.

The dual is

$$
\begin{array}{ll}
\min\limits_{\boldsymbol{x}} & \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{A}\boldsymbol{K}^{-1}\boldsymbol{A}^T\boldsymbol{\alpha} \;-\; \left(\boldsymbol{d}^T \;-\; \boldsymbol{c}^T\boldsymbol{K}^{-1}\boldsymbol{A}^T\right)\,\boldsymbol{\alpha} \\
\text{s.t.} & \boldsymbol{0} \;\leq\; \boldsymbol{\alpha}\,.
\end{array}
\tag{4.215}
$$

Note that the dual of the dual is in general not the primal but a similar problem. Dualizing twice, however, gives again the dual.

Optimization of Convex Problems.

The convex optimization can be solved by gradient descent or constraint gradient descent methods. See next chapter for such methods.

Efficient methods are *interior point methods*. An interior point is a pair $(\boldsymbol{x}, \boldsymbol{\alpha})$ which satisfies both the primal and dual constraints.

We can rewritten the optimality conditions of eq. (4.210) as

$$
\boldsymbol{K}\,\boldsymbol{x} \;+\; \boldsymbol{A}^T\boldsymbol{\alpha} \;+\; \boldsymbol{c} \;=\; 0 \tag{4.216}
$$

$$
\frac{\partial L}{\partial \boldsymbol{\alpha}} \;=\; \boldsymbol{A}\,\boldsymbol{x} \;+\; \boldsymbol{d} \;+\; \boldsymbol{\xi} \;=\; \boldsymbol{0} \tag{4.217}
$$

$$
\boldsymbol{\alpha}^T\boldsymbol{\xi} \;=\; \boldsymbol{0} \tag{4.218}
$$

$$
\boldsymbol{\alpha}, \boldsymbol{\xi} \;\geq\; \boldsymbol{0}\,, \tag{4.219}
$$

where we set $\boldsymbol{0} \;\leq\; \boldsymbol{\xi} \;=\; -\;(\boldsymbol{A}\,\boldsymbol{x} \;+\; \boldsymbol{d})$.

The first two equations are linear in the variables $\boldsymbol{\alpha}$ and $\boldsymbol{\xi}$, however the third equations is quadratic.

Interior point algorithms solve these equations in by an iterative method called "predictor-corrector" and set $\alpha_i\,\xi_i \;=\; \eta \;>\; 0$ which is decreased (annealed) to zero.

4.15.2 Sequential Minimal Optimization

Because support vector machines are applied to large problems efficient solvers are needed.

The first idea is to do *"chunking"*, i.e. optimize only on a subset of the data points an extract the support vectors. If this is done multiple times then the previous support vectors can be used to optimize the final problem. Idea is, if we selected all final support vectors then the solution is optimal.

Another idea is to select a *"working set"*. The working set are the variables which are optimized while the remaining variables are freezed (kept constant).

The working set idea can be driven to its extreme by only selecting two variables for optimization. This method is called *"Sequential Minimal Optimization"* (SMO) and introduced by Platt. Advantage of SMO algorithms is that the problems for two variables can be solved analytically.

Almost all implementations of SVMs are based on an SMO algorithm because of its efficiency.

We start with the C-SVM optimization problem given in eq. (4.42):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \, y^i \, \alpha_j \, y^j \, \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \; - \; \sum_{i=1}^{l} \alpha_i \\ \text{s.t.} \quad & 0 \; \leq \; \alpha_i \; \leq \; C \\ & \sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \; . \end{aligned} \tag{4.220}$$

If we fix all α_i except for two α-values. Without loss of generality we denote these two alphas by α_1 and α_2.

We obtain for the equality constraint

$$y^1 \, \alpha_1 \; + \; y^2 \, \alpha_2 \; + \; \sum_{i=3}^{l} \alpha_i \, y^i \; = \; 0 \tag{4.221}$$

which is

$$y^1 \, y^2 \, \alpha_1 \; + \; \alpha_2 \; = \; - \; \sum_{i=3}^{l} \alpha_i \, y^i \, y^2 \tag{4.222}$$

If we set $s \; = \; y^1 \, y^2$ and $\gamma \; = \; - \; \sum_{i=3}^{l} \alpha_i \, y^i \, y^2$ then we have

$$s \, \alpha_1 \; + \; \alpha_2 \; = \; \gamma \; . \tag{4.223}$$

This equation must still hold after α_1 and α_2 are optimized (changed) in order to fulfill the equality constraint. If we set $K_{ij} = \left(\boldsymbol{x}^i\right)^T \; \boldsymbol{x}^j$ or $K_{ij} = k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ then we obtain for the optimization of

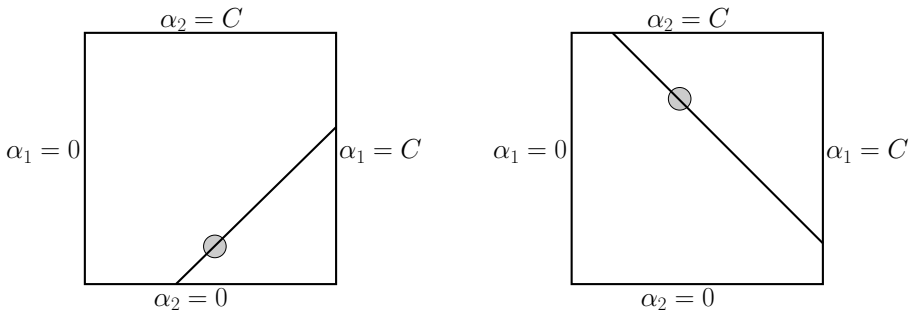


Figure 4.46: The two Lagrange multipliers α_1 and α_2 must fulfill the constraint $s\,\alpha_1 \; + \; \alpha_2 \; = \; \gamma$. The two possibilities for $s = -1$ (left) and $s = 1$ (right) are depicted.

two variables

$$\begin{array}{ll} \min\limits_{\alpha_1,\alpha_2} & \frac{1}{2}\,\left(\alpha_1^2\,K_{11} \; + \; \alpha_2^2\,K_{22} \; + \; 2\,s\,\alpha_1\,\alpha_2\,K_{12}\right) \; + \\ & c_1\,\alpha_1 \; + \; c_2\,\alpha_2 \\ \text{s.t.} & 0 \; \leq \; \alpha_1, \alpha_2 \; \leq \; C \\ & s\,\alpha_1 \; + \; \alpha_2 \; = \; \gamma\,, \end{array} \tag{4.224}$$

where $c_1 \; = \; -\,1 \; + \; y^1 \; \sum_{i=3}^{l} K_{1i}\,y^i\,\alpha_i$ and $c_2 \; = \; -\,1 \; + \; y^2 \; \sum_{i=3}^{l} K_{2i}\,y^i\,\alpha_i$. The constraint $s\,\alpha_1 \; + \; \alpha_2 \; = \; \gamma$ is depicted in Fig. 4.46.

Let us consider for the moment the optimization problem without the box constraints $0 \; \leq \; \alpha_1, \alpha_2 \; \leq \; C$.

We insert $\alpha_2 \; = \; \gamma \; - \; s\,\alpha_1$ into the objective and obtain

$$\begin{array}{l} \frac{1}{2}\,\left(\alpha_1^2\,K_{11} \; + \; (\gamma \; - \; s\,\alpha_1)^2\,K_{22} \; + \; 2\,s\,\alpha_1\;(\gamma \; - \; s\,\alpha_1)\,K_{12}\right) \\ + \, c_1\,\alpha_1 \; + \; c_2\;(\gamma \; - \; s\,\alpha_1) \; = \\ \frac{1}{2}\;\left((K_{11} \; + \; K_{22} \; - \; 2\,K_{12})\,\alpha_1^2 \; + \right. \\ \left.(-\,2\,\gamma\,s\,K_{22} \; + \; 2\,\gamma\,s\,K_{12} \; + \; 2\,c_1 \; - \; 2\,s\,c_2)\,\alpha_1\right) \; + \\ \frac{1}{2}\,\gamma^2\,K_{22} \; + \; c_2\,\gamma \end{array} \tag{4.225}$$

For the optimum, the derivative with respect to α_1 must be zero:

$$\begin{array}{l} (K_{11} \; + \; K_{22} \; - \; 2\,K_{12})\,\alpha_1 \; + \\ (-\,\gamma\,s\,K_{22} \; + \; \gamma\,s\,K_{12} \; + \; c_1 \; - \; s\,c_2) \; = \; 0 \end{array} \tag{4.226}$$

The optimal α_1 is

$$\alpha_1 = \frac{\gamma\,s\,K_{22} \; - \; \gamma\,s\,K_{12} \; - \; c_1 \; + \; s\,c_2}{K_{11} \; + \; K_{22} \; - \; 2\,K_{12}}\,. \tag{4.227}$$

We use the discriminant function values

$$f(\boldsymbol{x}^1) \;=\; \sum_{i=1}^{l} y^i \,\alpha_i \,K_{i1} \;+\; b \tag{4.228}$$

$$f(\boldsymbol{x}^2) \;=\; \sum_{i=1}^{l} y^i \,\alpha_i \,K_{i2} \;+\; b\,. \tag{4.229}$$

We now can rewrite c_1 and c_2 as

$$c_1 \;=\; y^1\, f(\boldsymbol{x}^1) \;-\; y^1\, b \;-\; 1 \;-\; K_{11}\,\alpha_1 \;-\; s\,K_{12}\,\alpha_2 \tag{4.230}$$

$$c_2 \;=\; y^2\, f(\boldsymbol{x}^2) \;-\; y^2\, b \;-\; 1 \;-\; K_{22}\,\alpha_2 \;-\; s\,K_{12}\,\alpha_1 \tag{4.231}$$

$$s\,c_2 \;=\; y^1\, f(\boldsymbol{x}^2) \;-\; y^1\, b \;-\; s \;-\; s\,K_{22}\,\alpha_2 \;-\; K_{12}\,\alpha_1 \tag{4.232}$$

$$\tag{4.233}$$

Because of

$$\gamma \;=\; \alpha_2 \;+\; s\,\alpha_1 \tag{4.234}$$

we obtain

$$\begin{aligned} &\gamma\, s\, K_{22} \;-\; \gamma\, s\, K_{12} \;= \\ &s\, K_{22}\,\alpha_2 \;+\; K_{22}\,\alpha_1 \;-\; s\, K_{12}\,\alpha_2 \;-\; K_{12}\,\alpha_1\,. \end{aligned} \tag{4.235}$$

Now we can rewrite the enumerator of eq. (4.227):

$$\begin{aligned} &\gamma\, s\, K_{22} \;-\; \gamma\, s\, K_{12} \;-\; c_1 \;+\; s\, c_2 \;= \\ &s\, K_{22}\alpha_2 \;+\; K_{22}\,\alpha_1 \;-\; s\, K_{12}\,\alpha_2 \;-\; K_{12}\,\alpha_1 \;- \\ &\;y^1\, f(\boldsymbol{x}^1) \;+\; y^1\, b \;+\; 1 \;+\; K_{11}\,\alpha_1 \;+\; s\, K_{12}\,\alpha_2 \;+ \\ &\;y^1\, f(\boldsymbol{x}^2) \;-\; y^1\, b \;-\; s \;-\; s\, K_{22}\,\alpha_2 \;-\; K_{12}\,\alpha_1 \;= \\ &y^1\left(f(\boldsymbol{x}^2) \;-\; f(\boldsymbol{x}^1)\right) \;+ \\ &\alpha_1\,(K_{22} \;+\; K_{11} \;-\; 2\,K_{12}) \;+\; 1 \;-\; s \;= \\ &y^1\left(\left(f(\boldsymbol{x}^2) \;-\; y^2\right) \;-\; \left(f(\boldsymbol{x}^1) \;-\; y^1\right)\right) \;+ \\ &\alpha_1\,(K_{22} \;+\; K_{11} \;-\; 2\,K_{12})\;, \end{aligned} \tag{4.236}$$

where the last equality stems from the fact that $(1 \;-\; s) \;=\; y^1\,(y^1 \;-\; y^2)$.

As update for α_1 we obtain

$$\alpha_1^{\text{new}} = \alpha_1 \;+\; \frac{y^1\left(\left(f(\boldsymbol{x}^2) \;-\; y^2\right) \;-\; \left(f(\boldsymbol{x}^1) \;-\; y^1\right)\right)}{K_{11} \;+\; K_{22} \;-\; 2\,K_{12}}\,. \tag{4.237}$$

Because

$$s\,\alpha_1^{\text{new}} \;+\; \alpha_2^{\text{new}} \;=\; \gamma \;=\; s\,\alpha_1 \;+\; \alpha_2 \tag{4.238}$$

we obtain for the update of α_2

$$\alpha_2^{\text{new}} = \alpha_2 \; + \; s \; (\alpha_1 \; - \; \alpha_1^{\text{new}}) \; . \tag{4.239}$$

However we did not consider the box constraints until now.

We define as new bounds

$$L \; = \; \begin{cases} \max\{0, \alpha_1 \; - \; \alpha_2\} & \text{if } \; s \; = \; -1 \\ \max\{0, \alpha_1 \; + \; \alpha_2 \; - \; C\} & \text{otherwise} \end{cases} \tag{4.240}$$

$$H \; = \; \begin{cases} \min\{C, C \; + \; \alpha_1 \; - \; \alpha_2\} & \text{if } \; s \; = \; -1 \\ \min\{C, \alpha_1 \; + \; \alpha_2\} & \text{otherwise} \end{cases} \; . \tag{4.241}$$

We obtain finally following update rules:

if

$$K_{11} \; + \; K_{22} \; - \; 2 \, K_{12} \; = \; 0 \tag{4.242}$$

then

$$\alpha_1^{\text{new}} \; = \; \begin{cases} H & \text{if } \; y^1 \left(\left(f(\boldsymbol{x}^2) \; - \; y^2 \right) \; - \; \left(f(\boldsymbol{x}^1) \; - \; y^1 \right) \right) \; > \; 0 \\ L & \text{otherwise} \end{cases} \; . \tag{4.243}$$

else

$$K_{11} \; + \; K_{22} \; - \; 2 \, K_{12} \; > \; 0 \tag{4.244}$$

then first compute

$$\alpha_1^{\text{temp}} = \alpha_1 \; + \; \frac{y^1 \left(\left(f(\boldsymbol{x}^2) \; - \; y^2 \right) \; - \; \left(f(\boldsymbol{x}^1) \; - \; y^1 \right) \right)}{K_{11} \; + \; K_{22} \; - \; 2 \, K_{12}} \; . \tag{4.245}$$

and then

$$\alpha_1^{\text{new}} \; = \; \begin{cases} H & \text{if } \; H \; \leq \; \alpha_1^{\text{temp}} \\ \alpha_1^{\text{temp}} & \text{if } L \; < \; \alpha_1^{\text{temp}} \; < \; H \\ L & \text{if } \alpha_1^{\text{temp}} \; \leq \; L \end{cases} \; . \tag{4.246}$$

In both cases (“if” and “else” case) the update for α_2 is

$$\alpha_2^{\text{new}} = \alpha_2 \; + \; s \; (\alpha_1 \; - \; \alpha_1^{\text{new}}) \; . \tag{4.247}$$

Selection Rules.

The question arises how to chose α_1 and α_2.

We have to chose two αs.

An outer loop chooses α_1. This loop iterates over all patterns i which violate the KKT conditions. α_i's on the boundary are skipped in a first round. Afterwards also α_i on the boundary are treated.

Sometimes a whole sweep through the KKT violating data should be made to keep everything consistent.

The second α_2 is chosen to make a large step towards the minimum. Here large steps in α are considered even if the objective is decreased by a small amount.

Pratt uses a heuristics where he only evaluates the numerator of the update rule. He avoids to evaluate the kernel for checking new alphas. Therefore large differences in the relative errors $f(\boldsymbol{x}^1) - y^1$ and $f(\boldsymbol{x}^2) - y^2$ are looked for.

If this heuristics for choosing α_2 fails then (1) at all non-bound examples are looked then (2) all examples are looked at to make progress and finally (3) if these choices fail then a new α_1 is chosen.

Other heuristics mark updated αs in order to avoid cycles where three or more αs are updated and small progress is made. Marked αs can be updated simultaneously in a special step to avoid the cycles.

Initially $\boldsymbol{\alpha} = \mathbf{0}$ is a good starting point and to prefer if few support vectors will be selected.

Another trick is to introduce ϵ from the regression setting or from the P-SVM in order to select only few support vectors. These support vectors can be used in another round with reduced ϵ which can be viewed as annealing of support vectors.

Note that after each α update the $f(\boldsymbol{x}^i)$ can be updated and sums over all $\alpha > 0$ can be avoided.

Similar SMO algorithms exist for all other SVM approaches, where the regression setting with α_i^+ and α_i^- can be made efficiently because only one of these α can be different from zero.

As stopping criterion the KKT conditions may be used.

4.16 Designing Kernels for Bioinformatics Applications

At the beginning of the chapter in Section 4.1, a couple of special kernels for bioinformatics have been mentioned.

Here we will explain some of these kernels which are relevant for bioinformatics.

Remember that every positive definite kernel is a dot product in some space as follows from Mercer's theorem.

In general performance is optimized if the kernel is designed, i.e. an expert defines which and how features are extracted from the original data. The mapping of the original data to the features is was called "feature map" and the feature map combined with dot product is a kernel.

4.16.1 String Kernel

The string kernel describes a string by all its subsequences. Subsequences do not need to be contiguous that is gaps are allowed.

The string kernel counts k how often the subsequence occurs and the distance t between the first and the last symbol. The contribution of a subsequence is

$$k\,\lambda^t\,, \tag{4.248}$$

where $0 < \lambda \leq 1$ is a decay factor. Therefore, the contribution decays exponentially with the number of gaps.

For a subsequence all contributions of its occurrences are summed up to give the final score for the subsequence.

The substring "BIO" in "BIOINFORMATICS" obtains $1 \cdot \lambda^3 + 2 \cdot \lambda^7$ as score, because the following subsequences exist:

BIOINFORMATICS
BIOINF**O**RMATICS
BIO**I**NF**O**RMATICS.

The string kernel as an upper limit n of length of the substrings it will consider. To be independent of the length of the string the feature vector is normalized to length one. A standard dot product or an RBF kernel can be used to perform the dot product. The string kernel is the feature vector generation together with the dot product.

4.16.2 Spectrum Kernel

The spectrum kernel is similar to the string kernel it counts occurrences of subsequences of length n in the string. However gaps are not allowed.

The kernel only checks whether the corresponding substring is contained in the string.

4.16.3 Mismatch Kernel

The mismatch kernel is similar to the spectrum kernel it counts occurrences of gapless subsequences of length n with no more than m mismatches. The kernel only checks whether the corresponding substring is contained in the string with less or equal to m mismatches.

4.16.4 Motif Kernel

A library of motifs from PROSITE or BLOCKS or PRINTS databases is build. The occurrence of a motif is the feature vector.

4.16.5 Pairwise Kernel

The feature vector is the vector of alignment scores to the training set. That means a new vector is aligned to all training sequences and the resulting scores give the feature vector.

4.16.6 Local Alignment Kernel

The Smith-Waterman algorithm finds the highest scoring local alignment $SW(\boldsymbol{x}, \boldsymbol{y}, S, g)$ (see Bioinformatics 1), where $\boldsymbol{x}$ and $\boldsymbol{y}$ are strings, S is a scoring Matrix (e.g. PAM or BLOSUM), and g is a gap penalty (or gap opening and extending penalty).

$$SW(\boldsymbol{x}, \boldsymbol{y}, S, g) \ = \ \max_{i_s, i_e, j_s, j_e} s(\boldsymbol{x}, \boldsymbol{y}, S, g, i_s, i_e, j_s, j_e) \ , \tag{4.249}$$

where $s(\boldsymbol{x}, \boldsymbol{y}, S, g, i_s, i_e, j_s, j_e)$ is the alignment score (Needleman-Wunsch) if the subsequence $(x^{i_s}, \ldots, x^{i_e})$ is aligned to $(y^{j_s}, \ldots, y^{j_e})$.

Because the Smith-Waterman score is not positive definite another kernel has to be defined.

The "local alignment kernel"

$$k_{LA}(\boldsymbol{x}, \boldsymbol{y}, S, g, \beta) \ = \ \sum_{i_s=1, i_e=i_s+1, j_s=1, j_e=j_s+1} \exp\left(\beta \ s(\boldsymbol{x}, \boldsymbol{y}, S, g, i_s, i_e, j_s, j_e)\right) \tag{4.250}$$

is a positive definite kernel.

4.16.7 Smith-Waterman Kernel

In some application the Smith-Waterman score is used as kernel. However the theoretical basis is missing because it is not positive definite. It may not be a dot product in some space.

But the P-SVM can naturally use the Smith-Waterman score as kernel because positive definiteness is not longer necessary.

4.16.8 Fisher Kernel

A class of kernels can be defined on generative models. They are very attractive because generative models can deal with different sequence length, with missing or incomplete data, or with uncertainty. An advantage of generative models is that they supply a likelihood $p(\{\boldsymbol{x}\}; \boldsymbol{w})$ for new data $\{\boldsymbol{x}\}$ which is a measure which can be interpreted statistically.

A class of kernels is defined as

$$k_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{y}) \ = \ \frac{\partial \ln p(\boldsymbol{x}; \boldsymbol{w})}{\partial \boldsymbol{w}} \ \boldsymbol{M}^{-1} \ \frac{\partial \ln p(\boldsymbol{y}; \boldsymbol{w})}{\partial \boldsymbol{w}} \ , \tag{4.251}$$

where $M \ = \ \boldsymbol{I}_F(\boldsymbol{w})$ is called the "Fisher kernel" and $M \ = \ \boldsymbol{I}$ is called the "plain kernel" which is often used to approximate the Fisher kernel and then also called "Fisher kernel".

The Fisher kernel was introduced for protein homology detection Jaakkola et al. [1999, 2000] where a hidden Markov model (a generative model) was trained on the positive examples. Thereafter a SVM with Fisher kernel was used as discriminative method which can also take into account the negative examples.

4.16.9 Profile and PSSM Kernels

The kernel working on the sequence can be generalized to kernels working on profiles or position specific scoring matrices (PSSMs). PSSMs are obtained by letting PSI-BLAST run against a data base like NR.

Instead of the original sequence an average over very similar sequences is processed. The similar sequences are found by PSI-BLAST in a large data base and then averaged by a multiple alignment. Per column of the alignment a frequency vector or a scoring vector (the PSSM) can be computed.

Instead of alignment methods profile alignment methods must be used.

4.16.10 Kernels Based on Chemical Properties

Kernels have been suggest which take into account the chemical properties of amino acid sequences like hydrophobicity, polarity, atomic weight, bulkiness, etc.

However these kernels where in general inferior to the kernels based on alignment or profiles.

4.16.11 Local DNA Kernel

Here DNA sequences are locally matched in a window. The window is shifted over the sequences.

This kernel is used to detect start codons where translation starts.

The kernel can be improved by shifting 3 nucleotides which accounts for frame shifts.

4.16.12 Salzberg DNA Kernel

Similar as with PSSMs also for DNA log-odd scores can be computed according to the Salzberg method.

4.16.13 Shifted Weighted Degree Kernel

String kernel where positions are important. “Weighted Degree Kernel” uses fixed position measured with respect to a reference point. “Shifted Weighted Degree Kernel” shifts the sequences against each other in order to make the positions more fuzzy.

These kernels are used to detect alternative splice sites in DNA sequences.

4.17 Kernel Principal Component Analysis

In Section 4.7 the kernel trick was mentioned. The kernel trick can be applied to every algorithm which is based on dot products. The kernel trick replaces the dot product by a kernel which is equivalent to mapping the vectors into a feature space and then building the dot product in this feature space.

The kernel trick is not always obvious because some algorithms can be formulated solely by dot products but their standard description is different.

Pre-images. Difficulties appear if the result is a vector like with clustering the cluster centers. In this case the result has to be projected back into the original space but the data points only form a sub-manifold in the feature space on which the resulting vector in general is not located on. This is the problem of pre-images for which different heuristics exist. However an exact solution to this problem is impossible because a distance measure in the feature space between the result and the manifold of data points must be defined.

We want to demonstrate the kernel trick for Principal Component Analysis (PCA) which is called *Kernel Principal Component Analysis* or *kernel PCA*.

PCA review. PCA computes the covariance matrix $\boldsymbol{C}$ of the data, then determines $\boldsymbol{C}$'s eigenvectors $\boldsymbol{w}$ which are sorted by their eigenvalues λ. The projection $\boldsymbol{x}^T \boldsymbol{w}_i$ of a data vector $\boldsymbol{x}$ onto the i-th eigenvector $\boldsymbol{w}_i$ is the i-th principal component of the data vector $\boldsymbol{x}$. PCA is often used for dimensionality reduction or for visualization of the data. If the first k principal components are used to represent the data then $\sum_{t=k+1}^{d} \lambda_t$ is the expected mean squared reconstruction error between the back-projected representation and the original data. We assume that the data was normalized so that $\sum_{t=1}^{d} \lambda_t \ = \ 1$.

Kernel PCA. We assume that the data is projected into the feature space by

$$x \ \mapsto \ \boldsymbol{\Phi}(\boldsymbol{x}) \ . \tag{4.252}$$

Let us for the moment assume that the data is centered in the feature space, that is

$$\sum_{i=1}^{l} \boldsymbol{\Phi}(\boldsymbol{x}^i) \ = \ \boldsymbol{0} \ . \tag{4.253}$$

The covariance matrix in feature space is given by

$$\boldsymbol{C} \ = \ \frac{1}{l} \ \sum_{i=1}^{l} \boldsymbol{\Phi}(\boldsymbol{x}^i) \ \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \ . \tag{4.254}$$

Note, that the Gram matrix is $\boldsymbol{K} \ = \ \sum_{i=1}^{l} \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^i)$.

We are searching for the eigenvectors of $\boldsymbol{C}$, i.e. for vectors fulfilling

$$\boldsymbol{C} \ \boldsymbol{w} \ = \ \lambda \ \boldsymbol{w} \ . \tag{4.255}$$

We are interested in solutions $\boldsymbol{w}$ which lie in the span of $\{\boldsymbol{\Phi}(\boldsymbol{x}^1), \ldots, \boldsymbol{\Phi}(\boldsymbol{x}^l)\}$, therefore we are searching for vectors $\boldsymbol{w}$ which fulfill

$$\begin{aligned} &\forall_{1 \leq n \leq l} : \ (\lambda \ \boldsymbol{w})^T \boldsymbol{\Phi}(\boldsymbol{x}^n) \ = \ \lambda \ \boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x}^n) \ = \\ &(\boldsymbol{C} \ \boldsymbol{w})^T \boldsymbol{\Phi}(\boldsymbol{x}^n) \ = \ \boldsymbol{w}^T \boldsymbol{C} \ \boldsymbol{\Phi}(\boldsymbol{x}^n) \ , \end{aligned} \tag{4.256}$$

because they are unique in the span of the mapped data vectors.

As with SVMs the eigenvectors can be expanded with respect to the feature vectors:

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l} \alpha_i \; \boldsymbol{\Phi}(\boldsymbol{x}^i) \; . \tag{4.257}$$

Inserting this equation together with the definition of $\boldsymbol{C}$ in eq. (4.254) into eq. (4.256) gives

$$\begin{aligned} &\lambda \sum_{i=1}^{l} \alpha_i \; \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^n) \;= \\ &\frac{1}{l} \left(\sum_{i=1}^{l} \alpha_i \sum_{j=1}^{l} \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \left(\boldsymbol{\Phi}(\boldsymbol{x}^j) \; \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \right) \right) \boldsymbol{\Phi}(\boldsymbol{x}^n) \; . \end{aligned} \tag{4.258}$$

If we use the Gram matrix $\boldsymbol{K}$ with $K_{ij} \;=\; \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i)$. This equation holds for $1 \;\leq\; n \;\leq\; l$ and the vectors $\boldsymbol{\Phi}(\boldsymbol{x}^n)$ span the whole space, therefore we obtain

$$l \; \lambda \; \boldsymbol{K} \; \boldsymbol{\alpha} \;=\; \boldsymbol{K}^2 \; \boldsymbol{\alpha} \; . \tag{4.259}$$

To solve this equation we solve the eigenvalue problem

$$l \; \lambda \; \boldsymbol{\alpha} \;=\; \boldsymbol{K} \; \boldsymbol{\alpha} \; . \tag{4.260}$$

The $\boldsymbol{\alpha}$ describes the eigenvector $\boldsymbol{w}$ which must have the length 1, resulting in

$$\begin{aligned} &1 \;=\; \boldsymbol{w}^T \boldsymbol{w} \;=\; \sum_{ij=(1,1)}^{(l,l)} \alpha_i \; \alpha_j \; \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i) \;= \\ &\sum_{ij=(1,1)}^{(l,l)} \alpha_i \; \alpha_j \; K_{ij} \;=\; \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \;=\; l \; \lambda \; \boldsymbol{\alpha}^T \boldsymbol{\alpha} \end{aligned} \tag{4.261}$$

The vector $\boldsymbol{\alpha}$ has to be normalized to fulfill

$$l \; \lambda \; \|\boldsymbol{\alpha}\|^2 \;=\; 1 \tag{4.262}$$

$$\|\boldsymbol{\alpha}\| \;=\; \frac{1}{\sqrt{l \; \lambda}} \tag{4.263}$$

by

$$\alpha_i^{\text{new}} \;=\; \frac{\alpha_i}{\|\boldsymbol{\alpha}\| \; \sqrt{l \; \lambda}} \tag{4.264}$$

The projection onto $\boldsymbol{w}$ can be computed as

$$\boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i \; \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i \; K_{ij} \; . \tag{4.265}$$

Here it can be seen that the explicit representation $\boldsymbol{\Phi}(\boldsymbol{x})$ is not necessary and we can set $K_{ij} = k(\boldsymbol{x}^i, \boldsymbol{x}^j)$, i.e. to apply the kernel trick.

However the kernel trick makes the assumption of centered data more complicated because we want to avoid explicit representations in feature space.

Note that

$$\begin{aligned}
&\left(\boldsymbol{\Phi}(\boldsymbol{x}^i) - \frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}(\boldsymbol{x}^t)\right)^T \left(\boldsymbol{\Phi}(\boldsymbol{x}^j) - \frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}(\boldsymbol{x}^t)\right) = \\
&\boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}^j) - \frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}^T(\boldsymbol{x}^t)\boldsymbol{\Phi}(\boldsymbol{x}^j) - \frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}^t) + \\
&\frac{1}{l^2}\sum_{(s,t)=(1,1)}^{(l,l)}\boldsymbol{\Phi}^T(\boldsymbol{x}^s)\boldsymbol{\Phi}(\boldsymbol{x}^t)
\end{aligned} \tag{4.266}$$

and that

$$\begin{aligned}
\frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}^T(\boldsymbol{x}^t)\boldsymbol{\Phi}(\boldsymbol{x}^i) &= \left[\frac{1}{l}\boldsymbol{K}\,\mathbf{1}\right]_i \\
\frac{1}{l}\sum_{t=1}^{l}\boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}^t) &= \left[\frac{1}{l}\mathbf{1}^T\boldsymbol{K}\right]_i \\
\frac{1}{l^2}\sum_{(s,t)=(1,1)}^{(l,l)}\boldsymbol{\Phi}^T(\boldsymbol{x}^s)\boldsymbol{\Phi}(\boldsymbol{x}^t) &= \frac{1}{l^2}\,\mathbf{1}^T\boldsymbol{K}\,\mathbf{1}\,.
\end{aligned} \tag{4.267}$$

Therefore the following equations produces a centered kernel matrix:

$$\boldsymbol{K} - \frac{1}{l}\,\boldsymbol{K}\,\mathbf{1}\,\mathbf{1}^T - \frac{1}{l}\,\mathbf{1}\,\mathbf{1}^T\boldsymbol{K} + \frac{1}{l^2}\,\left(\mathbf{1}^T\boldsymbol{K}\,\mathbf{1}\right)\mathbf{1}\,\mathbf{1}^T\,. \tag{4.268}$$

A new data point $\boldsymbol{x}$ can be centered by first computing

$$\boldsymbol{k}(\boldsymbol{x},.) = \Big(k(\boldsymbol{x},\boldsymbol{x}^1),\ldots,k(\boldsymbol{x},\boldsymbol{x}^l)\Big)^T \tag{4.269}$$

and then

$$\boldsymbol{k}(\boldsymbol{x},.) - \frac{1}{l}\,\boldsymbol{K}\,\mathbf{1} - \frac{1}{l}\,\mathbf{1}^T\boldsymbol{k}(\boldsymbol{x},.)\,\mathbf{1} + \frac{1}{l^2}\,\left(\mathbf{1}^T\boldsymbol{K}\,\mathbf{1}\right)\mathbf{1}\,. \tag{4.270}$$

Kernel PCA works as follows:

- center the Gram matrix according to eq. (4.268)
- compute eigenvectors and eigenvalues of the Gram matrix $\boldsymbol{K}$

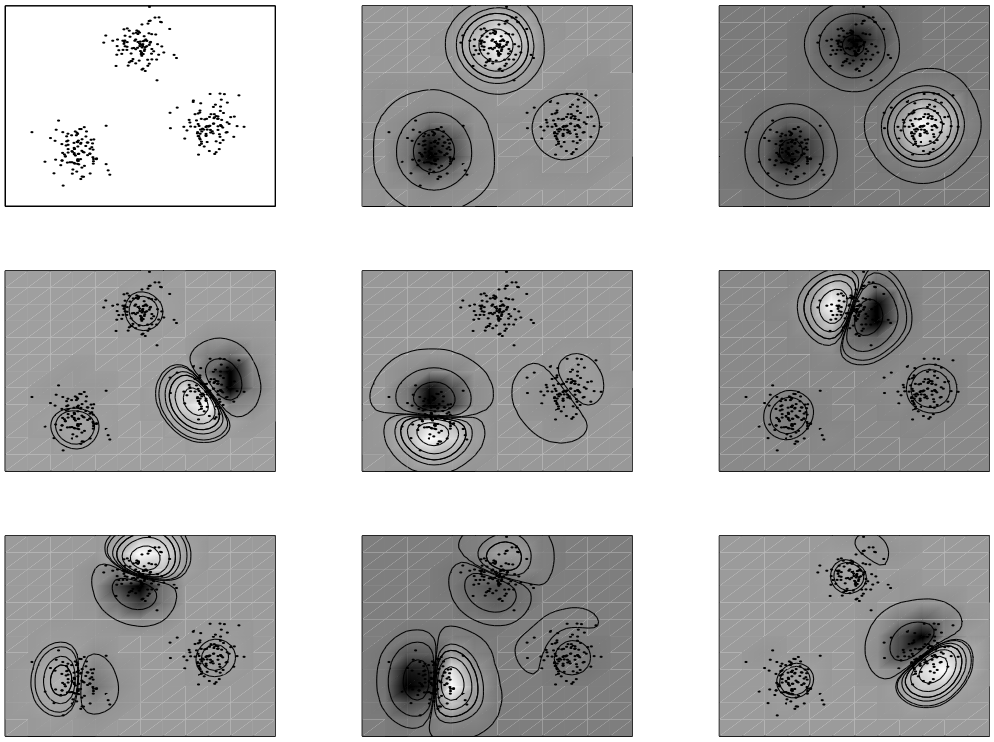


Figure 4.47: Kernel PCA example. Top left: original data. From top middle to upper right the 8 first principal components are depicted. The first two separate the clusters, the next 3 (middle panel) split the clusters, and the next 3 (bottom panel) split them orthogonal to the previous 3 components. An RBF-kernel with $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = \exp\left(-10 \left\|\boldsymbol{x}^i - \boldsymbol{x}^j\right\|^2\right)$ was used. From [Schölkopf et al., 1998]

- normalize eigenvectors $\boldsymbol{\alpha}$ according to eq. (4.264)
- project a new vector $\boldsymbol{x}$ onto eigenvectors by first centering it using eq. (4.270) and then according to eq. (4.265).

Kernel PCA toy examples are shown in Figures 4.47 and 4.48.

4.18 Kernel Discriminant Analysis

Here we will apply the kernel trick to another well known method: "Linear Discriminant Analysis" (LDA) or "Fisher Discriminant Analysis" (FDA).

Let us assume we have a binary problem as with the SVM approach. The Fisher discriminant algorithm maximizes the *Rayleigh coefficient* J with respect to $\boldsymbol{w}$:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{M}\, \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w}}\,, \tag{4.271}$$

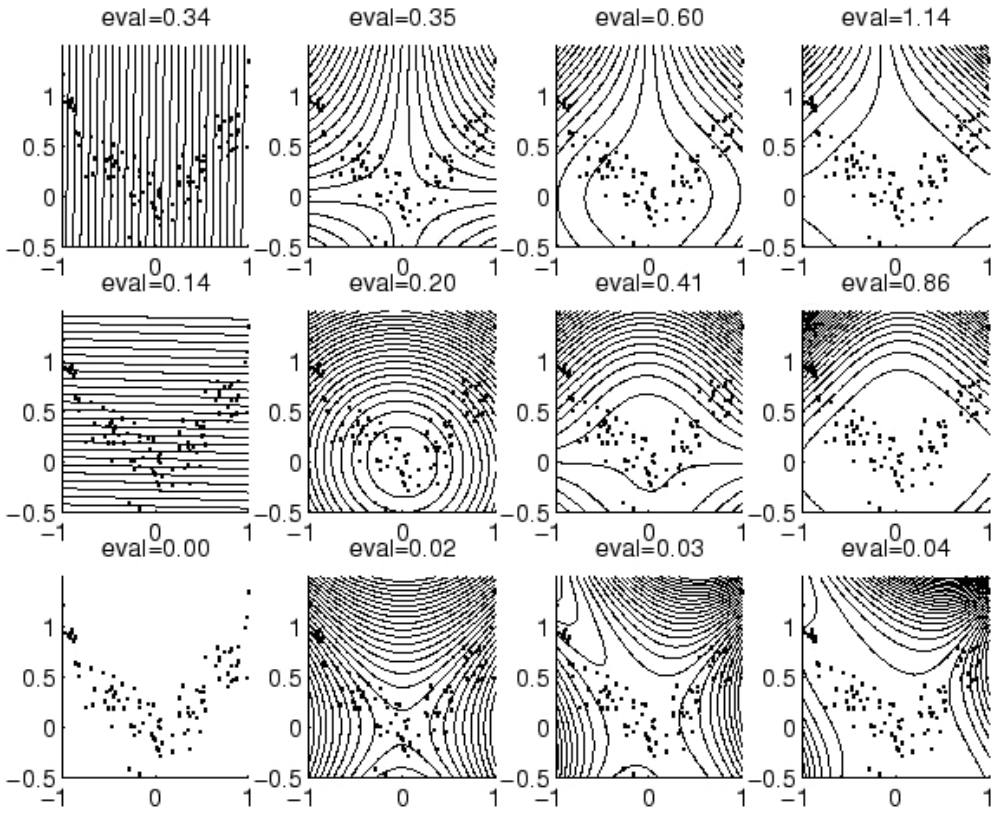


Figure 4.48: Another kernel PCA example. Each column corresponds to one method based on PCA. Each column gives the three largest eigenvalues with the according eigenvectors depicted as contour lines. In the first column linear PCA, and in the second, third and last column polynomial kernel with degree 2,3, and 4, respectively, is shown. From [Schölkopf and Smola, 2002]

where

$$\boldsymbol{M} \;=\; (\boldsymbol{m}_- \;-\; \boldsymbol{m}_+)\;(\boldsymbol{m}_- \;-\; \boldsymbol{m}_+)^T \tag{4.272}$$

is the between class variance and

$$\boldsymbol{N} \;=\; \sum_{i=1;y^i=1}^{l} \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_+\right)\;\left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_+\right)^T \;+ \tag{4.273}$$

$$\sum_{i=1;y^i=-1}^{l} \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_-\right)\;\left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_-\right)^T$$

is the within class variance.

The Fisher's discriminant attempts at minimizing the variance within one class whilst separating the class centers a good as possible.

Let l_+ be the number of class 1 examples and l_- the number of class -1 examples

Here we used the mean values

$$\boldsymbol{m}_+ \;=\; \frac{1}{l_+} \sum_{i=1;y^i=1}^{l} \boldsymbol{x}^i \tag{4.274}$$

$$\boldsymbol{m}_- \;=\; \frac{1}{l_-} \sum_{i=1;y^i=-1}^{l} \boldsymbol{x}^i \tag{4.275}$$

Note that

$$\boldsymbol{w}^T \boldsymbol{M}\, \boldsymbol{w} \;=\; \left(\boldsymbol{w}^T \boldsymbol{m}_- \;-\; \boldsymbol{w}^T \boldsymbol{m}_+\right)\;\left(\boldsymbol{m}_-^T \boldsymbol{w} \;-\; \boldsymbol{m}_+^T \boldsymbol{w}\right) \tag{4.276}$$

and

$$\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w} \;= \tag{4.277}$$

$$\sum_{i=1;y^i=1}^{l} \left(\boldsymbol{w}^T \boldsymbol{x}^i \;-\; \boldsymbol{w}^T \boldsymbol{m}_+\right)\;\left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{w} \;-\; \boldsymbol{m}_+^T \boldsymbol{w}\right) \;+$$

$$\sum_{i=1;y^i=-1}^{l} \left(\boldsymbol{w}^T \boldsymbol{x}^i \;-\; \boldsymbol{w}^T \boldsymbol{m}_-\right)\;\left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{w} \;-\; \boldsymbol{m}_-^T \boldsymbol{w}\right)$$

Now we proceed like with the kernel PCA and set

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l} \alpha_i\, \boldsymbol{\Phi}(\boldsymbol{x}^i)\;. \tag{4.278}$$

Let $\boldsymbol{K}$ be again the Gram matrix with $K_{ij} \;=\; k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;=\; \boldsymbol{\Phi}^T(\boldsymbol{x}^i)\, \boldsymbol{\Phi}(\boldsymbol{x}^j)$

We use expansion of w according to eq. (4.278) and obtain

$$\begin{aligned} \boldsymbol{w}^T \boldsymbol{m}_- \;&=\; \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x}^j) \;= \\ &\frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \sum_{i=1}^{l} \alpha_i \; \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \; \boldsymbol{\Phi}(\boldsymbol{x}^j) \;= \\ &\frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \sum_{i=1}^{l} \alpha_i \; k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; . \end{aligned} \tag{4.279}$$

We use again

$$\boldsymbol{k}(\boldsymbol{x}, .) \;=\; \Big(k(\boldsymbol{x}, \boldsymbol{x}^1), \ldots, k(\boldsymbol{x}, \boldsymbol{x}^l) \Big)^T \tag{4.280}$$

and define

$$\boldsymbol{\mu}_- \;=\; \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \boldsymbol{k}(\boldsymbol{x}^j, .) \;=\; \frac{1}{l_-} \; \boldsymbol{K} \; \mathbf{1}_- \tag{4.281}$$

to obtain

$$\boldsymbol{w}^T \boldsymbol{m}_- \;=\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_- \; . \tag{4.282}$$

Here we used $\mathbf{1}_-$ the l-dimensional vector which has ones at positions of class -1 examples and 0 otherwise.

Analog using

$$\boldsymbol{\mu}_+ \;=\; \frac{1}{l_+} \sum_{j=1;y^j=1}^{l} \boldsymbol{k}(\boldsymbol{x}^j, .) \;=\; \frac{1}{l_+} \; \boldsymbol{K} \; \mathbf{1}_+ \tag{4.283}$$

we obtain

$$\boldsymbol{w}^T \boldsymbol{m}_+ \;=\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_+ \; . \tag{4.284}$$

We used $\mathbf{1}_+$ the l-dimensional vector which has ones at positions of class 1 examples and 0 otherwise.

$$\begin{aligned} &\boldsymbol{w}^T \boldsymbol{M} \; \boldsymbol{w} \;=\; \left(\boldsymbol{\alpha}^T \boldsymbol{\mu}_- \;-\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_+ \right) \; \left(\boldsymbol{\mu}_-^T \boldsymbol{\alpha} \;-\; \boldsymbol{\mu}_+^T \boldsymbol{\alpha} \right) \;= \\ &\boldsymbol{\alpha}^T \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+ \right) \; \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+ \right)^T \boldsymbol{\alpha} \;=\; \left(\boldsymbol{\alpha}^T \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+ \right) \right)^2 \;= \\ &\boldsymbol{\alpha}^T \boldsymbol{M}_k \; \boldsymbol{\alpha} \; , \end{aligned} \tag{4.285}$$

where

$$\boldsymbol{M}_k \;=\; (\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+)\;(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+)^T\;. \tag{4.286}$$

The expansion of w according to eq. (4.278) gives

$$\begin{aligned}
&\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w} \;=\; \sum_{j=1;y^j=1}^{l} \left(\sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^j) \;-\; \boldsymbol{w}^T \boldsymbol{m}_+ \right) \\
&\left(\sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i) \;-\; \boldsymbol{m}_+^T \boldsymbol{w} \right) \;+ \\
&\sum_{j=1;y^j=-1}^{l} \left(\sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^j) \;-\; \boldsymbol{w}^T \boldsymbol{m}_- \right) \\
&\left(\sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i) \;-\; \boldsymbol{m}_-^T \boldsymbol{w} \right) \;= \\
&\sum_{i,n=(1,1)}^{(l,l)} \alpha_i\; \alpha_n \sum_{j=1;y^j=1}^{l} \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^j) \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i) \;- \\
&l_+ \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\mu}_+\, \boldsymbol{\alpha}^T \boldsymbol{\mu}_+ \;- \\
&l_+ \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\mu}_+\, \boldsymbol{\mu}_+^T \boldsymbol{\alpha} \;+\; l_+\, \boldsymbol{\mu}_+^T \boldsymbol{\alpha}\, \boldsymbol{\mu}_+^T \boldsymbol{\alpha} \;+ \\
&\sum_{i,n=(1,1)}^{(l,l)} \alpha_i\; \alpha_n \sum_{j=1;y^j=-1}^{l} \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}^j) \boldsymbol{\Phi}^T(\boldsymbol{x}^j) \boldsymbol{\Phi}(\boldsymbol{x}^i) \;- \\
&l_- \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\mu}_-\, \boldsymbol{\alpha}^T \boldsymbol{\mu}_- \;- \\
&l_- \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\mu}_-\, \boldsymbol{\mu}_-^T \boldsymbol{\alpha} \;+\; l_-\, \boldsymbol{\mu}_-^T \boldsymbol{\alpha}\, \boldsymbol{\mu}_-^T \boldsymbol{\alpha} \;= \\
&\sum_{i,n=(1,1)}^{(l,l)} \alpha_i\; \alpha_n \sum_{j=1}^{l} K_{ij} \boldsymbol{K}_{ji} \;-\; l_+\, \boldsymbol{\alpha}^T \boldsymbol{\mu}_+\, \boldsymbol{\mu}_+^T \boldsymbol{\alpha} \;- \\
&l_-\, \boldsymbol{\alpha}^T \boldsymbol{\mu}_-\, \boldsymbol{\mu}_-^T \boldsymbol{\alpha} \;= \\
&\boldsymbol{\alpha}^T \left(\boldsymbol{K}\, \boldsymbol{K}^T \;-\; l_+\, \boldsymbol{\mu}_+\, \boldsymbol{\mu}_+^T \;-\; l_-\, \boldsymbol{\mu}_-\, \boldsymbol{\mu}_-^T \right) \boldsymbol{\alpha}\;.
\end{aligned} \tag{4.287}$$

If we define

$$\boldsymbol{N}_k \;=\; \boldsymbol{K}\, \boldsymbol{K}^T \;-\; l_+\, \boldsymbol{\mu}_+\, \boldsymbol{\mu}_+^T \;-\; l_-\, \boldsymbol{\mu}_-\, \boldsymbol{\mu}_-^T \tag{4.288}$$

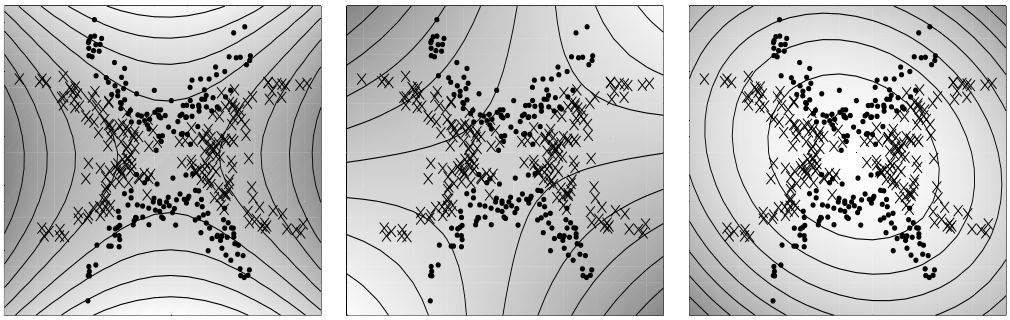


Figure 4.49: Kernel discriminant analysis (KDA) example. Left: KDA contour lines. Middle and Right: first and second component of kernel PCA. With KDA the contour lines are useful to separate the data. From [Mika et al., 1999].

then

$$\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w} \;=\; \boldsymbol{\alpha}^T \boldsymbol{N}_k\, \boldsymbol{\alpha}\;. \tag{4.289}$$

We obtain for the kernel version of Fisher's discriminant the following Rayleigh coefficient to maximize:

$$J(\boldsymbol{\alpha}) \;=\; \frac{\boldsymbol{\alpha}^T \boldsymbol{M}_k\, \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{N}_k\, \boldsymbol{\alpha}}\;. \tag{4.290}$$

The projection of a new vector onto the discriminant direction $\boldsymbol{w}$ is

$$\boldsymbol{w}^T\, \boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i\, \boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i\, k(\boldsymbol{x}^i, \boldsymbol{x})\;. \tag{4.291}$$

The Rayleigh coefficient eq. (4.290) can be maximized by the generalized eigenvalue problem

$$\boldsymbol{M}_k\, \boldsymbol{\alpha} \;=\; \lambda\, \boldsymbol{N}_k\, \boldsymbol{\alpha} \tag{4.292}$$

and selecting the eigenvector with maximal eigenvalue λ.

However until now the regularization, i.e. the capacity control was not considered. It is important for kernel methods because the l dimensional covariance matrix $\boldsymbol{N}_k$ is estimated by l data points.

One approach for regularization is to replace $\boldsymbol{N}_k$ by

$$\boldsymbol{N}_k \;+\; \epsilon \boldsymbol{I}\;. \tag{4.293}$$

Fig. 4.49 shows a toy example for kernel discriminant analysis (KDA) compared with kernel PCA.

However the idea of kernel Fisher discriminant analysis can be written in a more convenient form.

First assume that we affine transform the data in a way that

$$\boldsymbol{w}^T \boldsymbol{m}_+ \; + \; b \; = \; 1 \tag{4.294}$$

and

$$\boldsymbol{w}^T \boldsymbol{m}_- \; + \; b \; = \; -1 \, . \tag{4.295}$$

That means the between class variance is fixed.

Now we have only to minimize the variance of the data points around their class mean (-1 or +1). If we set

$$\boldsymbol{w}^T \boldsymbol{x}^i \; + \; b \; = \; y^i \; + \; \xi_i \tag{4.296}$$

then ξ_i gives the deviation of the projected $\boldsymbol{x}^i$ from its class label.

In order that 1 is the mean projection value of class 1 data points and -1 the mean projection value of class -1 data points, we must ensure that

$$\mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \tag{4.297}$$

$$\mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \, . \tag{4.298}$$

The class 1 variance plus the class 2 variance is given by

$$\boldsymbol{\xi}^T \boldsymbol{\xi} \; = \; \|\boldsymbol{\xi}\|^2 \; . \tag{4.299}$$

This variance must be minimized.

Using as regularization term either $\|\boldsymbol{\alpha}\|^2$ or $\boldsymbol{\alpha}^T \boldsymbol{K} \, \boldsymbol{\alpha}$ we obtain the following quadratic programs:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha},\boldsymbol{\xi},b} & \|\boldsymbol{\xi}\|^2 \; + \; C \, \|\boldsymbol{\alpha}\|^2 \\ \text{s.t.} & \boldsymbol{K} \, \boldsymbol{\alpha} \; + \; b\mathbf{1} \; = \; \boldsymbol{y} \; + \; \boldsymbol{\xi} \\ & \mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \\ & \mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \end{array} \tag{4.300}$$

or

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha},\boldsymbol{\xi},b} & \|\boldsymbol{\xi}\|^2 \; + \; C \boldsymbol{\alpha}^T \boldsymbol{K} \, \boldsymbol{\alpha} \\ \text{s.t.} & \boldsymbol{K} \, \boldsymbol{\alpha} \; + \; b\mathbf{1} \; = \; \boldsymbol{y} \; + \; \boldsymbol{\xi} \\ & \mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \\ & \mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \, . \end{array} \tag{4.301}$$

Note, that this formulation is very similar to the least square SVM in Subsection 4.13. $\boldsymbol{K} \, \boldsymbol{\alpha} \; = \; \boldsymbol{X}^T \boldsymbol{w}$ if $\boldsymbol{w} \; = \; \boldsymbol{X} \, \boldsymbol{\alpha}$ and $\boldsymbol{K} \; = \; \boldsymbol{X}^T \boldsymbol{X}$. Here the errors per class are forced to have mean zero.

4.19 Software

Online applets for SVM demos are available under

- `http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSVM.html` and
- `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

From these sources some screen shots are made in above examples.

Software can be found under

- `http://www.kernel-machines.org`,
- `http://www.support-vector-machines.org/SVM_stat.html`, and
- `http://kernelsvm.tripod.com`.

Software packages which we recommend are

- **P-SVM:** `http://www.bioinf.jku.at/software/psvm/`.
  - For UNIX and WINDOWS.
  - Matlab front end.
  - Cross-validation included.
  - Feature selection included.
  - Non-positive definite kernels possible.
  - Significance tests.
- **libSVM:** `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.
  - For UNIX and WINDOWS.
  - Sparse matrices.
  - Phyton scripts for hyperparameter selection. Front ends for Python, R (also Splus), MATLAB, Perl, Ruby, Weka, CLISP and LabVIEW.
  - C# .NET code is available.
  - Cross-validation included.
  - Multi-class.
- **SVM-light:** `http://svmlight.joachims.org/`.
  - For UNIX and WINDOWS.
- **SVMTorch:** `http://www.idiap.ch/~bengio/projects/SVMTorch.html` or `http://www.torch.ch/`.
  - UNIX.
  - Sparse matrices.
  - Multi-class.

Chapter 5

Error Minimization and Model Selection

We focus on a model class of parameterized models with parameter vector $\boldsymbol{w}$. Goal is to find or to select the optimal model. The optimal model is the one which optimizes the objective function. To find the optimal model, we have to search in the parameter space.

The objective function is defined by the problem to solve. In general the objective function includes the empirical error and a term which penalizes complexity. *The goal is to find the model from the model class which optimizes the objective function.*

5.1 Search Methods and Evolutionary Approaches

In principle any search algorithm can be used. The simplest was is *random search*, where randomly a parameter vector is selected and then evaluated – the so far best solution is kept.

Another method would be *exhaustive search*, where the parameter space is searched systematically.

These two methods will find the optimal solution for a finite set of possible parameters. They do not use any dependency between objective function and parameter space. For example, if the objective function should be minimized and is 1 for every parameter $\boldsymbol{w}$ except for one parameter $\boldsymbol{w}_{\text{opt}}$ which gives 0. That is a singular solution.

In general there are dependencies between objective function and parameters. These dependencies should be utilized to find the model which optimizes the objective function.

The first dependency is that good solutions are near good solutions in parameter space even if the objective is not continuous and not differentiable. In this case a *stochastic gradient* can be used to locally optimize the objective function. In the context of *genetic algorithms* a stochastic gradient corresponds to mutations which are small. With "small" we mean that every component, e.g. "gene" is mutated only slightly or only one component is mutated at all. In general a stochastic gradient tests solutions which are similar the current best solution or current best solutions. Finally an update of the current best solution is made sometimes by combining different parameter changes which improved the current best solution.

Another dependency is that good solutions share properties of their parameter vector which are independent of other components of the parameter vector. For example if certain dependencies

between specific parameters guarantee good solutions and these specific parameters do not influence other parameters. In this case *genetic algorithms* can utilize these dependencies through the "crossover mutations" where parts of different parameter vectors are combined. Important is that components are independent of other components, i.e. the solutions have different building blocks which can improve the objective independently of other building blocks.

Besides the genetic algorithms there are other evolutionary strategies to optimize models. These strategies include *genetic programming*, *swarm algorithms*, *ant algorithms*, *self-modifying policies*, *Optimal Ordered Problem Solver*, etc. Sometimes the model class is predefined by the search algorithm or the model class is not parameterized. The latter even modify their own search strategy.

All these methods have to deal with local optima of the objective function. To reduce the problem of optima, genetic algorithms search in general at different locations in the parameter space simultaneously. Also other approaches search in parallel at different locations in parameter space.

To overcome the problem of local optima *simulated annealing* (SA) has been suggested. SA can be shown to find the global solution with probability one if the annealing process is slow enough. SA probabilistic jumps from the current state into another state where the probability is given by the objective function. The objective function is transformed to represent an energy function, so that SA jumps into energetically favorable states. The energy function follows the Maxwell-Boltzmann distribution and the sampling is similar to the Metropolis-Hastings algorithm. A global parameter, the temperature, determines which jumps are possible. At the beginning large jumps even into energetically worse regions are possible due to high temperature whereas with low temperature only small jumps are possible and energetically worse regions are avoided. Therefore the parameter space is at the beginning scanned for favorable regions where later is searched in more detail.

Advantages of these methods are that they

- can deal with discrete problems and non-differentiable objective functions and
- are very easy to implement.

Disadvantages of these methods are that they

- are computationally expensive for large parameter spaces and
- depend on the representation of the model.

To see the computational load of stochastic gradient or genetic algorithms assume that the parameter vector has W components. If we only check whether these components should be increased or decreased, we have 2^W decisions to make. When the amount of change also matters then this number of candidates will be soon infeasible to check only to make one improvement step.

The dependency between the parameter space and the objective function which will be of interest in the following is that the objective function is differentiable with respect to the parameters.

Therefore we can make use of gradient information in the search of the (local) minimum.

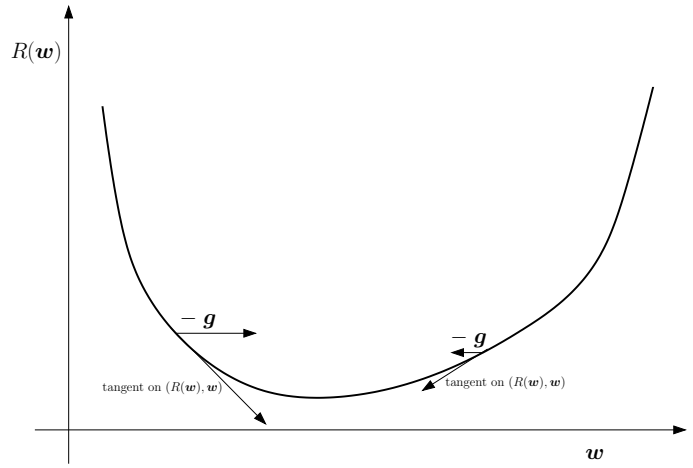


Figure 5.1: The negative gradient $-\boldsymbol{g}$ gives the direction of the steepest decent depicted by the tangent on $(R(\boldsymbol{w}), \boldsymbol{w})$, the error surface.

Problems like that were already treated in Section 4.15 in Subsection 4.15.1. However we focused on convex optimization. There only one minimum exists. Further we treated linear and quadratic objectives. The idea of Lagrange multipliers carries over to constraint optimization in general: assume that the problem is locally convex.

5.2 Gradient Descent

Assume that the objective function $R(g(.; \boldsymbol{w}))$ is a differentiable function with respect to the parameter vector $\boldsymbol{w}$. The function g is the model. For simplicity, we will write $R(\boldsymbol{w}) = R(g(.; \boldsymbol{w}))$.

The gradient is

$$\frac{\partial R(\boldsymbol{w})}{\partial \boldsymbol{w}} = \nabla_{\boldsymbol{w}} R(\boldsymbol{w}) = \left(\frac{\partial R(\boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial R(\boldsymbol{w})}{\partial w_W} \right)^T , \tag{5.1}$$

for a W-dimensional parameter vector $\boldsymbol{w}$.

For the gradient of $R(\boldsymbol{w})$ we use the vector

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} R(\boldsymbol{w}) . \tag{5.2}$$

The negative gradient is the direction of the steepest descent of the objective. The negative gradient is depicted in Fig. 5.1 or a one-dimensional error surface and in Fig. 5.2 for a two-dimensional error surface.

Because the gradient is valid only locally (for nonlinear objectives) we only go a small step in the negative gradient direction if we want to minimize the objective function. The step-size is controlled by $0 < \eta$, the *learning rate*.

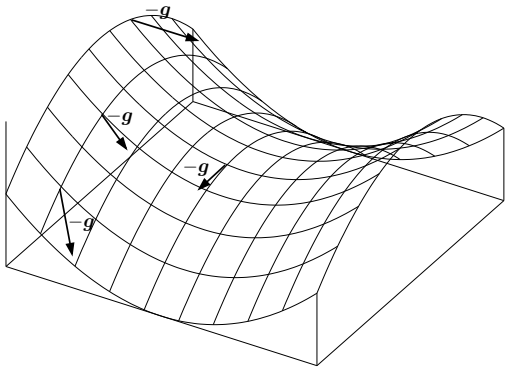


Figure 5.2: The negative gradient $-\boldsymbol{g}$ attached at different positions on an two-dimensional error surface $(R(\boldsymbol{w}), \boldsymbol{w})$. Again the negative gradient gives the direction of the steepest descent.

The gradient descent update is

$$\Delta \boldsymbol{w} \;=\; -\,\eta\, \nabla_{\boldsymbol{w}} R(\boldsymbol{w}) \tag{5.3}$$

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;+\; \Delta \boldsymbol{w}\,. \tag{5.4}$$

Momentum Term. Sometimes gradient descent oscillates or slows down because the error surface (the objective function) has a flat plateau. To avoid oscillation and to speed up gradient descent a momentum term can be used. The oscillation of the gradient is depicted in Fig. 5.3 and the reduction of oscillation through the momentum term in Fig. 5.4.

Another effect of the momentum term is that in flat regions the gradients pointing in the same directions are accumulated and the learning rate is implicitly increased. The problem of flat regions is depicted in Fig. 5.5 and the speed up of the convergence in flat regions in Fig. 5.6.

The gradient descent update with momentum term is

$$\Delta^{\text{new}} \boldsymbol{w} \;=\; -\,\eta\, \nabla_{\boldsymbol{w}} R(\boldsymbol{w}) \tag{5.5}$$

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;+\; \Delta^{\text{new}} \boldsymbol{w} \;+\; \mu\, \Delta^{\text{old}} \tag{5.6}$$

$$\Delta^{\text{old}} \;=\; \Delta^{\text{new}}\,, \tag{5.7}$$

where $0 \;\leq\; \mu \;\leq\; 1$ is the *momentum parameter* or *momentum factor*.

5.3 Step-size Optimization

Instead of choosing a fixed step-size the step-size should be adjusted to the curvature of the error surface. For a flat curve the step-size could be large whereas for high curvature and steep minima the step-size should be small. In Fig. 5.7 the

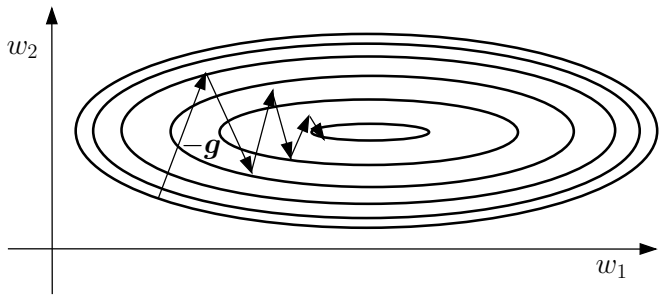


Figure 5.3: The negative gradient $-\boldsymbol{g}$ oscillates as it converges to the minimum.

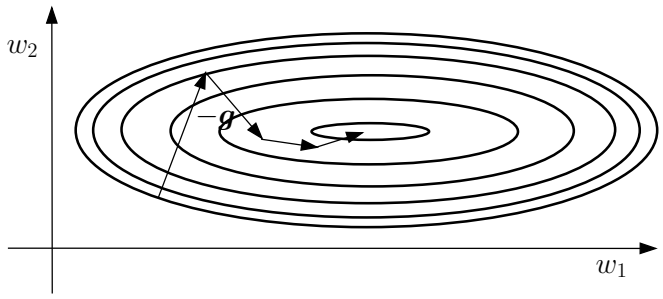


Figure 5.4: Using the momentum term the oscillation of the negative gradient $-\boldsymbol{g}$ is reduced because consecutive gradients which point in opposite directions are superimposed. The minimum found faster.

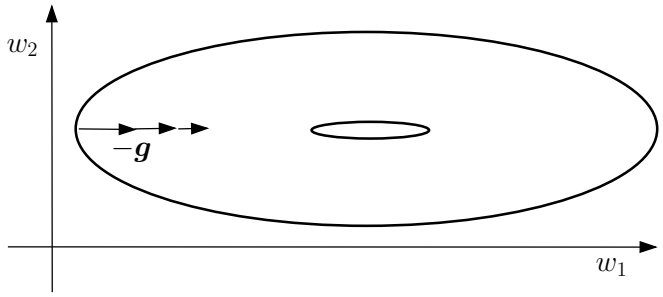


Figure 5.5: The negative gradient $-\boldsymbol{g}$ let the weight vector converge very slowly to the minimum if the region around the minimum is flat.

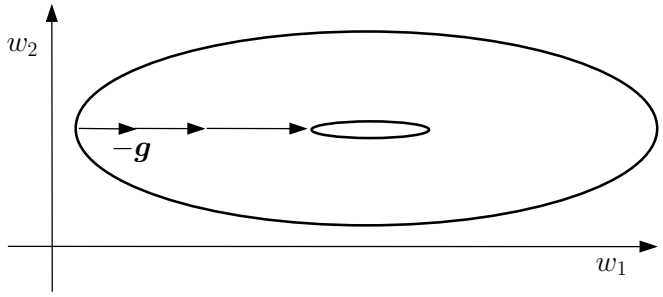


Figure 5.6: The negative gradient $-\boldsymbol{g}$ is accumulates through the momentum term because consecutive gradients point into the same directions. The minimum found faster.

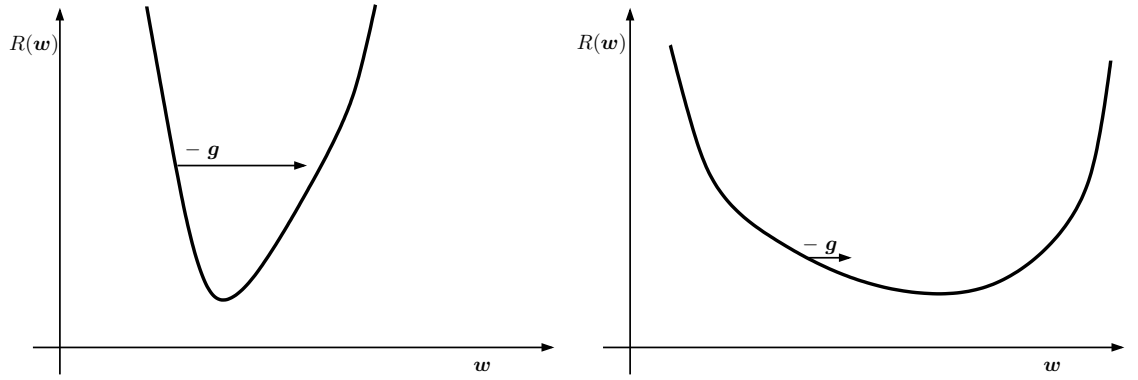


Figure 5.7: Left: the length of the negative gradient $-\boldsymbol{g}$ is large because of the steep descent. However a small gradient step would be optimal because of the high curvature at the minimum. Right: the length of the negative gradient $-\boldsymbol{g}$ is small because of the flat descent. However a large gradient step could be performed because the low curvature ensures that the minimum is not jumped over.

5.3.1 Heuristics

Learning rate adjustments. The learning rate is adjusted: if the change of the risk

$$\Delta R \;=\; R(\boldsymbol{w} \,+\, \Delta\boldsymbol{w}) \;-\; R(\boldsymbol{w}) \tag{5.8}$$

is positive then the learning rate is increased and otherwise decreased:

$$\eta^{\text{new}} \;=\; \begin{cases} \rho\,\eta^{\text{old}} & \text{if} \quad \Delta R \;\leq\; 0 \\ \sigma\,\eta^{\text{old}} & \text{if} \quad \Delta R \;>\; 0 \end{cases} \quad , \tag{5.9}$$

where $\rho > 1$ and $\sigma < 1$, e.g. $\rho = 1.1$ and $\sigma = 0.5$. Here ρ is only slightly larger than one but sigma is much smaller than one in order to reduce too large learning rates immediately.

Largest eigenvalue of the Hessian. Later in eq. (5.76) we will see that the largest eigenvalue $\lambda_{\max}$ of the Hessian, $\boldsymbol{H}$ given as

$$H_{ij} \;=\; \frac{\partial^2 R(\boldsymbol{w})}{\partial w_i\,\partial w_j}\;, \tag{5.10}$$

bounds the learning rate:

$$\eta \;\leq\; \frac{2}{\lambda_{\max}}\;. \tag{5.11}$$

The idea is to efficiently infer the largest eigenvalue of the Hessian in order to find a maximal learning rate. The maximal eigenvalue of the Hessian can be determined by matrix iteration. Let $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_W)$ be the from largest to smallest sorted eigenvectors of the Hessian with according eigenvalues $(\lambda_1, \ldots, \lambda_W)$ and let $\boldsymbol{a} \;=\; \sum_{i=1}^{W} \alpha_i\,\boldsymbol{e}_i$, then

$$\boldsymbol{H}^s\,\boldsymbol{a} \;=\; \sum_{i=1}^{W} \lambda_i^s\,\alpha_i\,\boldsymbol{e}_i \;\approx\; \lambda_1^s\,\alpha_1\,\boldsymbol{e}_1\;. \tag{5.12}$$

Normalizing $\boldsymbol{H}^s\ \boldsymbol{a}$ and multiplication with $\boldsymbol{H}$ gives both a hint how well $\boldsymbol{e}_1$ is approximated and how large λ_1 may be.

The method depends how time intensive it is to obtain the Hessian or the product of the Hessian with a vector. How the Hessian is computed depends on the model. For quadratic problems the Hessian is directly given and for neural networks there exist different methods to approximate the Hessian or to compute the product of the Hessian with a vector. See Section 6.4.4 for more details.

Individual learning rate for each parameter.

We have for parameter w_i

$$\Delta w_i \ = \ - \eta_i \, g_i \ , \tag{5.13}$$

where g_i is the i-th component of the gradient $\boldsymbol{g}$: $g_i \ = \ \left[\frac{\partial R(\boldsymbol{w})}{\partial \boldsymbol{w}}\right]_i$. If the parameters would be independent from each other, then individual learning rates are justified. However for dependent parameter, as long as all $\eta_i > 0$ the error decreases.

The *delta-delta rule* adjusts the local learning parameter according to

$$\Delta \eta_i \ = \ \gamma \, g_i^{\text{new}} \, g_i^{\text{old}} \ . \tag{5.14}$$

The delta-delta rule determines whether the local gradient information changes. If it changes then the learning rate is decreased otherwise it is increased. In a flat plateau the learning rate is increased and in steep regions the learning rate is decreased.

This rule was improved to the *delta-bar-delta rule*:

$$\Delta \eta_i \ = \ \begin{cases} \kappa & \text{if} \quad \bar{g}_i^{\text{old}} \, g_i^{\text{new}} \ > \ 0 \\ - \, \phi \, g_i^{\text{new}} & \text{if} \quad \bar{g}_i^{\text{old}} \, g_i^{\text{new}} \ \leq \ 0 \end{cases} \ , \tag{5.15}$$

where

$$\bar{g}_i^{\text{new}} \ = \ (1 \ - \ \theta) \, g_i^{\text{new}} \ + \ \theta \, \bar{g}_i^{\text{old}} \ . \tag{5.16}$$

$\bar{g}_i$ is an exponentially weighted average of the values of g_i. That means instead of the old gradient information an average is used to determine whether the local gradient directions changes.

Big disadvantage of the delta-bar-delta rule that it has many hyper-parameters: θ, κ, ϕ and μ if a momentum term is included as well.

Quickprop. This method was developed in the context of neural networks, therefore the name "quickprop" which reminds on the name "back-propagation" the most popular method for computing the gradient in neural networks.

Here again the parameters are treated individually. The *quickprop* learning rule is

$$\Delta^{\text{new}} w_i \ = \ \frac{g_i}{g_i^{\text{old}} \ - g_i^{\text{new}}} \, \Delta^{\text{old}} w_i \ . \tag{5.17}$$

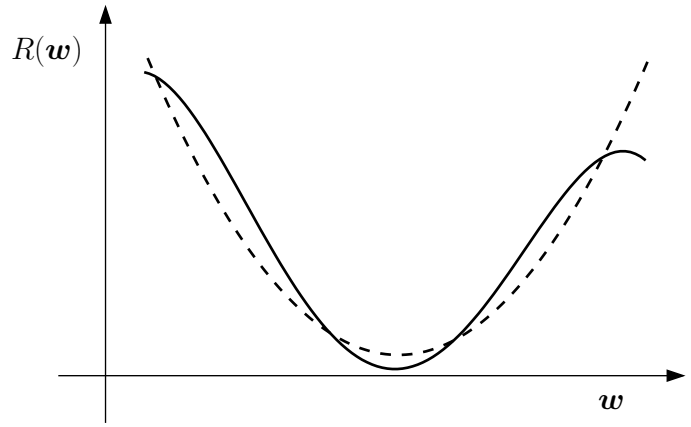


Figure 5.8: The error surface (the solid curve) is locally approximated by a quadratic function (the dashed curve).

Let us assume that $R(\boldsymbol{w})$ is a function of one variable $R(w_i)$ then the Taylor expansion is

$$\begin{aligned} R(w_i + \Delta w_i) &= R(w_i) + \frac{\partial R(\boldsymbol{w})}{\partial w_i}\,\Delta w_i + \\ &\frac{1}{2}\,\frac{\partial^2 R(\boldsymbol{w})}{(\partial w_i)^2}\,(\Delta w_i)^2 + O\left((\Delta w_i)^3\right) = \\ &R(w_i) + g_i\,\Delta w_i + \frac{1}{2}\,g_i'\,(\Delta w_i)^2 + O\left((\Delta w_i)^3\right)\,. \end{aligned} \tag{5.18}$$

That is a quadratic approximation of the function. See Fig. 5.8 for the quadratic approximation of the error function $R(\boldsymbol{w})$.

To minimize $R(w_i + \Delta w_i) - R(w_i)$ with respect to Δw_i we set the derivative of the right hand side to zero and obtain

$$\Delta w_i = -\frac{g_i}{g_i'}\,. \tag{5.19}$$

Now approximate $g_i' = g_i'(w_i)$ by $g_i'(w_i^{\mathrm{old}})$ and use a difference quotient:

$$g_i' = \frac{g_i^{\mathrm{new}} - g_i^{\mathrm{old}}}{\Delta^{\mathrm{old}} w_i}\,, \tag{5.20}$$

where $g_i^{\mathrm{old}} = g_i(w_i - \Delta^{\mathrm{old}} w_i)$. We now insert this approximation into eq. (5.19) which results in the quickprop update rule.

5.3.2 Line Search

Let us assume we found the update direction $\boldsymbol{d}$ either as the negative gradient or by also taking into account the curvature by the Hessian or its approximation.

The update rule is

$$\Delta \boldsymbol{w} \; = \eta \; \boldsymbol{d} \; . \tag{5.21}$$

We now want to find the value of η which minimizes

$$R(\boldsymbol{w} \; + \; \eta \; \boldsymbol{d}) \; . \tag{5.22}$$

For quadratic functions and $\boldsymbol{d} \; = \; - \; \boldsymbol{g}$ and very close to the minimum $\boldsymbol{w}^*$, η is given later in eq. (5.77) as

$$\eta \; = \; \frac{\boldsymbol{g}^T \; \boldsymbol{g}}{\boldsymbol{g}^T \; \boldsymbol{H}(\boldsymbol{w}^*) \; \boldsymbol{g}} \; . \tag{5.23}$$

However this is only valid near the minimum. Further we do not know $\boldsymbol{H}(\boldsymbol{w}^*)$.

Finding the best update step could be viewed as a separate task. If we assume that at the minimum the function $R(\boldsymbol{w})$ is convex then we can apply *line search*.

Line search fits first a parable though three points and determines its minimum. The point with largest value of R is discharged. The line search algorithm is given in Alg. 5.1.

Algorithm 5.1 Line Search

BEGIN initialization $a_0, b_0, c_0; R(a_0) > R(c_0); R(b_0) > R(c_0)$, Stop=false, $i = 0$
END initialization
BEGIN line search
  while STOP=false **do**
    fit quadratic function through a_i, b_i, c_i
    determine minimum d_i of quadratic function
    if stop criterion fulfilled, e.g. $|a_i - b_i| < \epsilon$ or $|R(b_0) - R(c_0)| < \epsilon$ **then**
      Stop=true
    else
      $c_{i+1} \; = \; d_i$
      $b_{i+1} \; = \; c_i$
      $a_{i+1} \; = \; \begin{cases} a_i & \text{if} \quad R(a_i) \; \leq \; R(b_i) \\ b_i & \text{if} \quad R(a_i) \; > \; R(b_i) \end{cases}$
    end if
    $i = i + 1$
  end while
END line search

Fig. 5.9 shows the line search procedure. At each step the length of the interval $[a, b]$ or $[b, a]$ is decreased.

Line search starts with $R(\boldsymbol{w}), R(\boldsymbol{w} \; + \; \eta_0 \boldsymbol{d})$ and $R(\boldsymbol{w} \; + \; \eta_{\max} \; \boldsymbol{d})$, where $a_0, b_0, c_0 \; \in \{eta_0, 0, eta_{\max}\}$. If a large range of η values are possible, then the search can be on a logarithmic scale as preprocessing.

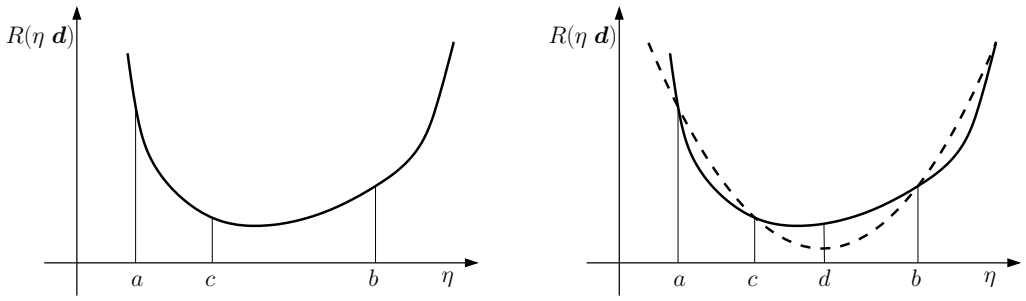


Figure 5.9: Line search. Left: the error function $R(\eta \boldsymbol{d})$ in search direction $\boldsymbol{d}$ is depicted as solid curve. The scalars a, b, c are given with $R(a\ \boldsymbol{d}) > R(c\boldsymbol{d})$ and $R(b\ \boldsymbol{d}) > R(c\ \boldsymbol{d})$. Right: The dashed curve is a quadratic approximation (parabola) of the error function by fitting the parabola through a, b, and c. The minimum of the parabola is at d. For the next step we set $a = b$, $b = c$, and $c = d$.

5.4 Optimization of the Update Direction

The default direction is the negative gradient $-\boldsymbol{g}$. However there are even better approaches.

5.4.1 Newton and Quasi-Newton Method

The gradient vanishes because it is a minimum $\boldsymbol{w}^*$: $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) = \boldsymbol{g}(\boldsymbol{w}^*) = \boldsymbol{0}$. Therefore we obtain for the Taylor series around $\boldsymbol{w}^*$:

$$R(\boldsymbol{w}) = R(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}^*)\ (\boldsymbol{w} - \boldsymbol{w}^*) + O\left(\|\boldsymbol{w} - \boldsymbol{w}^*\|^3\right) . \tag{5.24}$$

The gradient $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{w})$ of the quadratic approximation of $\mathbb{R}(\boldsymbol{w})$ is

$$\boldsymbol{g} = \boldsymbol{H}(\boldsymbol{w}^*)\ (\boldsymbol{w} - \boldsymbol{w}^*) . \tag{5.25}$$

Solving this equation for $\boldsymbol{w}^*$ gives

$$\boldsymbol{w}^* = \boldsymbol{w} - \boldsymbol{H}^{-1}\boldsymbol{g} . \tag{5.26}$$

The update direction $\boldsymbol{H}^{-1}\boldsymbol{g}$ is the *Newton direction*. The Newton direction is depicted in Fig. 5.10 for a quadratic error surface.

Disadvantages of the Newton direction are

- that it is computationally expensive because computing the Hessian is expensive and its inversion needs $O(W^3)$ steps;
- that it only works near the minimum if the Hessian is positive definite.

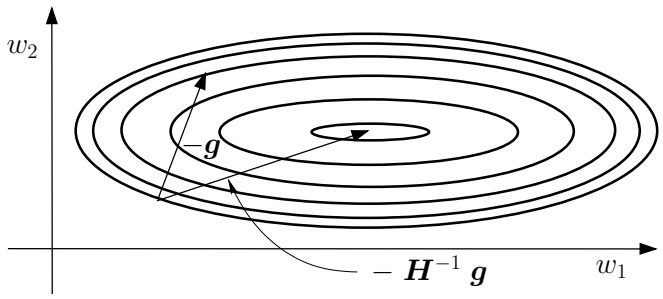


Figure 5.10: The Newton direction $-\boldsymbol{H}^{-1}\,\boldsymbol{g}$ for a quadratic error surface in contrast to the gradient direction $-\boldsymbol{g}$. The Newton direction points directly to the minimum and one Newton update step would find the minimum.

A remedy for the later disadvantage is the *model trust region* approach, where the model is only trusted up to a certain value. A positive definite matrix is added to the Hessian:

$$\boldsymbol{H} \;+\; \lambda \boldsymbol{I}\;. \tag{5.27}$$

The update step is a compromise between gradient direction (large λ) and Newton direction ($lambda \;=\; 0$).

To address the problem of the expensive inversion of the Hessian, it can be approximated by a diagonal matrix. The diagonal matrix is simple to invert.

Quasi-Newton Method.

From the Newton equation eq. (5.26) two weight vectors $\boldsymbol{w}^{\mathrm{old}}$ and $\boldsymbol{w}^{\mathrm{new}}$ are related by

$$\boldsymbol{w}^{\mathrm{new}} \;-\; \boldsymbol{w}^{\mathrm{old}} \;=\; -\,\boldsymbol{H}^{-1}\left(\boldsymbol{g}^{\mathrm{new}} \;-\; \boldsymbol{g}^{\mathrm{old}}\right)\;. \tag{5.28}$$

This is the quasi-Newton condition.

The best known method is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) method. The BFGS update is

$$\boldsymbol{w}^{\mathrm{new}} \;=\; \boldsymbol{w}^{\mathrm{old}} \;+\; \eta\,\boldsymbol{G}^{\mathrm{old}}\,\boldsymbol{g}^{\mathrm{old}}\;, \tag{5.29}$$

where η is found by line search.

The function $\boldsymbol{G}$ is an approximation of the inverse Hessian and is computed as follows:

$$\boldsymbol{G}^{\mathrm{new}} \;=\; \boldsymbol{G}^{\mathrm{old}} \;+\; \frac{\boldsymbol{p}\,\boldsymbol{p}^T}{\boldsymbol{p}^T\,\boldsymbol{v}} \;- \tag{5.30}$$
$$\frac{\left(\boldsymbol{G}^{\mathrm{old}}\,\boldsymbol{v}\right)\;\boldsymbol{v}^T\boldsymbol{G}^{\mathrm{old}}}{\boldsymbol{v}^T\boldsymbol{G}^{\mathrm{old}}\,\boldsymbol{v}} \;+\; \left(\boldsymbol{v}^T\boldsymbol{G}^{\mathrm{old}}\,\boldsymbol{v}\right)\;\boldsymbol{u}\,\boldsymbol{u}^T\;,$$

where

$$\boldsymbol{p} \;=\; \boldsymbol{w}^{\mathrm{new}} \;-\; \boldsymbol{w}^{\mathrm{old}} \tag{5.31}$$
$$\boldsymbol{v} \;=\; \boldsymbol{g}^{\mathrm{new}} \;-\; \boldsymbol{g}^{\mathrm{old}} \tag{5.32}$$
$$\boldsymbol{u} \;=\; \frac{\boldsymbol{p}}{\boldsymbol{p}^T\boldsymbol{v}} \;-\; \frac{G^{\mathrm{old}}\boldsymbol{v}}{\boldsymbol{v}^T\boldsymbol{G}^{\mathrm{old}}\,\boldsymbol{v}}\;. \tag{5.33}$$

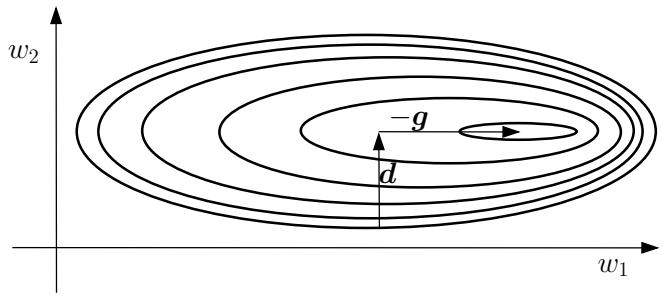


Figure 5.11: Conjugate gradient. After line search in search direction $\boldsymbol{d}$ the new gradient is orthogonal to the line search direction.

Initialization of $\boldsymbol{G}$ can be done by $\boldsymbol{I}$, the identity matrix (only a gradient step as first step).

5.4.2 Conjugate Gradient

Note that for the line search algorithm we optimized η so that

$$R(\boldsymbol{w} \, + \, \eta \, \boldsymbol{d}) \tag{5.34}$$

is minimized.

The minimum condition is

$$\frac{\partial}{\partial \eta} \, R(\boldsymbol{w} \, + \, \eta \, \boldsymbol{d}) \, = \, 0 \, , \tag{5.35}$$

which gives

$$(\boldsymbol{g}^{\mathrm{new}})^T \; \boldsymbol{d}^{\mathrm{old}} \, = \, 0 \, . \tag{5.36}$$

The gradient of the new minimum $\boldsymbol{g}^{\mathrm{new}}$ is orthogonal to the previous search direction $\boldsymbol{d}^{\mathrm{old}}$. This fact is depicted in Fig. 5.11.

However still oscillations are possible. The oscillations appear especially in higher dimensions, where oscillations like in Fig. 5.3 can be present in different two-dimensional subspaces which alternate. Desirable to avoid oscillations would be that a new search directions are orthogonal to all previous search directions where orthogonal is defined via the Hessian. The later means that in the parameter space not all directions are equal important. In the following we construct such search directions.

The oscillations can be avoided if we enforce not only

$$\boldsymbol{g}(\boldsymbol{w}^{\mathrm{new}})^T \, \boldsymbol{d}^{\mathrm{old}} \, = \, 0 \tag{5.37}$$

but also the same condition for the new gradient

$$\boldsymbol{g}(\boldsymbol{w}^{\mathrm{new}} \, + \, \eta \boldsymbol{d}^{\mathrm{new}})^T \, \boldsymbol{d}^{\mathrm{old}} \, = \, 0 \, . \tag{5.38}$$

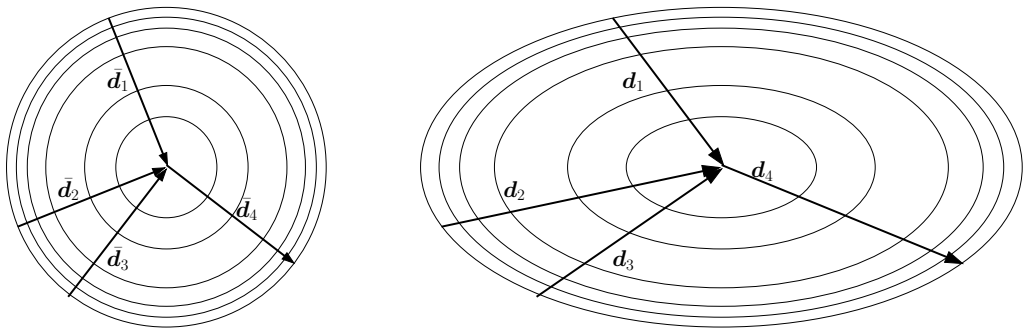


Figure 5.12: Conjugate gradient. Left: $\bar{\boldsymbol{d}}_1^T\ \bar{\boldsymbol{d}}_2\ =\ 0$ and $\bar{\boldsymbol{d}}_3^T\ \bar{\boldsymbol{d}}_4\ =\ 0$. Right: $\boldsymbol{d}_1^T\ \boldsymbol{H}\ \boldsymbol{d}_2\ =\ 0$ and $\boldsymbol{d}_3^T\ \boldsymbol{H}\ \boldsymbol{d}_4\ =\ 0$. Conjugate directions can be viewed as orthogonal directions which were transformed by a change of the coordinate system: $\boldsymbol{d}\ =\ \boldsymbol{H}^{-1/2}\bar{\boldsymbol{d}}$.

Taylor expansion of $\boldsymbol{g}(\boldsymbol{w}^{\text{new}}\ +\ \eta\boldsymbol{d}^{\text{new}})$ with respect to η around 0 gives

$$\boldsymbol{g}(\boldsymbol{w}^{\text{new}}\ +\ \eta\boldsymbol{d}^{\text{new}})^T\ =\ \boldsymbol{g}(\boldsymbol{w}^{\text{new}})\ +\ \eta\boldsymbol{H}(\boldsymbol{w}^{\text{new}})\boldsymbol{d}^{\text{new}}\ +\ O(\eta^2)\,. \tag{5.39}$$

We insert that into eq. (5.38) and apply eq. (5.37) and divide through η and obtain

$$(\boldsymbol{d}^{\text{new}})^T\ \boldsymbol{H}(\boldsymbol{w}^{\text{new}})\ \boldsymbol{d}^{\text{old}}\ =\ 0\,. \tag{5.40}$$

Directions which satisfy eq. (5.40) are said to be *conjugate*. See Fig. 5.12 for conjugate directions.

Let us assume a quadratic problem

$$R(\boldsymbol{w})\ =\ \frac{1}{2}\,\boldsymbol{w}^T\boldsymbol{H}\boldsymbol{w}\ +\ \boldsymbol{c}^T\boldsymbol{w}\ +\ k \tag{5.41}$$

with

$$\boldsymbol{g}(\boldsymbol{w})\ =\ \boldsymbol{H}\boldsymbol{w}\ +\ \boldsymbol{c} \tag{5.42}$$

and

$$\boldsymbol{0}\ =\ \boldsymbol{H}\boldsymbol{w}^*\ +\ \boldsymbol{c}\,. \tag{5.43}$$

We want to have W conjugate directions, i.e.

$$\forall_{i\neq j}:\ \ \boldsymbol{d}_j^T\boldsymbol{H}\ \boldsymbol{d}_i\ =\ 0\,, \tag{5.44}$$

which are linearly independent, therefore we can set

$$\boldsymbol{w}^*\ -\ \boldsymbol{w}_1\ =\ \sum_{i=1}^{W}\eta_i\ \boldsymbol{d}_i \tag{5.45}$$

$$\tag{5.46}$$

and

$$\boldsymbol{w}_j \; - \; \boldsymbol{w}_1 \; = \; \sum_{i=1}^{j-1} \eta_i \; \boldsymbol{d}_i \; . \tag{5.47}$$

In the last but one equation $\boldsymbol{w}^* \; = \; - \; \boldsymbol{H}^{-1} \; \boldsymbol{c}$ is multiplied by $\boldsymbol{d}_j^T \boldsymbol{H}$ and we obtain

$$- \; \boldsymbol{d}_j^T \; (\boldsymbol{c} \; + \; \boldsymbol{H} \boldsymbol{w}_1) \; = \; \sum_{i=1}^{W} \eta_i \; \boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{d}_i \; = \; \eta_j \; \boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{d}_j \; . \tag{5.48}$$

From eq. (5.47) we obtain

$$\boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{w}_j \; = \; \boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{w}_1 \; . \tag{5.49}$$

Using $\boldsymbol{g}_j \; = \; \boldsymbol{c} \; + \; \boldsymbol{H} \boldsymbol{w}_j$, η_j can be determined as

$$\eta_j \; = \; - \; \frac{\boldsymbol{d}_j^T \boldsymbol{g}_j}{\boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{d}_j} \; . \tag{5.50}$$

Because of

$$\boldsymbol{w}_{j+1} \; = \; \boldsymbol{w}_j \; + \; \eta_j \; \boldsymbol{d}_j \; . \tag{5.51}$$

we have determined the learning rate η_j.

Now we have find the search directions $\boldsymbol{d}_j$. We set

$$\boldsymbol{d}_{j+1} \; = \; - \; \boldsymbol{g}_{j+1} \; + \; \beta_j \; \boldsymbol{d}_j \; . \tag{5.52}$$

Multiplying by $\boldsymbol{d}_j^T \boldsymbol{H}$ gives

$$\beta_j \; = \; \frac{\boldsymbol{g}_{j+1}^T \boldsymbol{H} \; \boldsymbol{d}_j}{\boldsymbol{d}_j^T \boldsymbol{H} \; \boldsymbol{d}_j} \; . \tag{5.53}$$

Because of

$$\boldsymbol{g}_{j+1} \; - \; \boldsymbol{g}_j \; = \; \boldsymbol{H} \; (\boldsymbol{w}_{j+1} \; - \; \boldsymbol{w}_j) \; = \; \eta_j \; \boldsymbol{H} \; \boldsymbol{d}_j \tag{5.54}$$

we can rewrite eq. (5.53) as

$$\textbf{Hestenes} - \textbf{Stiefel} : \tag{5.55}$$

$$\beta_j \; = \; \frac{\boldsymbol{g}_{j+1}^T \; (\boldsymbol{g}_{j+1} \; - \; \boldsymbol{g}_j)}{\boldsymbol{d}_j^T \; (\boldsymbol{g}_{j+1} \; - \; \boldsymbol{g}_j)} \; . \tag{5.56}$$

Multiplying eq. (5.52) by $\boldsymbol{g}_{j+1}^T$ and using the conditions $\boldsymbol{d}_k^T\ \boldsymbol{g}_j\ =\ 0$ for $k < j$ gives

$$\boldsymbol{d}_j^T\ \boldsymbol{g}_j\ =\ -\ \boldsymbol{g}_j^T\ \boldsymbol{g}_j\ . \tag{5.57}$$

The equation for β_j can be rewritten as

$$\mathbf{Polak-Ribiere}: \tag{5.58}$$

$$\beta_j\ =\ \frac{\boldsymbol{g}_{j+1}^T\ (\boldsymbol{g}_{j+1}\ -\ \boldsymbol{g}_j)}{\boldsymbol{g}_j^T\ \boldsymbol{g}_j}\ . \tag{5.59}$$

Similar to previous reformulation this expression can be simplified to

$$\mathbf{Fletcher-Reeves}: \tag{5.60}$$

$$\beta_j\ =\ \frac{\boldsymbol{g}_{j+1}^T\boldsymbol{g}_{j+1}}{\boldsymbol{g}_j^T\ \boldsymbol{g}_j}\ . \tag{5.61}$$

Even if the equations eq. (5.55), eq. (5.58), and eq. (5.60) are mathematically equivalent, there are numerical differences. The Polak-Ribiere equation eq. (5.58) has an edge over the other update rules.

The computation of the values η_j need the Hessian, therefore the η_j are in most implementations found by line search.

Algorithm 5.2 Conjugate Gradient (Polak-Ribiere)

BEGIN initialization $\boldsymbol{g}_0\ =\ \nabla_{\boldsymbol{w}} R(\boldsymbol{w}_0)$, $i = 0$,$d_0\ =\ -\boldsymbol{g}_0$, Stop=false
END initialization
BEGIN Conjugate Gradient
  while STOP=false **do**
    determine η_i by line search
    $\boldsymbol{w}_{j+1}\ =\ \boldsymbol{w}_j\ +\ \eta_j\ \boldsymbol{d}_j$
    $\boldsymbol{g}_{i+1}\ =\ \nabla_{\boldsymbol{w}} R(\boldsymbol{w}_i)$
    $\beta_j\ =\ \frac{\boldsymbol{g}_{j+1}^T(\boldsymbol{g}_{j+1}\ -\ \boldsymbol{g}_j)}{\boldsymbol{g}_j^T\ \boldsymbol{g}_j}$
    $\boldsymbol{d}_{j+1}\ =\ -\ \boldsymbol{g}_{j+1}\ +\ \beta_j\ \boldsymbol{d}_j$
    if stop criterion fulfilled, e.g. $\|g_{i+1}\|| < \epsilon$ or $|R(\boldsymbol{w}_{j+1}) - R(\boldsymbol{w}_j)| < \epsilon$ **then**
      STOP=true
    end if
    $i = i + 1$
  end while
END Conjugate Gradient

The disadvantage of conjugate gradient compared to the quasi-Newton methods is

- the line search must be done precisely in order to obtain the conjugate and orthogonal gradients

The advantage of conjugate gradient compared to the quasi-Newton methods is

- that the storage is $O(W)$ compared to $O(W^2)$ for quasi-Newton

5.5 Levenberg-Marquardt Algorithm

This algorithm is designed for quadratic loss, i.e. for the mean squared error

$$R(\boldsymbol{w}) \;=\; \sum_{i=1}^{l} \left(e^i(\boldsymbol{w})\right)^2 \;. \tag{5.62}$$

We combine the errors e^i into a vector $\boldsymbol{e}$. The Jacobian of this vector is $\boldsymbol{Z}$ defined as

$$Z_{ij} \;=\; \frac{\partial e^i}{\partial w_j} \;. \tag{5.63}$$

The linear approximation of the error vector gives

$$\boldsymbol{e}(\boldsymbol{w}^{\text{new}}) \;=\; \boldsymbol{e}(\boldsymbol{w}^{\text{old}}) \;+\; \boldsymbol{Z}\left(\boldsymbol{w}^{\text{new}} \;-\; \boldsymbol{w}^{\text{old}}\right) \;. \tag{5.64}$$

The Hessian of the loss function $R(\boldsymbol{w})$ is

$$H_{jk} \;=\; \frac{\partial^2 R}{\partial w_j\, \partial w_k} \;=\; \sum_{i=1}^{l} \left(\frac{\partial e^i}{\partial w_j}\, \frac{\partial e^i}{\partial w_k} \;+\; e^i\, \frac{\partial^2 e^i}{\partial w_j\, \partial w_k} \right) \;. \tag{5.65}$$

If we assume small e^i (if we are close to the targets) or if we assume that the term $e^i\, \frac{\partial^2 e^i}{\partial w_j\, \partial w_k}$ averages out, then we can approximate the Hessian by

$$\boldsymbol{H} \;=\; \boldsymbol{Z}^T \boldsymbol{Z} \;. \tag{5.66}$$

Note that this "outer product approximation" is only valid for quadratic loss functions.

We now can formulate a unconstraint minimization problem where

$$\frac{1}{2} \left\| \boldsymbol{e}(\boldsymbol{w}^{\text{old}}) \;+\; \boldsymbol{Z}\left(\boldsymbol{w}^{\text{new}} \;-\; \boldsymbol{w}^{\text{old}}\right) \right\|^2 \;+\; \lambda \left\| \boldsymbol{w}^{\text{new}} \;-\; \boldsymbol{w}^{\text{old}} \right\|^2 \tag{5.67}$$

has to be minimized. The first term accounts for minimizing the error and the second term for minimizing the step size.

The solution is the *Levenberg-Marquardt* update rule

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;-\; \left(\boldsymbol{Z}^T \boldsymbol{Z} \;+\; \lambda\, \boldsymbol{I}\right)^{-1} \boldsymbol{Z}^T\, \boldsymbol{e}(\boldsymbol{w}^{\text{old}}) \;. \tag{5.68}$$

Small λ gives the Newton formula while large λ gives gradient descent.

The Levenberg-Marquardt algorithm is a model trust region approach.

5.6 Predictor Corrector Methods for $R(\boldsymbol{w}) = 0$

The problem is to solve $R(\boldsymbol{w}) = 0$. That problem is different from minimization problems.

The idea is to make a simple update (predictor step) by neglecting higher order terms. In a second step (corrector step) the value is corrected by involving the higher order terms. Here the higher order terms are evaluated with the solution obtained by the predictor step.

The Taylor series of $R(\boldsymbol{w}^{\mathrm{new}})$ is

$$R(\boldsymbol{w}^{\mathrm{new}}) = R(\boldsymbol{w}^{\mathrm{old}}) + S(\boldsymbol{w}^{\mathrm{old}}, \Delta\boldsymbol{w}) + TS(\boldsymbol{w}^{\mathrm{old}}, \Delta\boldsymbol{w}) \,. \tag{5.69}$$

Here $S(\boldsymbol{w}^{\mathrm{old}}, \boldsymbol{0}) = 0$ and $T(\boldsymbol{w}^{\mathrm{old}}, \boldsymbol{0}) = 0$.

In the predictor step solve

$$R(\boldsymbol{w}^{\mathrm{old}}) + S(\boldsymbol{w}^{\mathrm{old}}, \Delta\boldsymbol{w}) = 0 \tag{5.70}$$

with respect to $\Delta\boldsymbol{w}$ which gives $\Delta_{\mathrm{pred}}\boldsymbol{w}$. In the corrector step solve

$$R(\boldsymbol{w}^{\mathrm{old}}) + S(\boldsymbol{w}^{\mathrm{old}}, \Delta\boldsymbol{w}) + TS(\boldsymbol{w}^{\mathrm{old}}, \Delta_{\mathrm{pred}}\boldsymbol{w}) = 0 \,, \tag{5.71}$$

which gives the final update $\Delta\boldsymbol{w}$.

The predictor-corrector update can be formulated as an iterative algorithm.

5.7 Convergence Properties

Gradient Descent. We make a Taylor series expansion of the gradient function g locally at the minimum $\boldsymbol{w}^*$. The gradient vanishes because it is a minimum $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) = \boldsymbol{g}(\boldsymbol{w}^*) = \boldsymbol{0}$, therefore we obtain

$$\begin{aligned} &R(\boldsymbol{w}) = R(\boldsymbol{w}^*) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}^*) (\boldsymbol{w} - \boldsymbol{w}^*) + \\ &O\left(\|\boldsymbol{w} - \boldsymbol{w}^*\|^3 \right) \,. \end{aligned} \tag{5.72}$$

The improvement of the risk is

$$\begin{aligned} &R(\boldsymbol{w} - \eta\, \boldsymbol{g}) - R(\boldsymbol{w}^*) = \\ &\frac{1}{2} (\boldsymbol{w} - \eta\, \boldsymbol{g} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}^*) (\boldsymbol{w} - \eta\, \boldsymbol{g} - \boldsymbol{w}^*) = \\ &\frac{1}{2} \eta^2\, \boldsymbol{g}^T \boldsymbol{H}(\boldsymbol{w}^*)\, \boldsymbol{g} - \eta\, \boldsymbol{g}^T \boldsymbol{g} + c \,, \end{aligned} \tag{5.73}$$

where c is independent of η.

First we note that

$$\frac{1}{2} \eta^2\, \boldsymbol{g}^T \boldsymbol{H}(\boldsymbol{w}^*)\, \boldsymbol{g} - \eta\, \boldsymbol{g}^T \boldsymbol{g} \leq 0 \,, \tag{5.74}$$

to ensure improvement for arbitrary c. That is equivalent to

$$\eta \;\leq\; \frac{2\,\boldsymbol{g}^T\,\boldsymbol{g}}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}}\;, \tag{5.75}$$

which gives the bound (the $\boldsymbol{g}$ which gives the smallest value)

$$\eta \;\leq\; \frac{2}{\lambda_{\max}}\;. \tag{5.76}$$

The optimal update is obtained if we minimize the left hand side of eq. (5.74) with respect to η. Setting the derivative with respect to η to zero an solving for η gives

$$\eta \;=\; \frac{\boldsymbol{g}^T\,\boldsymbol{g}}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}}\;. \tag{5.77}$$

The update is

$$\boldsymbol{w}^{\mathrm{new}} \;=\; \boldsymbol{w}^{\mathrm{old}} \;-\; \frac{\boldsymbol{g}^T\,\boldsymbol{g}}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}}\,\boldsymbol{g}\;. \tag{5.78}$$

The improvement is

$$\begin{aligned} & R\left(\boldsymbol{w}^{\mathrm{old}}\right) \;-\; R\left(\boldsymbol{w}^{\mathrm{new}}\right) \;= \\ & \left(\boldsymbol{w}^{\mathrm{old}} \;-\; \boldsymbol{w}^*\right)\,\boldsymbol{H}(\boldsymbol{w}^*)\,\frac{\boldsymbol{g}^T\,\boldsymbol{g}}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}}\boldsymbol{g} \;- \\ & \frac{1}{2}\,\left(\frac{\boldsymbol{g}^T\,\boldsymbol{g}}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}}\right)^2\,\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g} \;= \\ & \frac{1}{2}\,\frac{\left(\boldsymbol{g}^T\,\boldsymbol{g}\right)^2}{\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}} \;=\; R\left(\boldsymbol{w}^{\mathrm{old}}\right)\left(\frac{\left(\boldsymbol{g}^T\,\boldsymbol{g}\right)^2}{\left(\boldsymbol{g}^T\,\boldsymbol{H}(\boldsymbol{w}^*)\,\boldsymbol{g}\right)\,\left(\boldsymbol{g}^T\,\boldsymbol{H}^{-1}(\boldsymbol{w}^*)\,\boldsymbol{g}\right)}\right)\;. \end{aligned} \tag{5.79}$$

The Kantorovich inequality states that

$$\frac{\left(\boldsymbol{g}^T\,\boldsymbol{g}\right)^2}{\left(\boldsymbol{g}^T\,\boldsymbol{H}\,\boldsymbol{g}\right)\,\left(\boldsymbol{g}^T\,\boldsymbol{H}^{-1}\,\boldsymbol{g}\right)} \;\geq\; \frac{4\,\lambda_{\min}\,\lambda_{\max}}{\left(\lambda_{\min}\;+\;\lambda_{\max}\right)^2} \;\geq\; \frac{1}{\mathrm{cond}(\boldsymbol{H})}\;, \tag{5.80}$$

where and $\lambda\max$ and $\lambda_{\min}$ are the maximal and minimal eigenvalues of the Hessian, respectively, and $\mathrm{cond}(\boldsymbol{H})$ is the condition of a matrix

$$\mathrm{cond}(\boldsymbol{H}) \;=\; \frac{\lambda_{\max}}{\lambda_{\min}}\;. \tag{5.81}$$

The improvement depends strongly on the condition of the matrix $\boldsymbol{H}(\boldsymbol{w}^*)$.

Newton Method.

The Newton method has the update rule:

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;-\; \nabla_{\boldsymbol{w}} R(\boldsymbol{w}^{\text{old}}) \boldsymbol{H}^{-1}(\boldsymbol{w}^{\text{old}}) \,. \tag{5.82}$$

With previous definitions and Taylor expansion of R around $\boldsymbol{w}^*$ we have $\boldsymbol{g}(\boldsymbol{w}^*) \;=\; \boldsymbol{0}$.

The Taylor expansion of the i-th component of $\boldsymbol{g}$ around $\boldsymbol{w}^{\text{old}}$ is

$$\begin{aligned} 0 \;=\; g_i(\boldsymbol{w}^*) \;=\; g_i(\boldsymbol{w}^{\text{old}}) \;+\; \nabla_{\boldsymbol{w}} g_i(\boldsymbol{w}^{\text{old}}) \left(\boldsymbol{w}^* \;-\; \boldsymbol{w}^{\text{old}}\right) \;+ \\ \boldsymbol{\xi}^T \boldsymbol{H}_i(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi} \,, \end{aligned} \tag{5.83}$$

where $\boldsymbol{H}_i$ is the Hessian of g_i and $\boldsymbol{\xi}$ is a vector $\boldsymbol{\xi}_i \;=\; \lambda \left(\boldsymbol{w}^* \;-\; \boldsymbol{w}^{\text{old}}\right)$ with $0 \;\leq\; \lambda \;\leq\; 1$, thus $\|\boldsymbol{\xi}_i\|^2 \;\leq\; \left\|\boldsymbol{w}^* \;-\; \boldsymbol{w}^{\text{old}}\right\|$.

We obtain

$$\begin{aligned} g_i(\boldsymbol{w}^{\text{old}}) \;=\; g_i(\boldsymbol{w}^{\text{old}}) \;-\; g_i(\boldsymbol{w}^*) \;=\; g_i(\boldsymbol{w}^{\text{old}}) \;- \\ \left(g_i(\boldsymbol{w}^{\text{old}}) \;+\; \left(\boldsymbol{w}^* \;-\; \boldsymbol{w}^{\text{old}}\right)^T \nabla_{\boldsymbol{w}} g_i(\boldsymbol{w}^{\text{old}}) + \right. \\ \left. \frac{1}{2} \boldsymbol{\xi}_i^T \boldsymbol{H}_i(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_i \right) \,. \end{aligned} \tag{5.84}$$

Combining above equations the we obtain

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{w}^{\text{old}}) \;=\; -\, \boldsymbol{H}(\boldsymbol{w}^{\text{old}}) \left(\boldsymbol{w}^* \;-\; \boldsymbol{w}^{\text{old}}\right) \;- \\ \frac{1}{2} \left(\boldsymbol{\xi}_1^T \boldsymbol{H}_1(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_W^T \boldsymbol{H}_W(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_W \right)^T \,. \end{aligned} \tag{5.85}$$

which gives

$$\begin{aligned} \boldsymbol{w}^{\text{old}} \;+\; \boldsymbol{g}(\boldsymbol{w}^{\text{old}})\, \boldsymbol{H}^{-1} \;-\; \boldsymbol{w}^* \;=\; - \\ \frac{1}{2} \left(\boldsymbol{\xi}_1^T \boldsymbol{H}_1(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_W^T \boldsymbol{H}_W(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_W \right)^T \,. \end{aligned} \tag{5.86}$$

that means

$$\boldsymbol{w}^{\text{new}} \;-\; \boldsymbol{w}^* \;=\; -\, \frac{1}{2}\, \boldsymbol{H}^{-1} \left(\boldsymbol{\xi}_1^T \boldsymbol{H}_1(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_W^T \boldsymbol{H}_W(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_W \right)^T \,. \tag{5.87}$$

Because

$$\rho \;=\; \max_i \|\boldsymbol{\xi}_i\| \;\leq\; \left\|\boldsymbol{w}^{\text{old}} \;-\; \boldsymbol{w}^*\right\| \tag{5.88}$$

and

$$\frac{1}{2}\, \boldsymbol{H}^{-1} \left(\boldsymbol{\xi}_1^T \boldsymbol{H}_1(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_W^T \boldsymbol{H}_W(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_W \right)^T \;=\; O(\rho^2) \tag{5.89}$$

the Newton method is quadratic convergent in $\left\|\boldsymbol{w}^{\text{old}} \;-\; \boldsymbol{w}^*\right\|$ assumed that

$$\left\| \frac{1}{2}\, \boldsymbol{H}^{-1} \left(\boldsymbol{\xi}_1^T \boldsymbol{H}_1(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_W^T \boldsymbol{H}_W(\boldsymbol{w}^{\text{old}})\, \boldsymbol{\xi}_W \right)^T \right\| \;<\; 1 \,. \tag{5.90}$$

5.8 On-line Optimization

Until now we considered optimization techniques where a fixed set of training examples was given.

However there are cases where we do not want to have a fixed set but update our solution incrementally. That means a single training example is used for update. The first method with fixed training size is called *batch* update whereas the incremental update is called *on-line*.

In a situation where the training examples are very cheap and a huge amount of them exist we would not like to restrict ourselves to a fixed training size. This might lead to overfitting which can be avoided if the training size is large enough. On the other hand using all examples may computationally too expensive. Here on-line methods are an alternative for learning because the danger of overfitting is reduced.

In a situation where the dynamics changes, i.e. for non-stationary problems, on-line methods are useful to track the dependencies in the data. If we have a solution, then as soon as the dynamics changes, the error increases and learning starts again. Therefore on-line methods are a good choice for non-stationary problems.

The goal is to find $\boldsymbol{w}^*$ for which

$$g(\boldsymbol{w}^*) \;=\; 0\,. \tag{5.91}$$

We assume that g is an conditional expectation

$$g(\boldsymbol{w}) \;=\; \mathrm{E}\left(f(\boldsymbol{w}) \mid \boldsymbol{w}\right)\,. \tag{5.92}$$

with finite variance

$$\mathrm{E}\left((g \;-\; f)^2 \mid \boldsymbol{w}\right) \;<\; \infty\,. \tag{5.93}$$

The *Robbins-Monro procedure* is

$$\boldsymbol{w}^{i+1} \;=\; \boldsymbol{w}^i \;-\; \eta_i\, f(\boldsymbol{w}^i)\,, \tag{5.94}$$

where $f(\boldsymbol{w}^i)$ is a random variable.

The learning rate sequence η_i satisfies

$$\lim_{i\to\infty} \eta_i \;=\; 0 \tag{5.95}$$

$$\sum_{i=1}^{\infty} \eta_i \;=\; \infty \tag{5.96}$$

$$\sum_{i=1}^{\infty} \eta_i^2 \;<\; \infty\,. \tag{5.97}$$

The first conditions ensures convergence. The second condition ensures that the changes are sufficient large to find the root $\boldsymbol{w}^*$. The third condition ensures that the noise variance is limited.

The next theorem states that the Robbins-Monro procedure converges to the root $\boldsymbol{w}^*$.

Theorem 5.1 (Robbins-Monro) *Under the conditions eq. (5.95) the sequence eq. (5.94) converges to the root $\boldsymbol{w}^*$ of $\boldsymbol{g}$ with probability 1.*

If we apply the Robbins-Monro procedure to maximum likelihood we obtain

$$\frac{1}{l} \frac{\partial}{\partial \boldsymbol{w}} \sum_{i=1}^{l} \ln p(\boldsymbol{x}^i \mid \boldsymbol{w}) \;=\; 0 \,. \tag{5.98}$$

The expectation is the limit

$$\lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \frac{\partial}{\partial \boldsymbol{w}} \ln p(\boldsymbol{x}^i \mid \boldsymbol{w}) \;=\; \mathrm{E}\left(\frac{\partial}{\partial \boldsymbol{w}} \ln p(\boldsymbol{x}^i \mid \boldsymbol{w}) \right) \,. \tag{5.99}$$

The maximum likelihood solution is asymptotically equivalent to

$$\mathrm{E}\left(\frac{\partial}{\partial \boldsymbol{w}} \ln p(\boldsymbol{x}^i \mid \boldsymbol{w}) \right) \;=\; 0 \,. \tag{5.100}$$

Therefore the Robbins-Monro procedure is applicable as

$$\boldsymbol{w}^{i+1} \;=\; \boldsymbol{w}^i \;+\; \eta_i \, \frac{\partial}{\partial \boldsymbol{w}} \ln p(\boldsymbol{x}^{i+1} \mid \boldsymbol{w}) \mid_{\boldsymbol{w}^i} \;\; . \tag{5.101}$$

This is an online update formula for maximum likelihood.

Chapter 6

Neural Networks

In this chapter we introduce artificial neural networks which have a long tradition as a basic tool in bioinformatics.

In bioinformatics support vector machines are nowadays more popular than neural networks. However in international challenges like the NIPS feature selection challenge and other neural networks outperformed support vector machines but needed a lot of computational time. Some of these challenges have been won by Radford Neal and estimating the posterior by Monte Carlo sampling.

6.1 Neural Networks in Bioinformatics

Neural networks have been used in bioinformatics for splice site recognition, Protein structure and function classification, protein secondary structure prediction, and much more (see list of references at the end of this subsection).

For example we will look into the history of protein secondary structure prediction:

- **1. generation predictors.** The prediction was based on single residues, e.g. Chou-Fasman "GOR" (1957-70/80), which achieved 50-55% accuracy at prediction.
- **2. generation predictors.** Instead of single residues segments, i.e. windows, are used to predict the secondary structure of proteins. For example "GORIII" (1986-92) was able to obtain an accuracy of 55-60%. The neural networks which were used here obtained as input a window over the current position and had as target the secondary structure of the middle window position. The neural network of [Qian and Sejnowski, 1988] was based on the NETTalk architecture (see Fig. 6.1) and achieved 64.3% accuracy.
- **3. generation predictors.** The next step in protein secondary structure prediction was to use profiles or Position Specific Scoring Matrices (PSSMs). PSSMs are produced by PSI-BLAST which searches in a data base for similar sequences to the query sequence. Finally an alignment of all sequences which are very similar to the query is produced. From this alignment either a amino acid frequency or a score can be computed for each column of the alignment. Therefore the original query sequence can be replaced by a frequency sequence or a scoring sequence (PSSM). The PSSM identifies conserved regions, patterns, hydrophobic regions, etc. which can only be deduced by comparing similar sequences with each other.

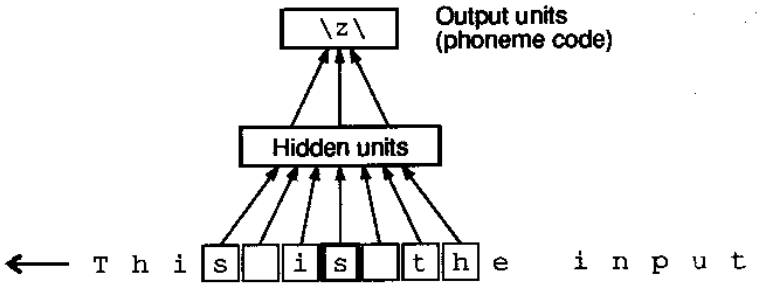


Figure 6.1: The NETTalk neural network architecture is depicted. A window scans the text and the network should predict the pronunciation of the letter in the center of the window. The same architecture was used for protein secondary structure prediction in [Qian and Sejnowski, 1988], where the input was a window of amino acids and the output the secondary structure of the center amino acid.

The neural network approach "PHD" [Rost and Sander, 1993] which uses as input a PSSM reached an accuracy of over 70%. Then other methods based on neural networks and a PSSM input were developed like PSIPRED (Jones, 1999), PROF (Quali and King 2000), JPred (Cuff et. Al, 1998) and accuracies up to 75% were obtained. In the late 90's the neural networks were replaced by support vector machines (e.g. Hua and Sun 2000) but the were not superior to neural networks.

Currently the best method is PORTER which is based on a recurent neural network architecture.

In the following are some references given which apply neural networks to protein secondary structure prediction:

- Baucom A, Cline M, Haussler D, Gregoret L M (1996): "Prediction of beta-sheet structure using neural networks", poster presented at the 10th Annual Protein Society Meeting, San Jose, California, August 3-7
- Bohr H, Bohr J, Brunak S, Cotterill R M J, Lautrup B, Narskov L, Olsen O, Petersen S (1988): "Protein secondary structure and homology by neural networks. The α-rhodopsin", FEBS Letters, 241, 223-228
- Chandonia J M, Karplus M (1995): "Neural networks for secondary structure and structural class prediction", Protein Science, 4, 275-285
- Chandonia J M, Karplus M (1996): "The importance of larger data sets for protein secondary structure prediction with neural networks", Protein Science, 5, 768-774
- Frishman D, Argos P (1996): "Incorporation of non-local interactions in protein secondary structure prediction from amino acid sequence", Protein Engineering, 9(2), 133-142
- Frishman D, Argos P (1997): "75% accuracy in protein secondary structure prediction accuracy", Proteins, 27, 329-335

- Gorse D (1995): "Prediction of protein structure from sequence using neural networks", unpublished paper
- Hansen L K, Salamon P (1990): "Neural network ensembles", IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10), 993-1001
- Holley H, Karplus M (1989): "Protein secondary structure prediction with a neural network", Proc. Nat. Acad. Sci. USA, 86, 152-156
- Kneller D G, Cohen F E, Langridge R (1990): "Improvements in protein secondary structure prediction by an enhanced neural network", Journal of Molecular Biology, 214, 171-182
- Krogh A, Riis S K (1996): "Prediction of beta sheets in proteins", in D S Touretsky, M C Mozer and M E Hasselmo (eds.), Advances in Neural Information Processing Systems 8, MIT Press
- Maclin R, Shavlik J (1993): "Using knowledge-based neural networks to improve algorithms: refining the Chou-Fasman algorithm for protein folding", Machine Learning, 11, 195-215
- Qian N, Sejnowski T J (1988): "Predicting the secondary structure of globular proteins using neural network models", Journal of Molecular Biology, 202, 865-884
- Riis S K, Krogh A (1996): "Improved prediction of protein secondary structure using structured neural networks and multiple sequence alignments", Journal of Computational Biology, 3, 163-183
- Rost B, Sander C (1993a): "Prediction of protein secondary structure at better than 70% accuracy", Journal of Molecular Biology, 232, 584-599
- Rost B, Sander C (1993b): "Secondary structure of all-helical proteins in two states", Protein Engineering, 6(8), 831-836
- Rost B, Sander C (1994): "Combining evolutionary information and neural networks to predict protein secondary structure", PROTEINS: Structure, Function and Genetics, 19, 55-72
- Stolorz P, Lapedes A, Xia Y (1992): "Predicting protein secondary structure using neural net and statistical methods", Journal of Molecular Biology, 225, 363-377
- Zhang X, Mesirov J P, Waltz D L (1992): "Hybrid system for protein secondary structure prediction", Journal of Molecular Biology, 225, 1049-1063

6.2 Principles of Neural Networks

Artificial neural networks are justified by the computational principles of the human brain which is so far the best known pattern recognition and pattern association device.

The processing units of the human brain are the neurons which are interconnected. The connections are able to transfer information from one processing unit to others. The processing units

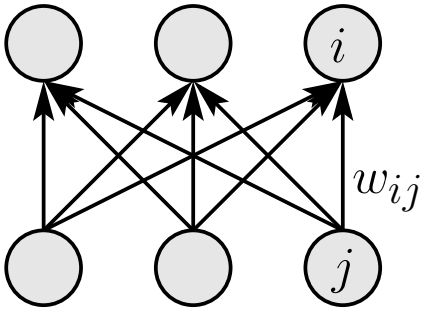


Figure 6.2: Artificial neural networks: units and weights. The weight w_{ij} gives the weight, connection strength, or synaptic weight from units j to unit i.

can combine and modulate the incoming information. The incoming information change the state of a neuron which represents the information at this neuron. The state serves to transfer the information to other neurons. The connections have different strength (synaptic weights) which give the coupling of the neurons and, therefore, the influence of one neuron onto the other.

Learning in the human brain is sought to be mainly determined by changing the synaptic weights, i.e. changing the strength of the connections between the processing units.

Artificial neural networks (ANN) treated in our context ignore that the information in the human brain is transferred through spikes (short bursts of high voltage). However the ANNs we will use can represent a rate coding, which means the information transferred is the firing rate of a neuron and the state of a neuron is its firing rate.

Neurons in artificial neural networks are represented by a variable a_i for the i-th neuron which gives the current state of the i-th neuron which is called *activation* of the neuron. Connections in ANNs are parameters w_{ij} giving the strength of the connection from unit j to unit i which are called *weights*. See Fig. 6.2 for the weight w_{ij} from unit j to unit i. These parameters are summarized in a vector (or weight vector) $\boldsymbol{w}$ which are the parameters of the neural network model.

Artificial neural networks possess special neurons. Neurons which are directly activated by the environment are called *input units*. Their activation is typically the value of the feature vector $\boldsymbol{x}$ which is assumed to represent the sensory input to the neural network. Neurons from which the result of the processing is taken are called *output units*. Typically the state of certain units is the output of the neural network. The remaining units are called *hidden units.* See Fig. 6.3 for a small architecture of a 3 layered net with only one hidden layer.

Artificial neural networks can in principle be processed in parallel, i.e. each unit can be updated independent of other units according to its current incoming signals.

The implementation of neural networks is simple as all neurons can be clones i.e. the combination of the incoming signals and the activation of the neuron can be equal for each neuron. Therefore hardware implementations of ANNs are easy to realize.

As we will see later ANNs are universal function approximators and recurrent ANNs are as powerful as Turing machines. Thus, ANNs are powerful enough to model a part of the world to solve a problem.

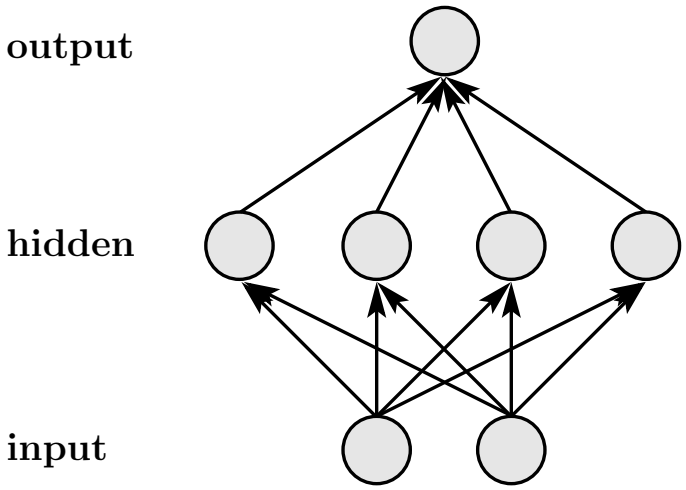


Figure 6.3: Artificial neural networks: a 3-layered net with an input, hidden, and output layer.

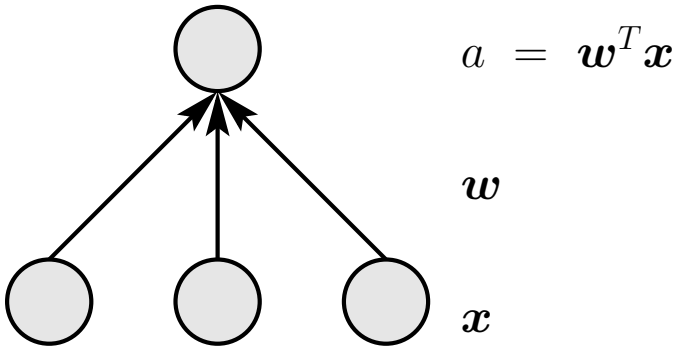


Figure 6.4: A linear network with one output unit.

6.3 Linear Neurons and the Perceptron

In this section we will focus on (artificial) neural networks (NNs) which represent linear discriminant functions.

As in Chapter 3 we assume that objects are represented or described by feature vectors $\boldsymbol{x} \in \mathbb{R}^d$. The training set consists of l objects $X = \left\{x^1, \ldots, x^l\right\}$ with a targets $y^i \in \mathbb{R}$. Again we use the matrix of feature vectors $\boldsymbol{X} = \left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\right)^T$, where $\boldsymbol{X} \in \mathbb{R}^{l \times d}$, and the vector of targets $\boldsymbol{y} = \left(y^1, \ldots, y^l\right)^T$.

The linear neural network is

$$g(\boldsymbol{x}; \boldsymbol{w}) = a = \boldsymbol{w}^T\boldsymbol{x} \,. \tag{6.1}$$

That means we have d input neurons where the i-th input neuron has activation x_i. There is only one output neuron with activation a. Connections exist from each input unit to the output unit where the connection from the i-th input unit to the output is denoted by w_i (see Fig. 6.4).

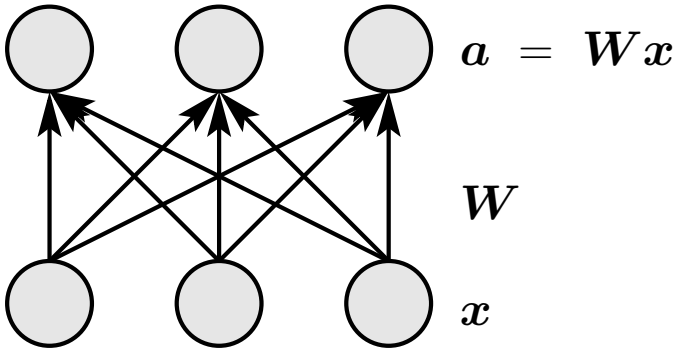


Figure 6.5: A linear network with three output units.

If we assume Gaussian noise, then the proper noise model is the least square error. In Chapter 3 in Section 3.5.1 we treated this case and found that the optimal solution is the least square estimator from eq. (3.98):

$$\hat{\boldsymbol{w}} \;=\; \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y}\,. \tag{6.2}$$

This solution may also be obtained by gradient descent on the empirical error function

$$R_{\text{emp}} \;=\; \frac{1}{2} \sum_{i=1}^{l} \left(y^i \;-\; g(\boldsymbol{x}^i; \boldsymbol{w})\right)^2 \;= \tag{6.3}$$

$$\frac{1}{2} \sum_{i=1}^{l} \left(y^i \;-\; \boldsymbol{w}^T \boldsymbol{x}^i\right)^2 \;=\; \frac{1}{2} \left(\boldsymbol{y} \;-\; \boldsymbol{X}\, \boldsymbol{w}\right)^T \left(\boldsymbol{y} \;-\; \boldsymbol{X}\, \boldsymbol{w}\right).$$

The gradient is

$$\frac{\partial}{\partial \boldsymbol{w}} R_{\text{emp}} \;=\; \sum_{i=1}^{l} \left(y^i \;-\; \boldsymbol{w}^T \boldsymbol{x}^i\right) \boldsymbol{x}^i \;=\; \left(\boldsymbol{y} \;-\; \boldsymbol{X}\, \boldsymbol{w}\right)^T \boldsymbol{X}\,. \tag{6.4}$$

The linear neuron can be extended to a linear net if the output contains more than one unit. The targets are now output vectors $\boldsymbol{Y} \;=\; \left(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^l\right)^T$, where $\boldsymbol{Y} \in \mathbb{R}^{l \times o}$.

The linear neural network is

$$\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{w}) \;=\; \boldsymbol{a} \;=\; \boldsymbol{W} \boldsymbol{x}\,, \tag{6.5}$$

where $\boldsymbol{W}$ is the weight matrix $\boldsymbol{W} \in \mathbb{R}^{o \times d}$ and o is the number of output units (see Fig. 6.5 with $o = 3$).

For convenience the weight matrix is sometimes written as a weight vector $\boldsymbol{w}$, where the columns of $\boldsymbol{W}$ are stacked on top of each other. This network is a combination of linear neurons with

$$g_i(\boldsymbol{x}; \boldsymbol{w}) \;=\; a_i \;=\; \boldsymbol{w}_i^T \boldsymbol{x}\,, \tag{6.6}$$

where $\boldsymbol{w}_i$ is the i-th line of $\boldsymbol{W}$ written as column vector.

Because the optimal solution is

$$\hat{\boldsymbol{w}_i} \;=\; \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}^i \;, \tag{6.7}$$

where $\boldsymbol{y}^i$ is the i-th column of $\boldsymbol{Y}$, we obtain as solution for the whole network

$$\hat{\boldsymbol{W}^T} \;=\; \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y} \;. \tag{6.8}$$

A linear neuron with a threshold function is called *perceptron*. The output of the perceptron is binary.

The perceptron model is

$$a \;=\; \text{sign}\left(\boldsymbol{w}^T\boldsymbol{x}\right) \;, \tag{6.9}$$

where sign is called *threshold function*. Note that the perceptron is a linear classifier from Section 4.2 without an offset b. Note, that without the offset b the learning problem is not translation invariant.

For the perceptron we assume a classification task, where $y \in \{-1, 1\}$.

The error function for the perceptron is defined as

$$R_{\text{emp}} \;=\; -\sum_{i=1;\; a^i y^i = -1}^{l} y^i\; \boldsymbol{w}^T\boldsymbol{x}^i \;. \tag{6.10}$$

The error is the margin error of the support vector machine e.g. in Section 4.6 in the eq. (4.54) with $b = 0$ and very small ρ. The C support vector error is obtained if $\boldsymbol{w}$ is scaled such that the minimal $|\boldsymbol{w}^T\boldsymbol{x}^i|$ of above misclassifications is one. Scaling does not influence the perceptron output.

Using a gradient update we obtain

$$\Delta\boldsymbol{w} \;=\; \eta \sum_{i=1;\; a^i y^i = -1}^{l} y^i\; \boldsymbol{x}^i \tag{6.11}$$

or in an on-line formulation

$$\text{if } a\; y \;=\; -1: \;\; \Delta\boldsymbol{w} \;=\; \eta\; y\; \boldsymbol{x} \;. \tag{6.12}$$

The update rule is very simple: if a pattern is wrongly classified then the label multiplied by the input vector is added to the weight vector. Fig. 6.6 depicts the perceptron update rule.

Assume that the problem is linear separable. The perceptron learning rule converges after finite many steps to a a solution which classifies all training examples correct.

However an arbitrary solution out of all possible solutions is chosen. If the problem in not linearly separable then the algorithms does not stop.

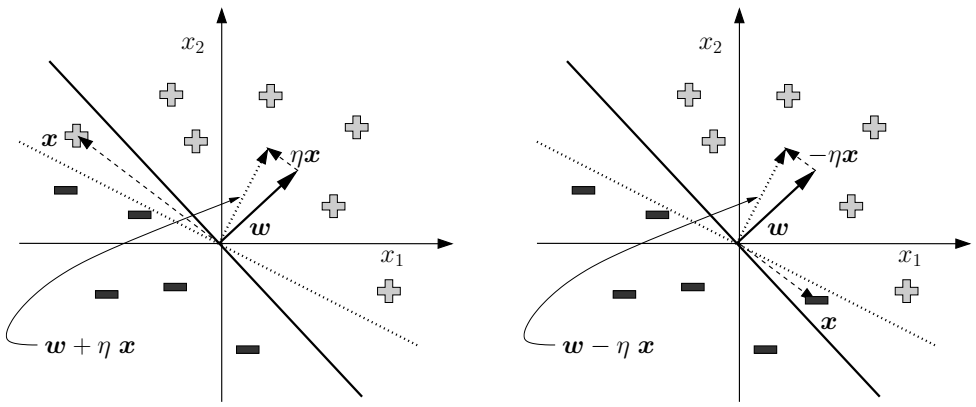


Figure 6.6: The perceptron learning rule. Left: a positive example is wrongly classified and correctly classified after the weight update. Right: a negative example is wrongly classified and correctly processed after weight update.

Minsky and Papert showed 1969 that many problems cannot be solved by the perceptron. For example the XOR problem (see Fig. 3.18) is a nonlinear problem which cannot be solved by the perceptron.

Because of the work of Minsky and Papert neural networks were not popular until the mid 80ies where the multi layer perceptron with nonlinear units and back-propagation was reintroduced.

6.4 Multi-Layer Perceptron

With the work Rumelhart et al. [1986b,a] neural networks and especially the multi-layer perceptron trained with back-propagation became very popular.

6.4.1 Architecture and Activation Functions

A multi-layer perceptron (MLP) consists of more than one perceptron where the output of one perceptron is the input of the next perceptron. Further, the activation (state) of a neuron is computed through a nonlinear function.

We define for the multi-layer perceptron (MLP, see Fig. 6.7):

- a_i: *activity* of the i-th unit
- $a_0 = 1$: activity of 1 of the bias unit
- w_{ij}: *weight* from unit j to unit i
- $w_{i0} = b_i$: *bias weight* of unit i
- W: number of weights
- N: number of units

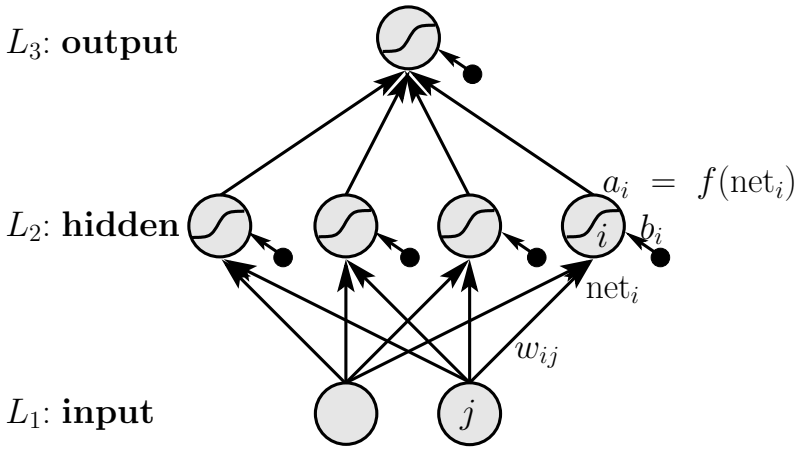


Figure 6.7: Figure of an MLP.

- I: number of inputs units ($1 \leq i \leq I$) located in the first layer called *input layer*.
- O: number of output units ($N - O + 1 \leq i \leq N$) located in the last layer called *output layer*.
- H: number of hidden units ($I < i \leq N - O$) located in the hidden layers.
- L: number of layers, where L_ν is the index set of the ν-th layer; $L_1 = \{1, \ldots, I\}$ and $L_L = \{N - O + 1, \ldots, N\}$.
- net_i: *network input* to the i-th unit ($I < i$) computed as

$$\mathrm{net}_i = \sum_{j=0}^{N} w_{ij}\, a_j \tag{6.13}$$

- f: *activation function* with

$$a_i = f(\mathrm{net}_i) \tag{6.14}$$

  It is possible to define different activation functions f_i for different units. The activation function is sometimes called *transfer function* (in more realistic networks one can distinguish between activation of a neuron and the signal which is transferred to other neurons).

- the *architecture* of a neural network is given through number of layers, units in the layers, and defined connections between units – the activations function may be accounted to the architecture.

A feed-forward MLP has only connections from units in lower layers to units in higher layers:

$$i \in L_\nu \text{ and } j \in L_{\nu'} \text{ and } \nu \leq \nu' \Rightarrow \boldsymbol{w}_{ij} = 0\,. \tag{6.15}$$

For a conventional multi-layer perceptron there are only connections or weights between consecutive layers. Other weights are fixed to zero. The network input is then for i in layer $1 < \nu$ is

$$\forall_{i \in L_\nu} : \ \mathrm{net}_i \ = \ \sum_{j; j \in L_{\nu-1}}^{N} w_{ij} \ a_j \ . \tag{6.16}$$

Connections between units in layers which are not adjacent are called *shortcut connections*.

The forward pass of a neural network is given in Alg. 6.1.

Algorithm 6.1 Forward Pass of an MLP

BEGIN initialization
provide input $\boldsymbol{x}$
for all $(i = 1 \ ; \ i \ \leq \ I \ ; \ i++)$ **do**

$$a_i \ = \ x_i$$

end for
END initialization
BEGIN Forward Pass
for $(\nu = 2 \ ; \ \nu \ \leq \ L \ ; \ \nu++)$ **do**
for all $i \in L_\nu$ **do**

$$\mathrm{net}_i \ = \ \sum_{j=0; w_{ij} \text{ exists}}^{N} w_{ij} \ a_j$$

$$a_i \ = \ f(\mathrm{net}_i)$$

end for
end for
provide output $g_i(\boldsymbol{x}; \boldsymbol{w}) \ = \ a_i, N - O + 1 \ \leq \ i \ \leq \ N$
END Forward Pass

Activation Functions.

The commonly used activation functions are sigmoid functions.

The logistic function is

$$f(a) \ = \ \frac{1}{1 \ + \ \exp(-\, a)} \tag{6.17}$$

and the $\tanh$ activation function is

$$f(a) \ = \ \tanh(a) \ = \ \frac{\exp(a) \ - \ \exp(-\, a)}{\exp(a) \ + \ \exp(-\, a)} \ . \tag{6.18}$$

Both functions are equivalent because

$$\frac{1}{2} \left(\tanh(a/2) \ + \ 1 \right) \ = \ \frac{1}{1 \ + \ \exp(-\, a)} \ . \tag{6.19}$$

(proof as exercise)

That means through weight scaling and through adjusting the bias weights the networks with logistic or $\tanh$ can be transformed into each other.

Also quadratic or other activation functions can be used but the fact that the sigmoid functions are squashed into a interval makes learning robust. Because the activation is bounded the derivatives are bounded as we see later. Networks with sigmoid activation are even more robust against unknown input which can drive some activations in regions which are not explored in the training phase.

Higher order units.

Higher order units use not a linear net_i. For example second order units have the form

$$\mathrm{net}_i \;=\; \sum_{(j_1,j_2)=(0,0)}^{N} w_{ij_1j_2}\; a_{j_1}\; a_{j_2}\;. \tag{6.20}$$

Note that linear and constant terms are considered because of the bias unit $a_0 \;=\; 1$.

We will use higher order units in Section 6.6.6 in order to gate information flow and to access certain information.

Symmetric Networks.

Symmetric networks can be produced with the $\tanh$ function if the signs of input weights and output weights are changed because $\tanh(-x) \;=\; -\;\tanh(x)$, therefore, $w_2 \tanh(w_1\; x) \;=\; (-w_2)\; \tanh((-w_1)\; x)$.

Permutations of the hidden units in one layer leads to equivalent networks.

That means the same function can be represented through different network parameters.

6.4.2 Universality

The in [Funahashi, 1989] it is proven that MLPs are universal function approximators:

Theorem 6.1 (MLPs are Universal Approximators) *If the activation function f is not constant, monotonic increasing, continuous, and bounded, then each continuous function $g(x)$ on a compact interval K can be approximated arbitrary exact through a three-layered neural network $o(x; \boldsymbol{w})$:*

$$\max_{x\in K} |o(x; \boldsymbol{w}) \;-\; g(x)| \;<\; \epsilon\,, \tag{6.21}$$

where ϵ is given and the number of hidden units H depend on ϵ and g.

This statement (neural networks are universal function approximators) was also shown in [Hornik et al., 1989, Stinchcombe and White, 1989, White, 1990].

We cite from [Hornik et al., 1989]:

> In other words, standard feed-forward networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well in the ρ_μ metric, regardless of the squashing function Ψ (continuous or not), regardless of the dimension of the input space r, and regardless of the input space environment μ. Thus, Σ networks are also universal approximators.

and

> In other words, there is a single hidden layer feed-forward network that approximates any measurable function to any desired degree of accuracy on some compact set K of input patterns that to the same degree of accuracy has measure (probability of occurrence) 1.

Note that in [Hornik et al., 1989] in contrast to [Funahashi, 1989] non-continuous activation functions are allowed.

6.4.3 Learning and Back-Propagation

To train neural network the gradient based methods from Chapter 5 can be applied.

The gradient of a neural network can be computed very efficiently by back-propagation [Rumelhart et al., 1986b,a] which was even earlier proposed by [Werbos, 1974]. The back-propagation algorithm made artificial neural networks popular since the mid 80ies.

In the following we derive the back-propagation algorithm in order to compute the gradient of a neural network.

In eq. (3.161) we defined the empirical error as

$$R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;=\; \frac{1}{l} \sum_{i=1}^{l} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \;, \tag{6.22}$$

where we generalized to multiple outputs. According to eq. (5.3) the gradient descent update is

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;-\; \eta \, \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;. \tag{6.23}$$

We obtain

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;=\; \frac{1}{l} \sum_{i=1}^{l} \nabla_{\boldsymbol{w}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \;. \tag{6.24}$$

We have to compute

$$\frac{\partial}{\partial w_{kl}} \, L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \tag{6.25}$$

for all w_{kl}.

$$\begin{aligned} &\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \\ &\frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; \frac{\partial \mathrm{net}_k}{\partial w_{kl}} \; = \\ &\frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; a_l \; . \end{aligned} \tag{6.26}$$

We define the δ-error at unit k as

$$\delta_k \; = \; \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \tag{6.27}$$

and obtain

$$\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \; \delta_k \; a_l \; . \tag{6.28}$$

The δ-error at the output units is

$$\begin{aligned} &\forall_{1 \leq s \leq O} : \quad \delta_{N-O+s} \; = \\ &\frac{\partial}{\partial g_s} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; f'(\mathrm{net}_{N-O+s}) \; = \\ &\frac{\partial L}{\partial a_{N-O+s}} f'(\mathrm{net}_{N-O+s}) \; , \end{aligned} \tag{6.29}$$

where f' denotes the derivative of the activation function f.

The δ-error at units not in the output layer is

$$\begin{aligned} &\delta_k \; = \; \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \\ &\sum_n \frac{\partial}{\partial \mathrm{net}_n} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; \frac{\partial \mathrm{net}_n}{\partial \mathrm{net}_k} \; = \\ &\sum_n \frac{\partial}{\partial \mathrm{net}_n} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; \frac{\partial \mathrm{net}_n}{\partial \mathrm{net}_k} \; = \\ &f'(\mathrm{net}_k) \; \sum_n \delta_n \; w_{nk} \; , \end{aligned} \tag{6.30}$$

where the $\sum_n$ goes over all n for which w_{nk} exists. Typically n goes over all units in the layer above the layer where unit k is located.

This efficient computation of the δ-errors led to the term "back-propagation" because the δ-errors of one layer are used to compute the δ-errors of the layer below. The algorithm is sometimes also called "δ-propagation". Fig. 6.8 depicts the back-propagation algorithm for a 4-layer feed-forward network.

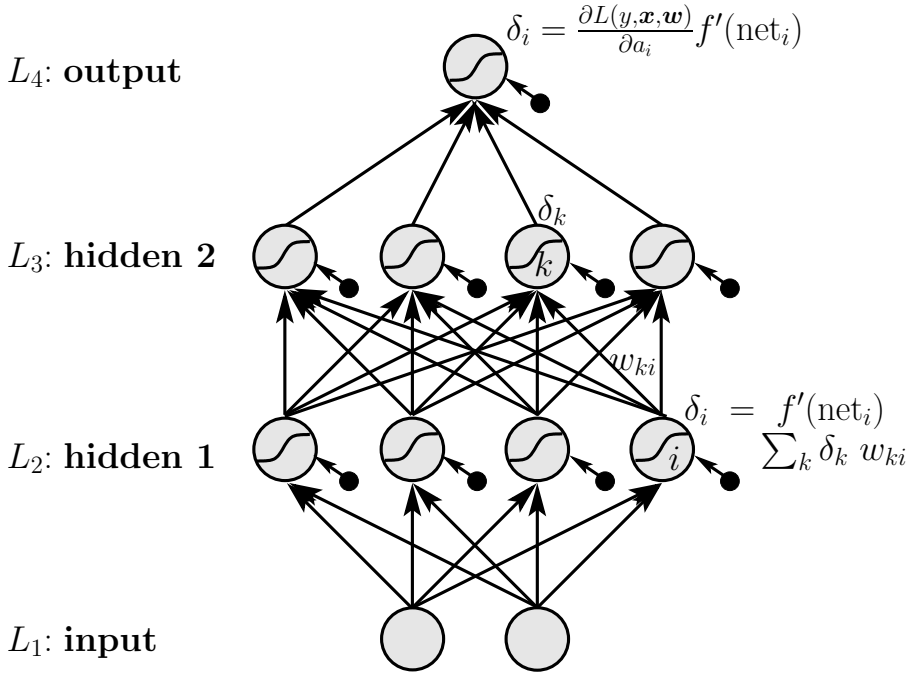


Figure 6.8: 4-layer MLP where the back-propagation algorithm is depicted. The $\delta_k = \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right)$ are computed from the δ's of the higher layer. The back-propagation algorithm starts form the top layer and ends with the layer 2 where the weights to layer 2 are updated.

Note that for the logistic function

$$f(a) \;=\; \frac{1}{1 \;+\; \exp(-\,a)} \tag{6.31}$$

the derivative is

$$f'(a) \;=\; f(a)\;(1 \;-\; f(a))\;. \tag{6.32}$$

This speeds up the algorithm because the exponential function must only be evaluated once.

Alg. 6.2 gives the backward pass for a single example where the weight update is accumulated in Δw_{ij}.

Algorithm 6.2 Backward Pass of an MLP

BEGIN initialization
provide activations a_i of the forward pass and the label y
for $(i = N - O + 1\;;\; i \;\leq\; N\;;\; i++)$ **do**

$$\delta_i \;=\; \frac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f'(\mathrm{net}_i)$$

for all $j \in L_{L-1}$ **do**

$$\Delta w_{ij} \;=\; -\,\eta\;\delta_i\;a_j$$

end for
end for
END initialization
BEGIN Backward Pass
for $(\nu = L - 1\;;\nu \;\geq\; 2\;;\; \nu --)$ **do**
for all $i \in L_\nu$ **do**

$$\delta_i \;=\; f'(\mathrm{net}_i) \sum_k \delta_k\; w_{ki}$$

for all $j \in L_{\nu-1}$ **do**

$$\Delta w_{ij} \;=\; -\,\eta\;\delta_i\;a_j$$

end for
end for
end for
END Backward Pass

Note that in Alg. 6.2 has complexity of $O(W)$ which is very efficient.

6.4.4 Hessian

The Hessian matrix is useful for training and regularizing neural networks. The Hessian is defined as

$$H_{lj} \;=\; \frac{\partial^2 R(\boldsymbol{w})}{\partial w_l\;\partial w_j}\;. \tag{6.33}$$

For most risk functions it is sufficient to calculate the Hessian of the loss function for one example to compute the Hessian the risk

$$H_{ij}(y^i, \boldsymbol{x}^i, \boldsymbol{w}) \;=\; \frac{\partial^2 L(y^i, \boldsymbol{x}^i, \boldsymbol{w})}{\partial w_l \, \partial w_j} \; . \tag{6.34}$$

and then combine these derivatives. In the following we write H_{ij} for the Hessian of the loss function for a certain example.

We have already seen that optimization techniques like the Newton method (see Section 5.4.1) uses the Hessian or uses characteristics of the Hessian to optimize the step-size (see Section 5.3).

Further the Hessian can be used to retrain a network or to identify units or weights which are not needed in the architecture (see Section 6.4.5.3). In other applications the Hessian is used to assign error bars to the outputs (see Section 7.4) or to determine regularization parameters (see Section 7.5) or to compare or average over models (see Section 7.8).

In this section we especially consider techniques which are adapted to neural networks to efficiently compute the Hessian or to approximate it.

Diagonal approximations.

According to eq. 6.26 we have

$$\frac{\partial^2 L(\boldsymbol{w})}{\partial w_{ij}^2} \;=\; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_i^2} \; a_j^2 \; . \tag{6.35}$$

Further the chain rule gives

$$\begin{aligned} &\frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_i^2} \;= \\ &\big(f'(\mathrm{net}_i)\big)^2 \sum_l \sum_k \; w_{lj} \; w_{kj} \; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_l \; \partial \mathrm{net}_k} \;+\; f''(\mathrm{net}_i) \sum_l \frac{\partial L\,(\boldsymbol{w})}{\partial \mathrm{net}_l} \;\approx \\ &\big(f'(\mathrm{net}_i)\big)^2 \sum_l \; w_{lj}^2 \; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_l^2} \;+\; f''(\mathrm{net}_i) \sum_l \frac{\partial L\,(\boldsymbol{w})}{\partial \mathrm{net}_l} \; , \end{aligned} \tag{6.36}$$

where the off-diagonal elements of the Hessian were neglected.

This approximation can be done with an additional forward and backward pass and, therefore, is as efficient as back-propagation with complexity $O(W)$.

However the approximation is often rough because the off-diagonal elements have large absolute value.

Outer product / Levenberg-Marquardt Approximation.

The Levenberg-Marquardt algorithm in Section 5.5 approximated the Hessian in eq. (5.66)

through

$$H_{ij,kl} = \frac{\partial^2 R}{\partial w_{ij}\,\partial w_{kl}} = \sum_{i=1}^{l}\left(\frac{\partial e^i}{\partial w_{ij}}\,\frac{\partial e^i}{\partial w_{kl}} + e^i\,\frac{\partial^2 e^i}{\partial w_{ij}\,\partial w_{kl}}\right) \approx \sum_{i=1}^{l}\frac{\partial a_N(\boldsymbol{x}^i)}{\partial w_{ij}}\,\frac{\partial a_N(\boldsymbol{x}^i)}{\partial w_{kl}}\,, \tag{6.37}$$

where

$$e^i = (a_N(\boldsymbol{x}^i;\boldsymbol{w}) - y^i) \tag{6.38}$$

and

$$R(\boldsymbol{w}) = \sum_{i=1}^{l}\left(e^i(\boldsymbol{w})\right)^2\,. \tag{6.39}$$

Attention: that only holds for squared error!

Inverse Hessian Approximation.

We will need the inverse Hessian in Section 6.4.5.3 which can be approximated by an outer product approximation.

If $\boldsymbol{g}^i$ is the weight gradient for example $\boldsymbol{x}^i$ then

$$\boldsymbol{H} = \sum_{i=1}^{l}\boldsymbol{g}^i\,\left(\boldsymbol{g}^i\right)^T\,. \tag{6.40}$$

This means the Hessian can be build sequentially

$$\boldsymbol{H}^k = \sum_{i=1}^{k}\boldsymbol{g}^i\,\left(\boldsymbol{g}^i\right)^T\,, \tag{6.41}$$

where $\boldsymbol{H}^l = \boldsymbol{H}$.

We obtain

$$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k + \boldsymbol{g}^{k+1}\left(\boldsymbol{g}^{k+1}\right)^T\,. \tag{6.42}$$

To this formula the matrix inversion lemma

$$(\boldsymbol{A} + \boldsymbol{B}\,\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{I} + \boldsymbol{C}\,\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\,\boldsymbol{A}^{-1} \tag{6.43}$$

is applied which gives

$$\left(\boldsymbol{H}^{k+1}\right)^{-1} \;=\; \left(\boldsymbol{H}^{k}\right)^{-1} \;-\; \frac{\left(\boldsymbol{H}^{k}\right)^{-1} \boldsymbol{g}^{k+1} \left(\boldsymbol{g}^{k+1}\right)^{T} \left(\boldsymbol{H}^{k}\right)^{-1}}{1 \;+\; \left(\boldsymbol{g}^{k+1}\right)^{T} \left(\boldsymbol{H}^{k}\right)^{-1} \boldsymbol{g}^{k+1}} \;. \tag{6.44}$$

Because only $\left(\boldsymbol{H}^{k}\right)^{-1} \boldsymbol{g}^{k+1}$ must be computed. and then each component of the Hessian can be updated, the algorithm has $O(W^2)$ complexity per iteration.

If the initial matrix is $\alpha\, \boldsymbol{I}$ then this algorithm gives the inverse of $(\boldsymbol{H} \;+\; \alpha\, \boldsymbol{I})$. In many algorithms $(\boldsymbol{H} \;+\; \alpha\, \boldsymbol{I})$ is more robust because the inversion with small eigenvalues is avoided.

If the Hessian is build up iteratively over more epochs (whole training set is updated) then also quasi-Newton methods (see Section 5.4.1) may be considered because they also build up an approximation of the Hessian which improves step by step.

Finite differences.

Either finite differences can be applied to the error function or the gradient of the error function.

Because for finite differences of the error function we have to disturb each pair of weights ($R(w_{ij} \pm \epsilon, w_{kl} \pm \epsilon)$ and perform for each disturbance a forward pass we obtain complexity of $O(W^3)$.

The finite differences applied to the gradient gives

$$H_{ij,kl} \;\approx\; \frac{1}{2\,\epsilon} \left(\frac{\partial R}{\partial w_{ij}}(w_{kl} \;+\; \epsilon) \;-\; \frac{\partial R}{\partial w_{ij}}(w_{kl} \;-\; \epsilon) \right) \;. \tag{6.45}$$

This is more efficient than finite differences of the error function because only W weights have to be disturbed and the gradient can be computed in $O(W)$ time by back-propagation. Therefore the Hessian can be approximated in $O(W^2)$ complexity.

Exact Computation of the Hessian.

So far only the diagonal approximation was as efficient as back-propagation with $O(W)$ and the other approximations of the Hessian have complexity $O(W^2)$.

However the Hessian can be computed *exactly* with complexity $O(W^2)$. Ideas from the back-propagation are used for computing the Hessian.

From eq. (6.27) and eq. (6.28) we have

$$\begin{aligned} \delta_k \;&=\; \frac{\partial}{\partial \mathrm{net}_k}\, L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \\ \frac{\partial}{\partial w_{kl}}\, L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \;&=\; \delta_k\, a_l \;. \end{aligned} \tag{6.46}$$

$$
\begin{aligned}
H_{ij,kl} \;&=\; \frac{\partial^2 L}{\partial w_{ij}\,\partial w_{kl}} \;= \\
a_j\,\frac{\partial}{\partial \mathrm{net}_i}\frac{\partial L}{\partial w_{kl}} \;&=\; a_j\,\frac{\partial}{\partial \mathrm{net}_i}\left(\delta_k\,a_l\right) \;= \\
\delta_k\,a_j\,f'(\mathrm{net}_l)\,\frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_i} \;&+\; a_j\,a_l\,\frac{\partial \delta_k}{\partial \mathrm{net}_i} \;= \\
\delta_k\,a_j\,f'(\mathrm{net}_l)\,V_{li} \;&+\; a_j\,a_l\,b_{ki}\;,
\end{aligned}
\tag{6.47}
$$

where we used the definitions

$$
\begin{aligned}
V_{li} \;&=\; \frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_i} \\
b_{ki} \;&=\; \frac{\partial \delta_k}{\partial \mathrm{net}_i}\;.
\end{aligned}
\tag{6.48}
$$

The V_{li} are computed by forward propagation through

$$
V_{li} \;=\; \sum_{n:w_{ln}} \frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_n}\frac{\partial \mathrm{net}_n}{\partial \mathrm{net}_i} \;=\; \sum_{n:w_{ln}} f'(\mathrm{net}_l)\,w_{ln}\,V_{ni}\;. \tag{6.49}
$$

Note, that for input units the V_{li} are not required, so that the forward propagation can be initialized by

$$
V_{ii} \;=\; 1 \tag{6.50}
$$

and for i in a higher than or the same layer as l set

$$
V_{li} \;=\; 0\;. \tag{6.51}
$$

Now for all l in a higher layer than i the value V_{li} can be determined by the forward propagation.

A backward propagation can now determine the values b_{li} remember the back-propagation eq. (6.30):

$$
\delta_k \;=\; f'(\mathrm{net}_k)\,\sum_n \delta_n\,w_{nk} \tag{6.52}
$$

which leads to

$$
\begin{aligned}
b_{ki} \;&=\; \frac{\partial}{\partial \mathrm{net}_i} f'(\mathrm{net}_k)\,\sum_n \delta_n\,w_{nk} \;= \\
f''(\mathrm{net}_k)\,V_{ki}\,\sum_n \delta_n\,w_{nk} \;&+\; f'(\mathrm{net}_k)\,\sum_n w_{nk}\,b_{ni}\;.
\end{aligned}
\tag{6.53}
$$

Note that $w_{ij} = w_{nk}$ must be avoided in above equations because we expanded $\frac{\partial}{\partial w_{ij}}$ into $a_j \frac{\partial}{\partial \mathrm{net}_i}$. However it is always possible to use the symmetric Hessian entry because it avoids this equality otherwise the network would contain cycles.

The initial values b_{ki} are given as

$$b_{ki} = \sum_n S_{kn} V_{ni} = [\boldsymbol{S}\boldsymbol{V}_i]_k = [\boldsymbol{S}\boldsymbol{V}]_{ki} , \tag{6.54}$$

where

$$S_{kj} = \frac{\partial^2 L}{\partial \mathrm{net}_k \, \partial \mathrm{net}_j} . \tag{6.55}$$

The initial conditions for the backward pass are for output unit i

$$\delta_i = \frac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f'(\mathrm{net}_i) \tag{6.56}$$

and for i and j output units (where output units are not interconnected):

$$S_{ij} = \frac{\partial^2 L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i \, \partial a_j} f'(\mathrm{net}_i) \, f'(\mathrm{net}_j) . \tag{6.57}$$

The algorithm Alg. 6.3 has complexity $O(W^2)$ which is optimal because the Hessian has $O(W^2)$ entries (it is not W^2 because of symmetries).

Multiplication of the Hessian with a Vector.

Similar to previous algorithm the product of the Hessian with a vector can be computed very efficiently. It can be computed in $O(W)$ time.

Following Pearlmutter [Pearlmutter, 1994] (see similar approach of Møller [Mller, 1993]) we define a differential operator as

$$\mathcal{R}\{.\} = \boldsymbol{v}^T \nabla_{\boldsymbol{w}} \tag{6.58}$$

in order to derive a efficient way to compute

$$\boldsymbol{v}^T \boldsymbol{H} = \mathcal{R}\{\nabla_{\boldsymbol{w}} L\} = \boldsymbol{v}^T \nabla^2_{\boldsymbol{w}} L . \tag{6.59}$$

We have

$$\mathcal{R}\{\boldsymbol{w}\} = \boldsymbol{v} \tag{6.60}$$

and

$$\mathcal{R}\{a_i\} = \boldsymbol{0} \tag{6.61}$$

Algorithm 6.3 Hessian Computation

forward pass according to Alg. 6.1
BEGIN Hessian Forward Pass
for $(\nu = 2\ ;\ \nu\ \leq\ L\ ;\ \nu++)$ **do**
for all $i \in L_\nu$ **do**

$$V_{ii}\ =\ 1$$

for $(\nu_1 = \nu\ ;\ \nu_1\ \geq\ 2\ ;\ \nu_1--)$ **do**
for all $l \in L_{\nu_1}$ **do**

$$V_{li}\ =\ 0$$

end for
end for
for $(\nu_1 = \nu + 1\ ;\ \nu_1\ \leq\ L\ ;\ \nu_1++)$ **do**
for all $l \in L_{\nu_1}$ **do**

$$V_{li}\ =\ \sum_n f'(\mathrm{net}_l)\ w_{ln}\ V_{ni}$$

end for
end for
end for
end for
END Hessian Forward Pass
backward pass according to Alg. 6.2
BEGIN Hessian Backward Pass
for $(i = N - O + 1\ ;\ i\ \leq\ N\ ;\ i++)$ **do**
for $(j = N - O + 1\ ;\ j\ \leq\ N\ ;\ j++)$ **do**

$$S_{ij}\ =\ \frac{\partial^2 L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i\ \partial a_j} f'(\mathrm{net}_i)\ f'(\mathrm{net}_j)$$

end for
end for
for $(\nu = L\ ;\ \nu\ \geq\ 2\ ;\ \nu--)$ **do**
for all $l \in L_\nu$ **do**
for $(i = N - O + 1\ ;\ i\ \leq\ N\ ;\ i++)$ **do**

$$b_{il}\ =\ \sum_{n=N-O+1}^{N} S_{in}\ V_{nl}$$

end for
for $(\nu = L - 1\ ;\nu\ \geq\ 2\ ;\ \nu--)$ **do**
for all $i \in L_\nu$ **do**

$$b_{il}\ =\ f''(\mathrm{net}_i)\ V_{il}\ \sum_n \delta_n\ w_{ni}\ +\ f'(\mathrm{net}_i)\ \sum_n w_{ni}\ b_{nl}$$

end for
end for
end for
end for
END Hessian Backward Pass
for all $(i, j)\ \geq\ (k, l)$ **do**

$$H_{ij,kl}\ =\ \delta_k\ a_j\ f'(\mathrm{net}_l)\ V_{li}\ +\ \ a_j\ a_l\ b_{li}$$

end for

for a_i input unit.

We obtain

$$\mathcal{R}\{a_i\} \;=\; f'(\mathrm{net}_i)\,\mathcal{R}\{\mathrm{net}_i\} \tag{6.62}$$

and

$$\mathcal{R}\{\mathrm{net}_i\} \;=\; \sum_j w_{ij}\mathcal{R}\{a_j\} \;+\; \sum_j v_{ij}\,a_j\;, \tag{6.63}$$

where the first sum vanishes for j in the input layer.

Now the operator is applied to the δ's of the back-propagation algorithm.

For an output unit i we have

$$\delta_i \;=\; \frac{\partial L(y,\boldsymbol{x},\boldsymbol{w})}{\partial a_i} f'(\mathrm{net}_i) \tag{6.64}$$

which gives

$$\begin{aligned}&\mathcal{R}\{\delta_i\} \;= \\ &\left(\frac{\partial^2 L(y,\boldsymbol{x},\boldsymbol{w})}{\partial a_i^2}\left(f'(\mathrm{net}_i)\right)^2 \;+\; \frac{\partial L(y,\boldsymbol{x},\boldsymbol{w})}{\partial a_i} f''(\mathrm{net}_i)\right)\;\mathcal{R}\{\mathrm{net}_i\}\end{aligned} \tag{6.65}$$

For non-output units i we have

$$\begin{aligned}&\mathcal{R}\{\delta_i\} \;=\; f''(\mathrm{net}_i)\mathcal{R}\{\mathrm{net}_i\}\sum_k w_{ki}\,\delta_k \;+ \\ &f'(\mathrm{net}_i)\sum_k v_{ki}\,\delta_k \;+\; \;+ f'(\mathrm{net}_i)\sum_k w_{ki}\,\mathcal{R}\{\delta_k\}\end{aligned} \tag{6.66}$$

The Hessian can be computed as

$$\mathcal{R}\left\{\frac{\partial L}{\partial w_{ij}}\right\} \;=\; \mathcal{R}\{\delta_i\,a_j\} \;=\; \mathcal{R}\{\delta_i\}\,a_j \;+\; \mathcal{R}\{a_j\}\delta_i\;, \tag{6.67}$$

where again for the input units j the second term vanishes.

The algorithm for computing the product $\boldsymbol{v}^T\boldsymbol{H}$ of a vector with the Hessian has complexity $O(W)$.

This algorithm applied to the W unit vectors allows to extract the Hessian in $O(W^2)$ time but Alg. 6.3 is more efficient.

Algorithm 6.4 Hessian-Vector Multiplication

forward pass according to Alg. 6.1
BEGIN Hessian-Vector Forward Pass
for all $i \in L_1$ **do**

$$\mathcal{R}\{a_i\} \;=\; \mathbf{0}$$

end for
for $(\nu = 2\,;\; \nu \;\leq\; L\,;\; \nu++)$ **do**
for all $i \in L_\nu$ **do**

$$\mathcal{R}\{\mathrm{net}_i\} \;=\; \sum_j w_{ij}\mathcal{R}\{a_j\} \;+\; \sum_j v_{ij}\, a_j$$
$$\mathcal{R}\{a_i\} \;=\; f'(\mathrm{net}_i)\mathcal{R}\{\mathrm{net}_i\}$$

end for
end for
END Hessian-Vector Forward Pass
backward pass according to Alg. 6.2
BEGIN Hessian-Vector Backward Pass
for $(i = N - O + 1\,;\; i \;\leq\; N\,;\; i++)$ **do**

$$\mathcal{R}\{\delta_i\} \;=\; \left(\frac{\partial^2 L(y,\boldsymbol{x},\boldsymbol{w})}{\partial a_i^2}\left(f'(\mathrm{net}_i)\right)^2 \;+\; \frac{\partial L(y,\boldsymbol{x},\boldsymbol{w})}{\partial a_i} f''(\mathrm{net}_i)\right)\; \mathcal{R}\{\mathrm{net}_i\}$$

end for
for $(\nu = L - 1\,;\; \nu \;\geq\; 2\,;\; \nu--)$ **do**
for all $i \in L_\nu$ **do**

$$\mathcal{R}\{\delta_i\} \;=\; f''(\mathrm{net}_i)\mathcal{R}\{\mathrm{net}_i\}\sum_k w_{ki}\;\delta_k \;+$$
$$f'(\mathrm{net}_i)\sum_k v_{ki}\;\delta_k \;+\; \;+\; f'(\mathrm{net}_i)\sum_k w_{ki}\;\mathcal{R}\{\delta_k\}$$

end for
end for
END Hessian-Vector Backward Pass
for all (i, j) **do**

$$\left[\boldsymbol{v}^T\boldsymbol{H}\right]_{ij} \;=\; \mathcal{R}\{\frac{\partial L}{\partial w_{ij}}\} \;=\; \mathcal{R}\{\delta_i\}\, a_j \;+\; \mathcal{R}\{a_j\}\delta_i$$

end for

6.4.5 Regularization

In Section 3.6.4 we introduced the idea of structural risk minimization, where the empirical risk (the training error) is minimized for a function class with bounded complexity.

In Section 3.6.5 the margin was define as a complexity measure and in Chapter 4 the support vector machines minimize the complexity given as the margin for a perfect classification of the training data. In Section 4.4 perfect classification was replaced by margin errors and a new variable C was introduced. The variable C controlled the trade-off between margin errors and large margin. Therefore in Section 4.4 we already found the trade-off between low complexity and low training error.

In Fig. 3.20 the trade-off between complexity and training error (empirical risk) is depicted:

$$R \leq R_{\text{emp}} + \text{complexity} . \tag{6.68}$$

Increasing complexity leads to smaller training error but the test error, the risk, increases at some point.

Also Fig. 3.15 shows the relation between the test error $R(g)$ and the training error as a function of the complexity. The test error R first decreases and then increases with increasing complexity. The training error decreases with increasing complexity. The test error R is the sum of training error and a complexity term. At some complexity point the training error decreases slower than the complexity term increases – this is the point of the optimal test error.

For low test error, i.e. high generalization, we have to control the complexity of the neural network.

Typical regularization terms are

- smoothing terms, which control the curvature and higher order derivatives of the function represented by the network
- complexity terms which control number of units and weights or their precision.

If the network extracts characteristics which are unique for training data only (the data points which have been drawn), stem from noise, or are due to outliers in the training data then overfitting is present. In general all regularization terms which avoid that the network can extract such characteristics from the training data can regularize. These terms allow the network only to extract the most prominent characteristics which are not observed at one example but for many examples.

The regularization can be done

- during or
- after

training the network.

Regularization during training has the problem that there is a trade-off between decreasing the empirical risk and the complexity. This trade-off is difficult to regulate during learning. Advantage is that a constant pressure is on the learning to efficiently extract the structures.

Regularization after training has the problem that the network can be spoiled by the training procedure. Important structures may distributed over the whole network so that pruning weights or nodes is impossible without destroying the important structure. E.g. a structure which can be represented by a small subnetwork may be in a large architecture containing copies of this small subnetwork. If now one of these copies is deleted the error decreases because it contributes to the whole output. On the other hand the subnetwork share the task of representing the important structure therefore have free capacity to represent more characteristics of the training data including noise. Advantage of the regularization after training is that one can better control the complexity and the trade-off between empirical error and complexity.

Because all regularization must be expressed through weight values, a problem appears for all these methods. The network function can be implemented through different weight values and some weight values may indicate lower complexity even if they implement the same function.

6.4.5.1 Early Stopping

To control the complexity of the network it is possible to stop learning before the minimum is reached. This regularization technique is called "*early stopping*" and belongs to regularization after learning. Early stopping can be seen as a training procedure which produces a sequences of networks with increasing complexity. Complexity is controlled by selecting one of these networks after learning.

During learning the network will first extract the most dominant rules, that is the most important structures in the data. Here "important" or "dominant" means that the rule can be applied or structure is present at many training examples. These rules or structures can decrease the training error most efficiently because the loss decreases for many examples. Later in the learning procedure characteristics are found which apply only to few or just one example.

If the network is initialized with small weights and sigmoid activation functions are used, then at the beginning of learning the network implements an almost linear function. The initial network function is made more and more nonlinear during learning. For a highly nonlinear network function the weights in the network must be large. This means as early the learning is stopped as more linear the network function will be and as lower is the complexity.

In Fig. 3.15 the relation between the test error and the training error is shown. If the complexity grows with learning time, then the test error first decreases and then increases again. Optimal would be to stop where the test error is minimal.

To analytically determine the best stopping time is complicated. Only rough guesses can be given and do not help in practice.

A practical way is to use a validation set besides the training set to determine the best stopping time. However the validation examples are lost as training examples.

Disadvantage of early stopping is that there is no pressure on the learning algorithm to focus on the most prominent structure and efficiently use the resources. Also dominant but complicated structures are not found because to extract them would take more time.

6.4.5.2 Growing: Cascade-Correlation

The architecture, the number of units and weights, strongly influences the complexity of the network. *Growing algorithms* start with small networks and add stepwise new units or new weights which amount to increasing the complexity.

We classify growing algorithms as regularization techniques after training because during training no regularization is done. Regularization is obtain through selecting a network from a hierarchy of networks with increasing complexity similar to early stopping.

Well know algorithms for classification are the *pocket* and the *tiling* algorithm.

We will introduce the best known algorithm: *cascade-correlation*.

Cascade-correlation works as follows: If a new hidden unit k is added then first its weights are trained to maximize the correlation between the residual error and the unit's activation. The objective to maximize is

$$C_k \;=\; \sum_{j=N-O+1}^{N} \left| \sum_{i=1}^{l} (a_k \;-\; \bar{a}_k)\;(\epsilon_j \;-\; \bar{\epsilon}_j) \right| \;, \tag{6.69}$$

where $\epsilon_j = (a_j \;-\; y_{j-N+O})$ is the error of output unit j ($\bar{\epsilon}_j$ its mean value) and a_k is the activation of the new hidden unit k ($\bar{a}_k$ its mean value).

The derivative of C_k with respect to the weights is just

$$\frac{\partial C_k}{\partial w_{kj}} \;=\; \sum_{j=N-O+1}^{N} \pm \sum_{i=1}^{l} (\epsilon_j \;-\; \bar{\epsilon}_j)\;\; f'(\mathrm{net}_k)\; a_j \;, \tag{6.70}$$

where the sign is given by the sign of the correlation in eq. (6.69).

The training is very fast because only the incoming connections to the new hidden units have to be trained.

The hidden to output weights are then found by a linear least-square estimate (see eq. (3.98)) if the output unit(s) is (are) linear. Otherwise again a gradient descent method can be applied to determine the hidden to output weights. Here also a single layer of weights has to be trained. Fig. 6.9 shows the architecture of the cascade-correlation network.

Disadvantage of growing or "constructive" algorithms is that units or weights are added only in small groups. Therefore, the combined effect of units cannot be used. If for example a reduction of the error can be done only through a couple of units then this would not be detected. It is unclear when the error starts to decrease therefore it is hard to decide when to stop or add a new unit.

6.4.5.3 Pruning: OBS and OBD

The opposite approach to growing is pruning. With *pruning* first a neural network is trained until a local minimum of the training error (empirical risk) is found. Now the complexity of trained network is reduced by removing weights (setting them to zero) from the network. Those

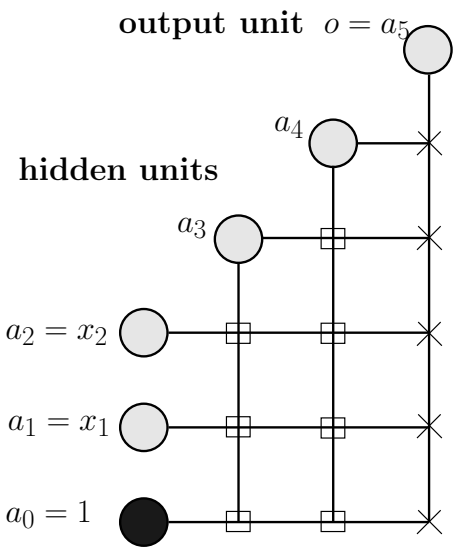


Figure 6.9: Cascade-correlation: architecture of the network. Squares are weights which are trained whereas crosses represent weights which retrained only after the addition of a hidden unit. First the hidden unit 3 with activation a_3 was added and then hidden unit 4 with activation a_4.

weights are removed which do not dramatically increase the training error to ensure that the major information extracted from the training data is still coded in the network.

Pruning methods are typical methods for regularization after training.

Method like in [White, 1989, Mozer and Smolensky, 1989, Levin et al., 1994] remove units and in [Moody, 1992, Refenes et al., 1994] even input units are removed. Before removing a units or a weight its importance must be measured. The importance is determined by the influence or contribution of the unit to producing the output.

To determine the increase of the error if a weight is deleted, a Taylor expansion of the empirical error $R(\boldsymbol{w})$ around the local minimum $\boldsymbol{w}^*$ is made (that was already done in eq. (5.72). The gradient vanishes in the minimum $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) = \mathbf{0}$, therefore we obtain with $\Delta\boldsymbol{w} = (\boldsymbol{w} - \boldsymbol{w}^*)$

$$R(\boldsymbol{w}) = R(\boldsymbol{w}^*) + \frac{1}{2}\Delta\boldsymbol{w}^T \boldsymbol{H}(\boldsymbol{w}^*)\, \Delta\boldsymbol{w} + O\left((\Delta\boldsymbol{w})^3\right) , \tag{6.71}$$

where $\boldsymbol{H}$ is the Hessian.

For small $|\Delta\boldsymbol{w}|$ the error increase $R(\boldsymbol{w}) - R(\boldsymbol{w}^*)$ for $\boldsymbol{w} = \boldsymbol{w}^* + \Delta\boldsymbol{w}$ is determined by $\Delta\boldsymbol{w}^T \boldsymbol{H}(\boldsymbol{w}^*)\, \Delta\boldsymbol{w}$.

To delete the weight w_i we have to ensure

$$\boldsymbol{e}_i^T\, \Delta\boldsymbol{w} + w_i = 0 , \tag{6.72}$$

where $\boldsymbol{e}_i$ is the unit vector with an one at the i-th position and otherwise zeros.

"Optimal Brain Damage" (OBD) LeCun et al. [1990] uses only the main diagonal of the Hessian which can be computed in $O(W)$ time as we saw through eq. (6.36). The error increase is

determined by

$$\sum_i H_{ii} \ (\Delta w_i)^2 \ . \tag{6.73}$$

If we want to remove a weight then this term should be as small as possible. At the minimum the Hessian is positive definite, which means that for all $\boldsymbol{v}$

$$\boldsymbol{v}^T \boldsymbol{H} \ \boldsymbol{v} \ \geq \ 0 \ , \tag{6.74}$$

therefore

$$\boldsymbol{e}_i^T \boldsymbol{H} \ \boldsymbol{e}_i \ = \ H_{ii} \ \geq \ 0 \ . \tag{6.75}$$

Therefore the vector $\Delta \boldsymbol{w}$ is chosen as

$$\Delta \boldsymbol{w} \ = \ - \ w_i \ \boldsymbol{e}_i \tag{6.76}$$

so that all components except the i-th are zero. The minimal error increase is given by

$$\min_i H_{ii} \ w_i^2 \tag{6.77}$$

and the weight w_k with

$$k \ = \ \arg\min_i H_{ii} \ w_i^2 \tag{6.78}$$

is removed.

If the error increases after removal of some weights the network is retrained again until a local minimum is found.

OBD is not able to recognize redundant weights. For example if two weights performing the same tasks which can be done by one weight alone: assume each of the two weights has value of 0.5 then this may be equivalent to set one of the weights to 1 and delete the other.

OBD was successfully applied to recognize handwritten zip (postal) codes where a network with 10.000 parameters was reduced to 1/4 of its original size.

A method called "Optimal Brain Surgeon" (OBS) Hassibi and Stork [1993] which uses the complete Hessian was introduced by Hassibi & Stork. The full Hessian allows for correcting other weights which also allows to detect redundant weights and remove redundancies. Also retraining can be avoided are at least made reduced.

Above Taylor expansion and the constraint of removing weight i can be stated as an quadratic optimization problem

$$\begin{aligned} \min_{\Delta \boldsymbol{w}} \quad & \frac{1}{2} \ \Delta \boldsymbol{w}^T \boldsymbol{H} \ \Delta \boldsymbol{w} \\ \text{s.t.} \quad & \boldsymbol{e}_i^T \ \Delta \boldsymbol{w} \ + \ w_i \ = \ 0 \end{aligned} \tag{6.79}$$

The Lagrangian is

$$L \;=\; \frac{1}{2}\,\Delta \boldsymbol{w}^T \boldsymbol{H}\,\Delta \boldsymbol{w} \;+\; \alpha\left(\boldsymbol{e}_i^T\,\Delta \boldsymbol{w} \;+\; w_i\right)\;. \tag{6.80}$$

The derivative has to be zero

$$\frac{\partial L}{\partial \Delta \boldsymbol{w}} \;=\; \boldsymbol{H}\,\Delta \boldsymbol{w} \;+\; \alpha\,\boldsymbol{e}_i \;=\mathbf{0} \tag{6.81}$$

which is

$$\Delta \boldsymbol{w} \;=\; -\,\alpha\,\boldsymbol{H}^{-1}\,\boldsymbol{e}_i \tag{6.82}$$

Further we have

$$w_i \;=\; -\,\boldsymbol{e}_i^T\,\Delta \boldsymbol{w} \;=\; \alpha\,\boldsymbol{e}_i^T \boldsymbol{H}^{-1}\,\boldsymbol{e}_i \;=\; \alpha\,\boldsymbol{H}_{ii}^{-1} \tag{6.83}$$

which gives

$$\alpha \;=\; \frac{w_i}{\boldsymbol{H}_{ii}^{-1}} \tag{6.84}$$

and results in

$$\Delta \boldsymbol{w} \;=\; -\,\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\,\boldsymbol{H}^{-1}\,\boldsymbol{e}_i\;. \tag{6.85}$$

The objective is

$$\frac{1}{2}\,\Delta \boldsymbol{w}^T \boldsymbol{H}\,\Delta \boldsymbol{w} \;=\; \frac{1}{2}\frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}}\;, \tag{6.86}$$

where one $\boldsymbol{H}^{-1}$ vanishes with $\boldsymbol{H}$ and the other has the form $\boldsymbol{e}_i^T \boldsymbol{H}^{-1}\,\boldsymbol{e}_i \;=\; \boldsymbol{H}_{ii}^{-1}$.

The criterion for selecting a weight to remove is the

$$\frac{1}{2}\frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}} \tag{6.87}$$

and the correction is done by

$$\Delta \boldsymbol{w} \;=\; -\,\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\,\boldsymbol{H}^{-1}\,\boldsymbol{e}_i\;. \tag{6.88}$$

Exercise: Why is the expression $\frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}}$ always larger than or equal to zero?

An approximation technique for the inverse Hessian was given in Section 6.4.4.

Problem with OBS is that most weights may be larger than 1.0 and the Taylor expansion is not valid because higher terms in $\Delta \boldsymbol{w}$ do not vanish. However in practice OBS does also work in these cases because higher order derivatives are small.

Heuristics to improve OBS are

- checking for the first order derivative $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) = \mathbf{0}$; if the gradient is not zero then also the linear term can be used to determine the increase of the error
- retraining after weight deletion
- checking for $\left\| \boldsymbol{I} - \boldsymbol{H}^{-1} \boldsymbol{H} \right\| > \beta$ to see the quality of the approximation of the inverse Hessian.
- check for weight changes larger than γ (e.g. $\gamma = 5.0$), if changes appear which are larger than γ then the weight correction should be scaled.

In general OBS will not find the smallest network because the gradient descent training was not forced to use the smallest architecture to solve the problems. Therefore some structures are distributed in a larger network which avoids to prune the network to an optimal size.

6.4.5.4 Weight Decay

Now we move on to regularization during learning. In general the error function is extended by a complexity term Ω which expresses the network complexity as a function of the weights:

$$R(\boldsymbol{w}) = R_{\text{emp}}(\boldsymbol{w}) + \lambda \, \Omega(\boldsymbol{w}) \, . \tag{6.89}$$

This has the same form as the expressions in Theorem 3.9. In Section 3.6.5 the complexity term was expressed through the margin and the margin in turn through the weight vector. Therefore this is the expression which is also optimized by support vector machines.

Gradient descent in now performed on this extended error term.

$$\nabla_{\boldsymbol{w}} R(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}) + \lambda \, \nabla_{\boldsymbol{w}} \Omega(\boldsymbol{w}) \, . \tag{6.90}$$

The best known term for $\Omega(\boldsymbol{w})$ is *weight decay*, where small absolute weights values are preferred over large absolute weight values.

Motivation for this term as a complexity term is similar to early stopping with small weights. If a sigmoid activation function is used then small weights kept the activation in the linear range. Therefore small absolute weight values prefer linear solutions. The same argument holds for higher order derivatives of the network function. High derivatives can only be obtained by large weights, therefore weight decay is a bias towards linear solutions.

Different weight decay terms were suggested like a term based on the 1-norm $\|\boldsymbol{w}\|_1$ [Hanson and Pratt, 1989] (Laplace distribution of weights in the Bayes treatment see Chapter 7), a term based on $\log\left(1 + \boldsymbol{w}^T \boldsymbol{w}\right)$ [Williams, 1994] (Cauchy distribution of weights in the Bayes treatment see Chapter 7), or the 2-norm $\|\boldsymbol{w}\|^2 = \boldsymbol{w}^T \boldsymbol{w}$ [Krogh and Hertz, 1992] (Gaussian distribution of weights in the Bayes treatment see Chapter 7).

Later in Section 7.6 we will investigate how to adjust the hyper-parameter λ for the Gaussian weight distribution according to [MacKay, 1992].

All these complexity terms tend to replace large absolute weights through a couple of small absolute weights. For large networks also small absolute weights can lead to high nonlinearities if the small weights accumulate to transfer a signal.

Therefore the complexity term was further developed and a threshold for large weights introduced [Weigend et al., 1991]. Only changes of weights near the threshold have large impact on the complexity term. All weights below the threshold are pushed towards zero and the weights beyond the threshold are not further penalized and can be even made larger in order to approximate the training data.

The weight decay term according to [Weigend et al., 1991] is

$$\Omega(\boldsymbol{w}) \;=\; \sum_i \frac{w_i^2/w_0^2}{1 \;+\; w_i^2/w_0^2} \;, \tag{6.91}$$

where w_0 is the threshold value. For example $w_0 = 0.2$.

To have more control and to better adjust the hyper-parameter λ the gradient of $\Omega(\boldsymbol{w})$ can be normalized and multiplied by the length of the gradient $\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w})$ which amount to a variable weighting of the complexity term.

For an MLP the weight decay terms can be separated for each layer of weights and individual hyper-parameters given for each layer. For example the input layer can be treated separately e.g. also for selecting or ranking input variables.

6.4.5.5 Training with Noise

To regularize and to avoid that the network extracts individual characteristics of few training examples either to the training examples noise is added or to networks components [Minai and Williams, 1994, Murray and Edwards, 1993, Neti et al., 1992, Matsuoka, 1992, Bishop, 1993, Kerlirzin and Vallet, 1993, Carter et al., 1990, Flower and Jabri, 1993].

For example Murray and Edwards [Murray and Edwards, 1993] inject white noise into the weights which results in a new complexity term. The complexity term consists of squared weight values and second order derivatives. However due to the second order derivatives the algorithms is very complex.

It can be shown that noise injection is for small weight values equivalent to Tikhonov regularization. Tikhonov regularization penalizes large absolute higher order derivatives of the output with respect to the inputs and therefore highly nonlinear functions.

6.4.5.6 Weight Sharing

Nowland and Hinton [Nowlan and Hinton, 1992] propose that the weights have special preferred values and should be grouped. Each group of weights should have similar values.

Each group has a preferred value and a certain range which says what weights are considered as similar. Therefore each group j is modeled by a Gaussian

$$G(\sigma_j, \mu_j) \;=\; \frac{1}{(2\,\pi)^{1/2}\;\sigma_j}\;\exp\left(-\frac{1}{2\,\sigma_j^2}\;(w \;-\; \mu_j)^2\right) \;. \tag{6.92}$$

Each group j has a prior size α_j which estimates what percentage of weights will belong to this group.

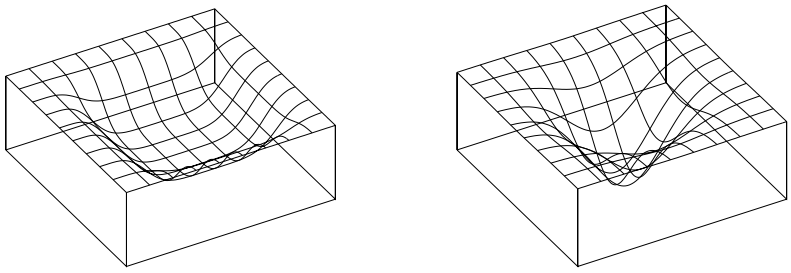


Figure 6.10: Left: example of a flat minimum. Right: example of a steep minimum.

We obtain for the probability of a weight belonging to group j

$$p(w \mid j) \;=\; \frac{\alpha_j \, G(\sigma_j, \mu_j)}{\sum_k \alpha_k \, G(\sigma_k, \mu_k)} \tag{6.93}$$

and the probability of observing this weight value

$$p(w) \;=\; \sum_j p(w \mid j) \; . \tag{6.94}$$

The value $p(w)$ should be maximized and we obtain for the derivative of $R(\boldsymbol{w})$ with respect to a single weight

$$\frac{\partial R(\boldsymbol{w})}{\partial w_i} \;=\; \frac{\partial R_{\mathrm{emp}}(\boldsymbol{w})}{\partial w_i} \;+\; \lambda \, \sum_j p(w_i \mid j) \, \frac{(w_i \;-\; \mu_j)}{\sigma_j^2} \; . \tag{6.95}$$

Also the centers μ_j and the variances σ_j^2 can be adaptively adjusted during learning.

6.4.5.7 Flat Minimum Search

Another algorithm for regularization during training is the "Flat Minimum Search" (FMS) algorithms [Hochreiter and Schmidhuber, 1997a]. It searches for large regions in the weight space where the network function does not change but the empirical risk is small. Each parameter vector from this region leads to the same empirical risk. Such a region is called "flat minimum" (see Fig. 6.10).

A steep minimum corresponds to a weight vector which has to be given with high precision in order to ensure low empirical error. In contrast a flat minimum corresponds to a weight vector which can be given with low precision without influencing the empirical risk. Precision is here defined as how exact the weight vector is given in terms of intervals. For example 1.5 means the interval $[1.45, 1.55]$ of length 0.1 and 1.233 means the interval $[1.2325, 1.2335]$ of length 0.001, where the later is give more precisely than the former but needs more digits to describe. The FMS algorithm removes weights and units and reduces the sensitivity of the output with respect to the weights and other units. Therefore is can be viewed as an algorithm which enforces robustness.

From the point of view of "Minimum Message Length" [Wallace and Boulton, 1968] and "Minimum Description Length" [Rissanen, 1978] fewer bits are needed to describe a flat minimum. That means a flat minimum corresponds to a low complexity neural network.

An error term $\Omega(\boldsymbol{w})$ describing the local flatness is minimized. This error term consists of first order derivatives and can be minimized by gradient methods based on Alg. 6.4.

The FMS error term is

$$\Omega(\boldsymbol{w}) = \frac{1}{2}\left(-L\log\epsilon + \sum_{i,j}\log\sum_{k=N-O+1}^{N}\left(\frac{\partial a_k}{\partial w_{ij}}\right)^2 + W\log\sum_{k=N-O+1}^{N}\left(\sum_{i,j}\frac{\left|\frac{\partial a_k}{\partial w_{ij}}\right|}{\sqrt{\sum_{k=N-O+1}^{N}(\frac{\partial a_k}{\partial w_{ij}})^2}}\right)^2\right), \tag{6.96}$$

where ϵ gives the tolerable output change as Euclidian distance (output changes below ϵ are considered to be equal).

The derivative is

$$\frac{\partial\Omega(\boldsymbol{w})}{\partial w_{uv}} = \sum_{k=N-O+1}^{N}\sum_{i,j}\frac{\partial\Omega(\boldsymbol{w})}{\partial\left(\frac{\partial a_k}{\partial w_{ij}}\right)}\frac{\partial^2 a_k}{\partial w_{ij}\,\partial w_{uv}} \tag{6.97}$$

which is with $\frac{\partial a_k}{\partial w_{ij}}$ as new variables

$$\nabla_{\boldsymbol{w}}\Omega(\boldsymbol{w}) = \sum_{k=N-O+1}^{N}\boldsymbol{H}^k\left(\nabla_{\frac{\partial a_k}{\partial w_{ij}}}\Omega(\boldsymbol{w})\right), \tag{6.98}$$

where $\boldsymbol{H}^k$ is the Hessian of the output unit a_k.

The algorithm has complexity of $O(O\,W)$ (remember that O is the number of output units).

FMS prefers as weight decay weights close to zero, especially outgoing weights from units which are not required (then the ingoing weights can be given with low precision). In contrast to weight decay FMS also prefers large weight values. Especially large negative bias weights to push the activation of units either to zero (outgoing weights can be given with low precision.). Other large weights push activation into regions of saturation (ingoing weights can be given with low precision).

6.4.5.8 Regularization for Structure Extraction

Regularization of neural networks can be used to extract structures from data.

The idea to build an *auto-associator* network, where the input and the output should be identical (see Fig. 6.11). That means an auto-associator is a network which represents the identity. Regularization ensures a low complex bottleneck which stores the information of the input for constructing the output.

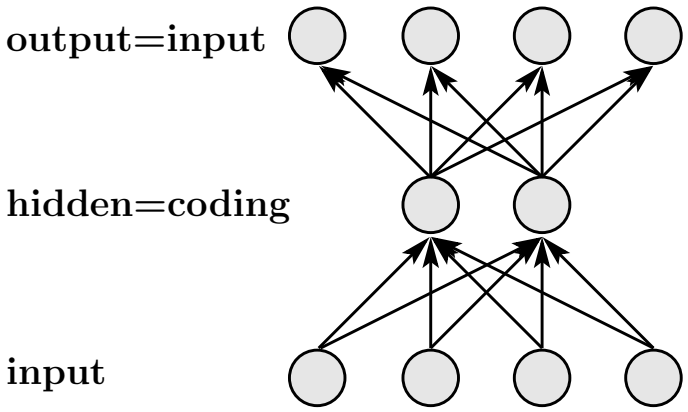
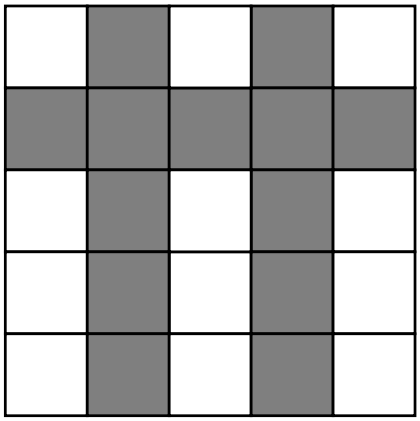


Figure 6.11: An auto-associator network where the output must be identical to the input. The hidden layer must code the input information to supply it to the output layer.



-0.5	+0.5	-0.5	+0.5	-0.5
+0.5	+0.5	+0.5	+0.5	+0.5
-0.5	+0.5	-0.5	+0.5	-0.5
-0.5	+0.5	-0.5	+0.5	-0.5
-0.5	+0.5	-0.5	+0.5	-0.5

Figure 6.12: Example of overlapping bars. The second and fourth vertical as well as the second horizontal bar is active. On the right the corresponding inputs to the neural network.

However using regularization the identity must be build using a low complexity network where also the output must not exactly match the input. The network is forced to extract the major structures from the network and present them at the output – only in this way the input can be generated sufficiently well.

In the following we show FMS applied to auto-association.

In the first task a 5×5 pixel square contains horizontal and vertical bars. See Fig. 6.12 for an example. A good coding strategy would be to code bars and not the single pixels.

To the input noise is added which gives training examples as shown in Fig. 6.13.

The result of the auto-associator trained with FMS is shown in Fig. 6.14. The bar structure is extracted and efficiently coded.

Now we extract structures from images of size 150×150 pixel with 256 grey values. The input to the network is a 7×7 square of these images.

We analyze the image of a village as in Fig. 6.15. The image is black except for special white

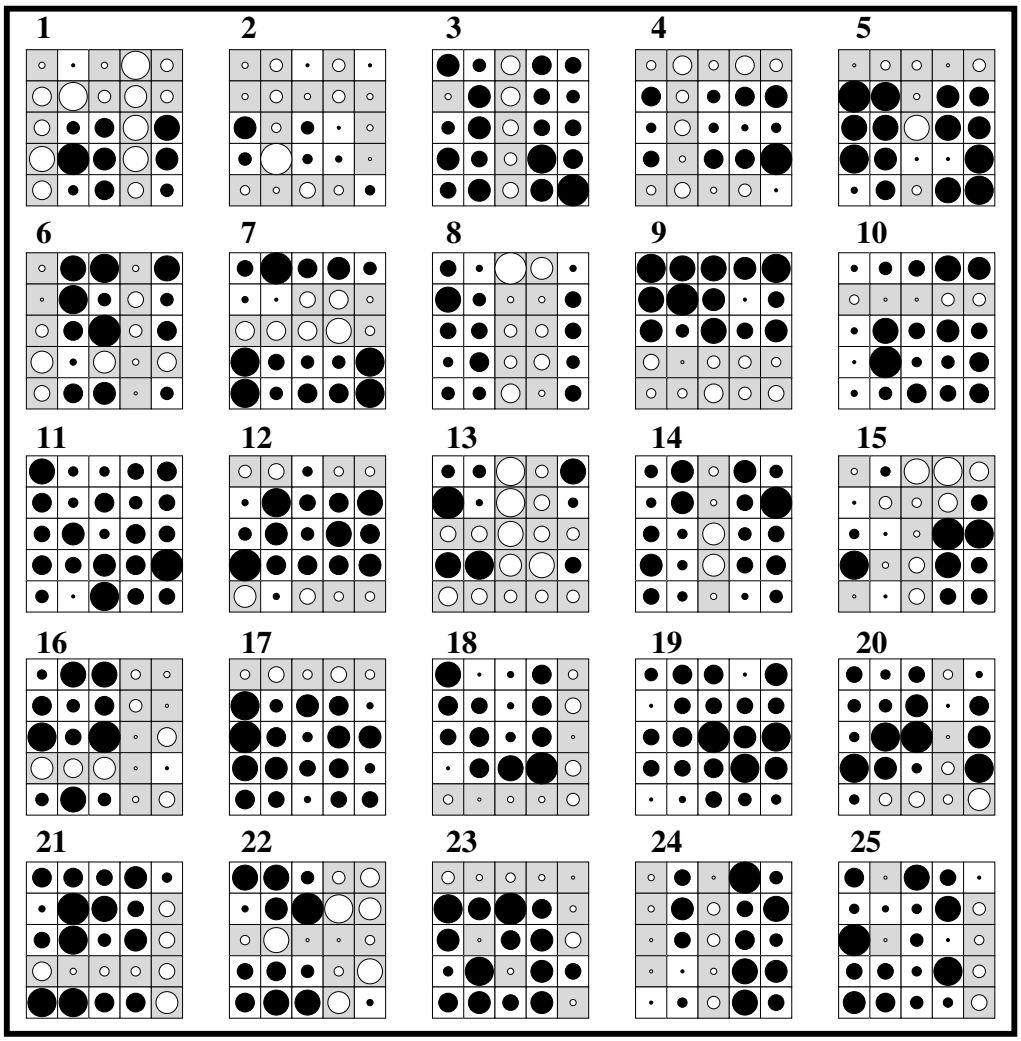


Figure 6.13: 25 examples for noise training examples of the bars problem where each example is a 5×5 matrix. The white circles are positive and the black circles are negative values. The radius of the circle is proportional to the absolute value. The values are normalized for each 5×5 matrix so that one maximal circle is present.

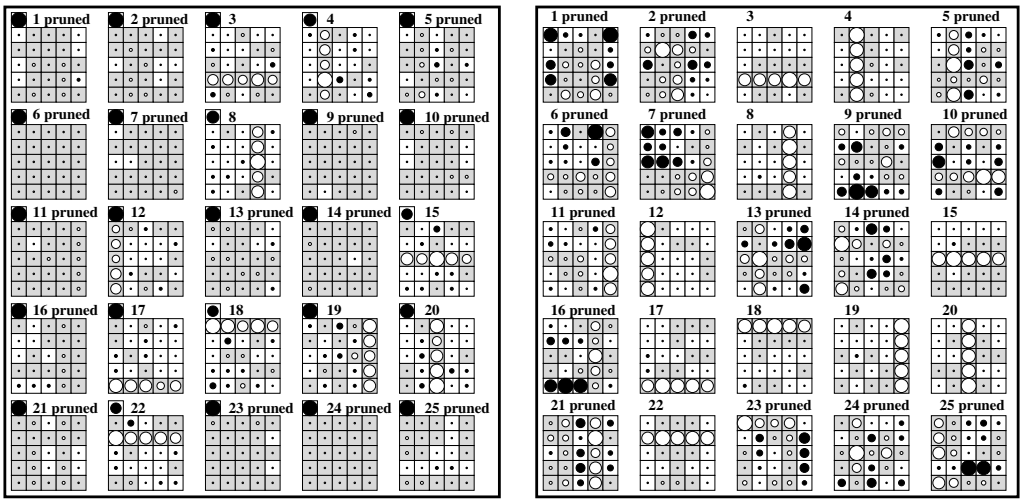


Figure 6.14: Noise bars results for FMS. Left: weights from inputs to the hidden units. Each 5×5 square represents one hidden unit with its ingoing weights. At the upper left corner the bias weights are given. The color coding is as in Fig. 6.13 but also the bias weights is used for normalization. Units which are marked by "pruned" have been removed by FMS. Right: weights from hidden units to output units.

regions which show buildings or streets. The result is shown in Fig. 6.15. The black background with white structures was extracted by localized white spot detectors.

We analyze the image of wood cells as in Fig. 6.17. Fig. 6.18 shows the results after training with FMS.

We analyze the image of wood piece with grain as shown in Fig. 6.19. The results after training with FMS are shown in Fig. 6.20.

In all these experiments a regularized auto-associator was able to extract structures in the input data.

6.4.6 Tricks of the Trade

6.4.6.1 Number of Training Examples

According to Baum & Haussler 1989 if from

$$l \;\geq\; \frac{W}{\epsilon}\,\log_2\left(\frac{M}{\epsilon}\right) \tag{6.99}$$

examples a fraction $(1 - \epsilon/2)$ are correctly classified then a fraction of $(1 - \epsilon)$ of future examples are classified correctly. Here M is the number of units.

Further they found that

$$d_{\mathrm{VC}} \;\geq\; (M \;-\; 2)\,d\,, \tag{6.100}$$

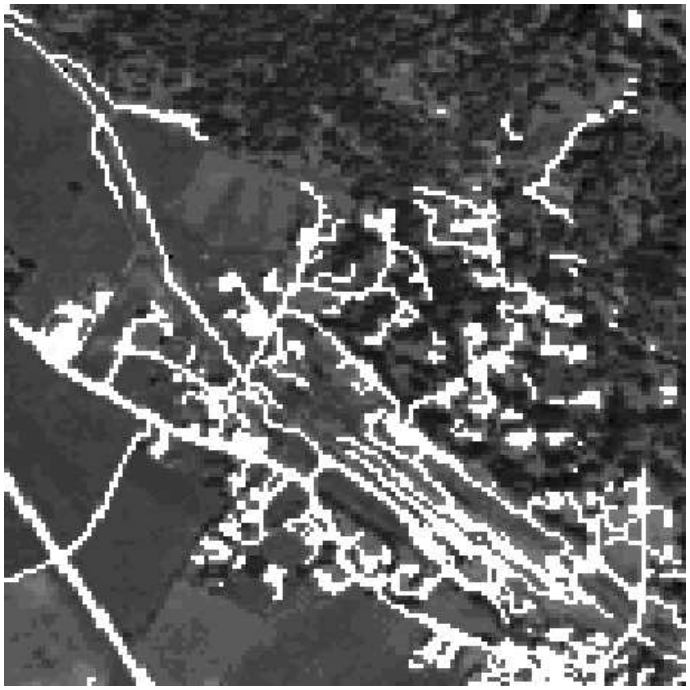


Figure 6.15: An image of a village from air.

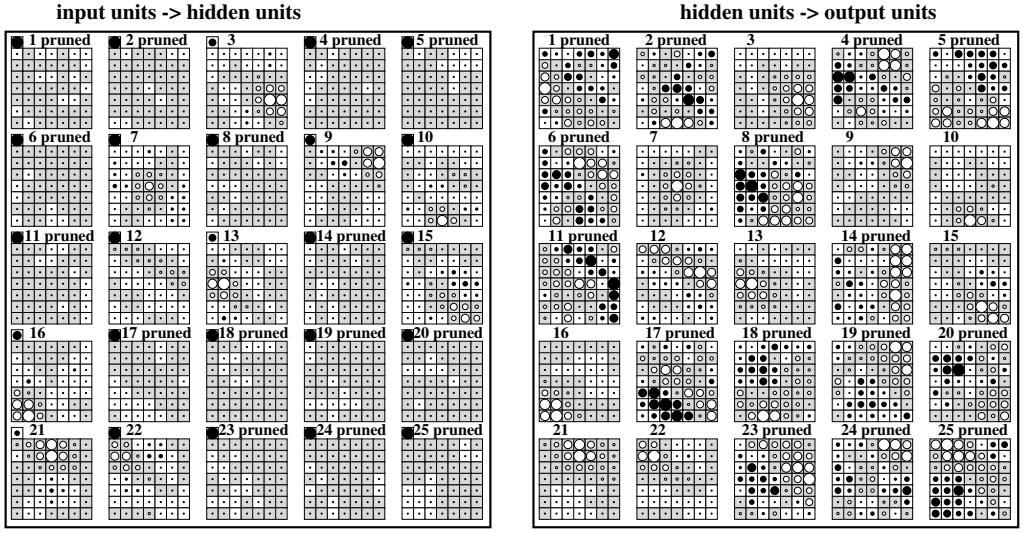


Figure 6.16: Result of FMS trained on the village image. Left: weights from input units to hidden units. The most units are deleted. Right: weights from hidden units to output units.

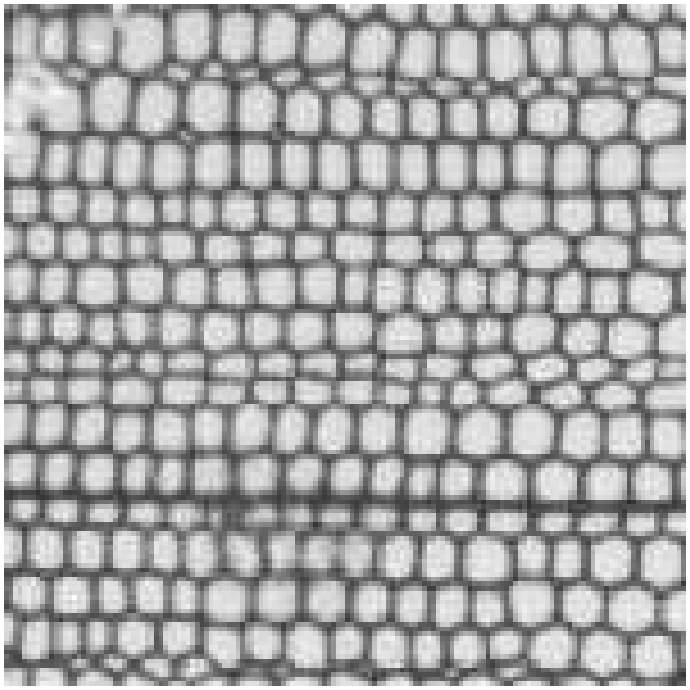


Figure 6.17: An image of wood cells.

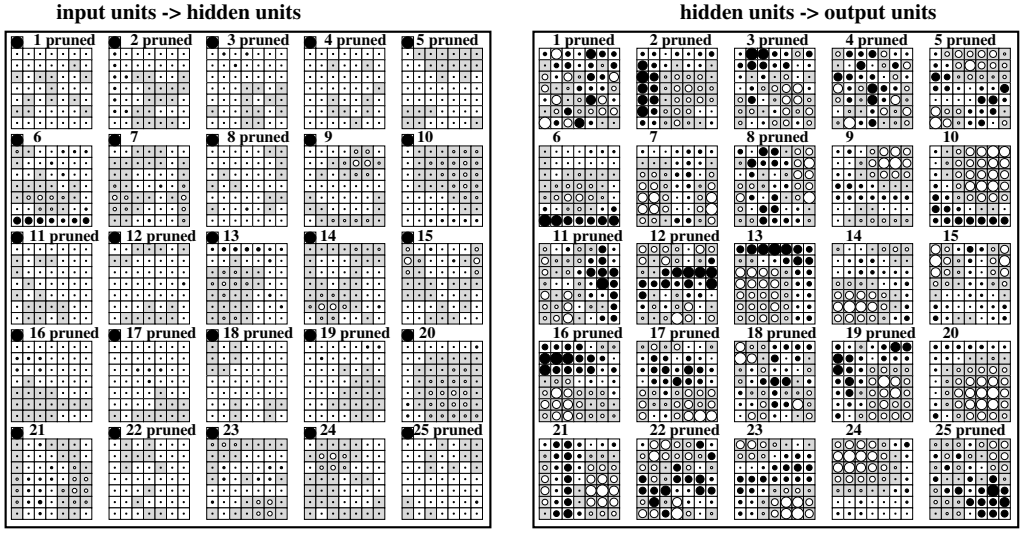


Figure 6.18: Result of FMS trained on the wood cell image. Left: weights from input units to hidden units. The most units are deleted. Right: weights from hidden units to output units.

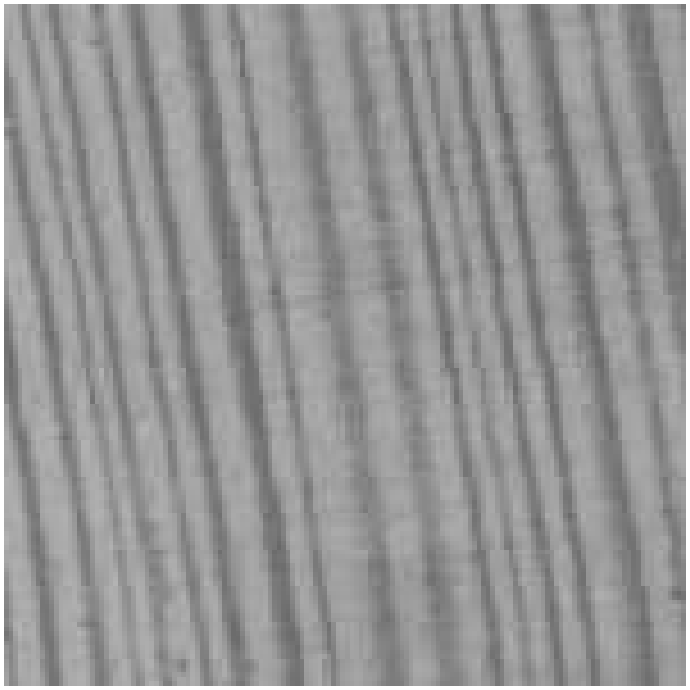


Figure 6.19: An image of a wood piece with grain.

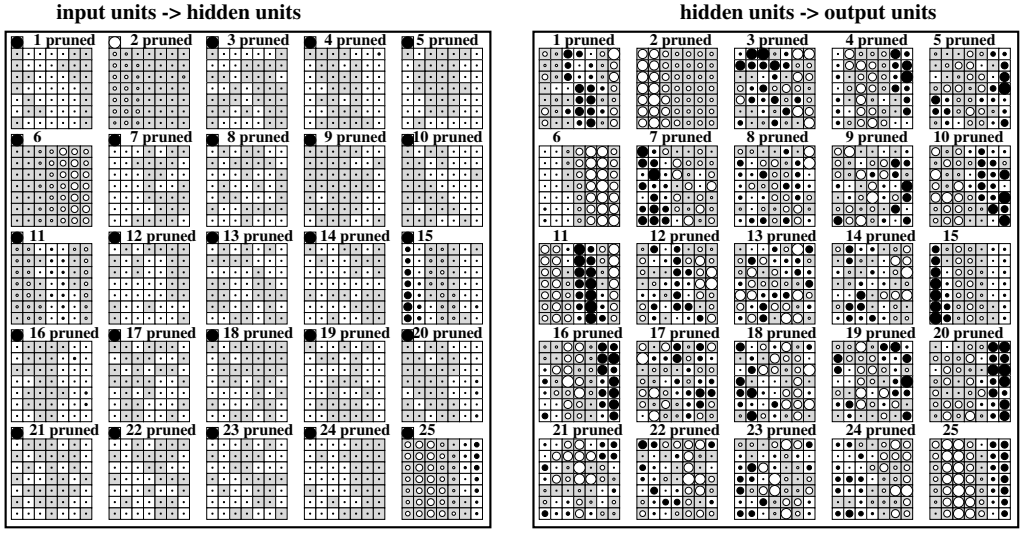


Figure 6.20: Result of FMS trained on the wood piece image. Left: weights from input units to hidden units. The most units are deleted. Right: weights from hidden units to output units.

where d is the number of inputs. For a two-layered network with one output unit we have $d\,(M - d - 1)$ weights from input to the $(M - d - 1)$ hidden units, $(M - d - 1)$ weight from the hidden units to the output, $(M - d - 1)$ hidden bias weights, and a output bias which gives

$$W \;=\; (d+2)\,(M - d - 1) \;+\; 1\,. \tag{6.101}$$

Approximatively we have

$$W \;=\; M\,d\,. \tag{6.102}$$

For $\epsilon = 0.1$ we have

$$l \;\geq\; W/\epsilon \;=\; 10\,W\,. \tag{6.103}$$

That amount to 10 times more examples than weights in the network.

6.4.6.2 Committees

In many application the performance is more robust if over different networks is averaged.

Let g_j the function which is represented by the j-th network then as an committee output with C members we obtain

$$g(\boldsymbol{x}) \;=\; \frac{1}{C}\,\sum_{j=1}^{C} g_j(\boldsymbol{x})\,. \tag{6.104}$$

We assume that

$$g_j(\boldsymbol{x}^i) \;=\; y^i \;+\; \epsilon_j^i\,, \tag{6.105}$$

where ϵ_j is distributed with zero mean $\mathrm{E}(\epsilon_j) \;=\; 0$ for each j and with zero covariance $\mathrm{E}(\epsilon_j\,\epsilon_k) \;=\; 0$. We further assume that all ϵ_j follow the same distribution for all j.

The expected error of the committee is

$$\begin{aligned}
&\mathrm{E}\left(\left(\frac{1}{C}\,\sum_{j=1}^{C} g_j(\boldsymbol{x}) \;-\; y\right)^2\right) \;=\; \mathrm{E}\left(\left(\frac{1}{C}\,\sum_{j=1}^{C} \left(g_j(\boldsymbol{x}) \;-\; y\right)\right)^2\right) \;= \\
&\mathrm{E}\left(\left(\frac{1}{C}\,\sum_{j=1}^{C} \epsilon_j\right)^2\right) \;=\; \frac{1}{C^2}\,\sum_{j=1}^{C} \mathrm{E}\left(\epsilon_j^2\right) \;= \\
&\frac{1}{C}\,\mathrm{E}\left(\epsilon_j^2\right)\,,
\end{aligned} \tag{6.106}$$

where $\mathrm{E}\left(\epsilon_j^2\right)$ is the error obtained by a single network.

That means the error was reduced by a factor of $\frac{1}{C}$ because the individual error are averaged out.

However that does not work in practice because the errors of the individual networks are not de-correlated. In the worst case all networks converge to the same minimum and no improvement is obtained.

If the error correlation matrix is known then a weighted average

$$g(\boldsymbol{x}) \; = \; \sum_{j=1}^{C} \alpha_j \; g_j(\boldsymbol{x}) \tag{6.107}$$

with $\sum_{j=1}^{C} \alpha_j = 1$ is possible. The α_j can be determined by using the correlation matrix.

6.4.6.3 Local Minima

In principle are the optimization techniques for neural networks prone to finding only local minima of the empirical error and not its global minimum.

However in practice large enough networks never get stuck in local minima because the empirical error can be minimized until it reaches zero, i.e. all training examples are perfectly learned.

More problematic is structural risk minimization where the risk is the empirical error plus a complexity term. Here most sensitive is the adjustment of the hyper-parameter which scales the complexity term.

The only way to find a good solution is to use a simple complexity term which has not many optima in itself and to explore different regions in weight space.

The best approach will be treated later in Section 7.8 where the optimum of the risk is found by monte carlo sampling.

6.4.6.4 Initialization

Initialization with small weights is to prefer because of three reasons. First the network starts with simple functions which are almost linear. Secondly the derivatives are large in the linear activation range compared to ranges with saturation of the units. Thirdly, weights around zero have on average the minimal distance to their optimal values if we assume a zero mean distribution of the final weight values.

Typical initialization values may be uniformly in $[-0.1; 0.1]$.

Sometimes bias weights can have special treatment for initialization. The bias weight to the output can be adjusted that the output unit supplies the mean target value at the beginning of learning.

Bias weights to hidden units can be staged so that the hidden units are not used all at once and doing the same job. Ideally the hidden unit with the smallest negative bias is used first and then the second smallest and so on.

The negative bias can also be used for regularization because some units are kept away from processing the output and only if there is enough error flowing back they come into the game.

6.4.6.5 δ-Propagation

During training different training examples have different contribution to the weight update. The contribution of few examples is sometimes important.

To speed up learning the training examples with empirical error smaller than δ are marked and not used for training in the next epochs. After certain number of epochs a sweep through all training examples is made and again the examples with error smaller than δ are marked and not used in the next epochs.

6.4.6.6 Input Scaling

All input components should be scaled so that they are within $[-1, 1]$ which is the activation range of standard units. Larger input values lead to large weight updates and unstable learning.

Often each input component is normalized to zero mean and variance 1.

If for a input component positive and negative values express different facts then this should be kept and normalization to zero mean should be avoided.

If a input component contains outliers that is extreme large absolute values then these values should be mapped to a maximum or a squashing function should be used. For example if for examples the value of one component is in $[-1, 1]$ and one example has a value of 100 of this component, then after scaling the component is for all except one value in $[-0.01, 0.01]$ and learning of this component is difficult. Also the weight update of the large value is 100 times larger than any other update for equal delta-error at an unit in the first hidden layer. That means one example overwrites the information of many other examples.

6.4.6.7 Targets

For classification it is sometimes useful to use targets of 0.2 and 0.8 for sigmoid activation in $[0; 1]$ instead of 0 and 1 or targets of -0.8 and 0.8 for sigmoid activation in $[-1; 1]$ instead of -1 and 1. Problem with targets at the boundary of the activation interval is that some examples may get stuck in saturated regions and the derivatives are small. In this case a wrongly classified example takes a long time to be made correctly.

For a multi-class network it is useful to define an output unit for each class which obtains 1 as target if the input belongs to the according class and zero otherwise.

If in a classification task the size of the classes is different then it can help to assign higher outputs to classes with few members so that the overall error contribution is similar for all classes.

6.4.6.8 Learning Rate

Typical learning rates per example are 0.1 or 0.01. In batch learning the learning rate must be divided by the number of examples in the training set.

However in batch learning the learning rate can be increased again because error signals get superimposed and the accumulative update signal is not the sum of all absolute update values. The updates for one weight are positive and negative depending on the example.

6.4.6.9 Number of Hidden Units and Layers

In many practical applications it was of advantage to have more hidden units in a layer than output and input units.

A network with two hidden layers is appropriate for many tasks.

Shortcut connections should be avoided because these connections obtain a larger error signal (see Section 6.6.5) and start oscillate so that the between layer weights are hard to adjust.

For some applications it is useful to reduce the activation of the constant bias unit to 0.1 because its outgoing weights obtain the largest update signal and begin to oscillate.

6.4.6.10 Momentum and Weight Decay

In Section 5.2 the momentum term was introduced which in most cases help to speed up the back-propagation algorithm.

For regularization weight decay method is easy to realize.

6.4.6.11 Stopping

When should learning be stopped?

With regularization the empirical error may converge to a constant value which indicates that learning can be stopped.

Without regularization the empirical error may decrease by a certain rate to zero.

It is often interesting to know the maximal deviation of the output from the target value over the training examples. If the maximal deviation is below a threshold then learning can be stopped.

Other stopping criterion include the maximal weight change of the last epochs, the error improvement of the last epochs.

6.4.6.12 Batch vs. On-line

In many application learning can be sped up by using an on-line update (see Section 5.8), that means after each example the weights are immediately changed.

Here it is important to shuffle the training examples after each epoch in order to avoid update loops which make no progress.

Advantage of the on-line method is that all future examples see the weight update and are not evaluated on the old weights (e.g. the bias weight to the output may be adjusted after a few training examples and all other see the new bias weight.). On-line learning together with shuffling also includes a small random effect which helps to explore the local environment in weight space.

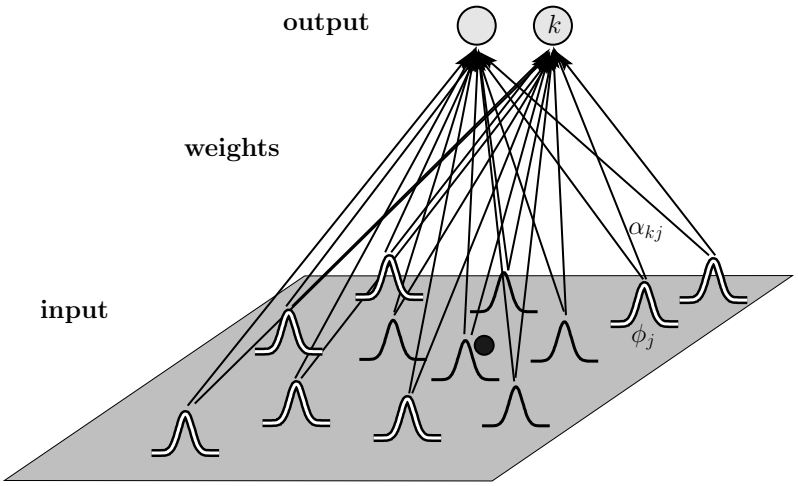


Figure 6.21: A radial basis function network is depicted. Inputs activate the Gaussians according how close the input is to the center. For the current input (dark point) the dark Gaussians are activated.

6.5 Radial Basis Function Networks

In contrast to the neural networks which we treated so far, the support vector machines where local approximators in the sense that training examples are used as a reference to classify new data points.

Local approximators also exist in the neural network literature, where the best known approach is *radial basis function* (RBF) networks.

Assume we have C basis functions ϕ then the RBF network is

$$g_k(\boldsymbol{x}) \;=\; \sum_{j=1}^{C} \alpha_{kj}\; \phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;+\; \alpha_{k0}\;. \tag{6.108}$$

For example Gaussian radial basis functions are

$$\phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;=\; \exp\left(-\,\frac{1}{2\,\sigma_j^2}\,\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|^2\right)\;. \tag{6.109}$$

Fig. 6.21 depicts how RBF networks are working.

The parameters of this network are the weighting coefficients α_{kj}, the centers $\boldsymbol{\mu}_j$ and the width σ_j.

It is possible to use a covariance matrix

$$\phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;=\; \exp\left(-\,\frac{1}{2}\,(\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)^T\,\boldsymbol{\Sigma}_j^{-1}\;(\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)\right)\;. \tag{6.110}$$

6.5.1 Clustering and Least Squares Estimate

To adjust the parameters of RBF networks there exist different approaches.

First we consider a two-stage procedure, where the in the first stage $\boldsymbol{\mu}_j$ and σ_j are determined and then the parameters α_{kj}.

In the first stage $\boldsymbol{\mu}_j$ and σ_j can be found by unsupervised methods like clustering or density estimation (mixture of Gaussians) as will be discussed in sections 10.6 and 10.6.1.

The values $\phi_j\left(\|\boldsymbol{x} - \boldsymbol{\mu}_j\|\right)$ are then given for the training set and can be summarized in the vector which can be expanded to a matrix $\boldsymbol{\Phi}$ which also summarizes all training examples.

The parameters α_{kj} can be summarized in the matrix $\boldsymbol{\Lambda}$ and the targets y_j are summarized in the vector $\boldsymbol{y}$ and the targets over the whole training set in the matrix $\boldsymbol{Y}$.

We obtain

$$\boldsymbol{Y} = \boldsymbol{\Lambda}\,\boldsymbol{\Phi} \tag{6.111}$$

which can be solved by a least-square estimate according to eq. (3.98) if a squared error is used:

$$\boldsymbol{\Lambda}^T = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{Y}\,. \tag{6.112}$$

6.5.2 Gradient Descent

Of course the RBF networks can be trained by gradient based methods. For the mean squared error we have $L(\boldsymbol{g}(\boldsymbol{x}),\boldsymbol{y}) = \sum_k \left(g_k(\boldsymbol{x}) - y_k\right)^2$ and if spherical Gaussians are use, the derivatives are

$$\begin{aligned}
&\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}),\boldsymbol{y})}{\partial \alpha_{kj}} = \left(g_k(\boldsymbol{x}) - y_k\right)\,\phi_j = \left(g_k(\boldsymbol{x}) - y_k\right) = \\
&\exp\left(-\frac{1}{2\,\sigma_j^2}\,\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2\right) \\
&\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}),\boldsymbol{y})}{\partial \sigma_j} = \sum_k \left(g_k(\boldsymbol{x}) - y_k\right)\,\alpha_{kj} \\
&\exp\left(-\frac{1}{2\,\sigma_j^2}\,\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2\right)\,\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{\sigma_j^3} \\
&\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}),\boldsymbol{y})}{\partial \mu_{jl}} = \sum_k \left(g_k(\boldsymbol{x}) - y_k\right)\,\alpha_{kj} \\
&\exp\left(-\frac{1}{2\,\sigma_j^2}\,\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2\right)\,\frac{(x_l - \mu_{jl})}{\sigma_j^2}\,.
\end{aligned} \tag{6.113}$$

6.5.3 Curse of Dimensionality

A compact region grows exponentially with d in a d-dimensional space. If we assume a cube where each axis is divided into k intervals then we obtain k^d small hypercubes. This fact is known as the "curse of dimensionality" [Bellman, 1961].

Therefore also the number of basis function should increase exponentially with d which in turn leads to complex models which are likely to overfit. According to [Stone, 1980], the number of training examples has to increase exponentially with the number of dimensions in order to ensure that an estimator also performs well for higher dimensional data.

That means RBF networks are not suited for high-dimensional data – similar statement holds for other local approximation methods.

Local approximation methods must assign a value to each region in space based on local information. However, if there is no local information then local methods fail to assign an appropriate value to data point at this region.

6.6 Recurrent Neural Networks

Until now we only considered neural networks with feed-forward connections, i.e. the directed connections did not form a loop. Without a loop there is a forward pass which can use the input variables to activate the network and produce output values.

Real neural networks however possess loops which serve to store information over time. That means the new activation depends on the old activation of the network.

The feed-forward network can be considered as a function which maps the input vector $\boldsymbol{x}$ to an output vector $\boldsymbol{g}(\boldsymbol{x})$.

Neural networks with loops, the so-called *recurrent networks*, map an input sequence

$$(\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t), \ldots, \boldsymbol{x}(T))$$

to an output sequence $(\boldsymbol{g}(1), \ldots, \boldsymbol{g}(t), \ldots, \boldsymbol{g}(T))$, where

$$\boldsymbol{g}(t) \;=\; \boldsymbol{g}\left(\boldsymbol{a}_0, \boldsymbol{x}(1), \ldots, \boldsymbol{x}(t)\right) \;, \tag{6.114}$$

where $\boldsymbol{a}_0$ is the initial activation of the network. The index t of the sequence elements is often called "time" because recurrent networks are often used to time series prediction. Also feed-forward networks can be used for time series prediction if an input vector is build of the current input together with past inputs. However such networks with an input window of the time series cannot see information outside the window and their complexity (number of parameters) increases with window size. Recurrent networks can process long sequences with few parameters and can use past information to optimally process information which they will see in the future sequence.

Fig. 6.22 shows an architecture of a recurrent network and Fig. 6.23 shows how a sequences is processed.

In bioinformatics recurrent networks are however used for sequence processing.

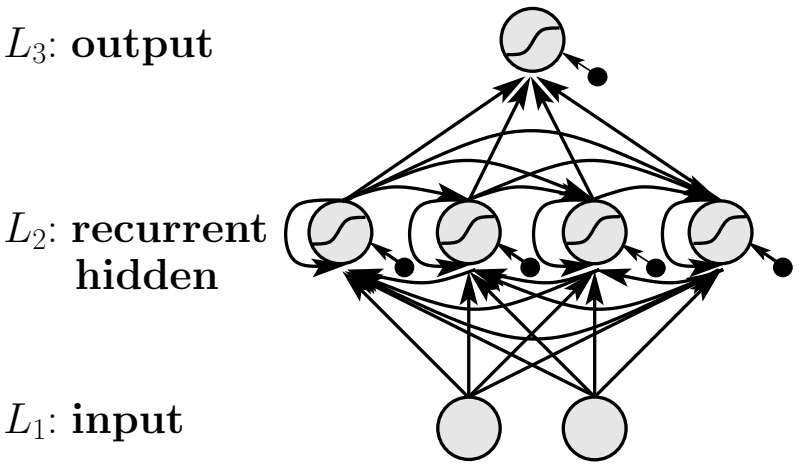


Figure 6.22: An architecture of a recurrent network. The recurrent hidden layer is fully connected, i.e. all units are interconnected. The hidden units are able to store information, i.e. information from previous inputs is kept in the hidden units.

The loops are used to store information from the previous time steps because the old activation of a unit can influence the new activation of other units or its own new activation. For example a unit which gets activated can keep this activation by a strong positive self-recurrent connection. Such a unit can therefore store the occurrence of an event.

The computational power of recurrent networks is that of Turing machines [Siegelmann and Sontag, 1991, Sun et al., 1991, Siegelmann, 1995].

6.6.1 Sequence Processing with RNNs

The activation functions and network inputs are now time dependent and are computed from the activations of previous time step:

$$\mathrm{net}_i(t) \;=\; \sum_{j=0}^{N} w_{ij}\; a_j(t-1) \tag{6.115}$$

$$a_i(t) \;=\; f(\mathrm{net}_i(t))\;. \tag{6.116}$$

Further we need initial activations $a_i(0)$ from which are the activations before the network sees the first input element.

Some of the units can be defined as input and some as output units.

For learning, for each input sequence $(\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t), \ldots, \boldsymbol{x}(T))$, a target sequence $(\boldsymbol{y}(1), \ldots, \boldsymbol{y}(t), \ldots, \boldsymbol{y}(T))$ is given. We denote by $\{\boldsymbol{x}(t)\}$ the sequence $(\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t))$ and by $\{\boldsymbol{y}(t)\}$ the sequence $(\boldsymbol{y}(1), \ldots, \boldsymbol{y}(t))$.

Similar to feed-forward networks we define the empirical error as

$$\nabla_{\boldsymbol{w}} R_{\mathrm{emp}}(\boldsymbol{w}, \{\boldsymbol{X}(T)\}, \{\boldsymbol{Y}(T)\}) \;= \tag{6.117}$$
$$\frac{1}{l}\sum_{i=1}^{l}\sum_{t=1}^{T} \nabla_{\boldsymbol{w}} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right)\;.$$

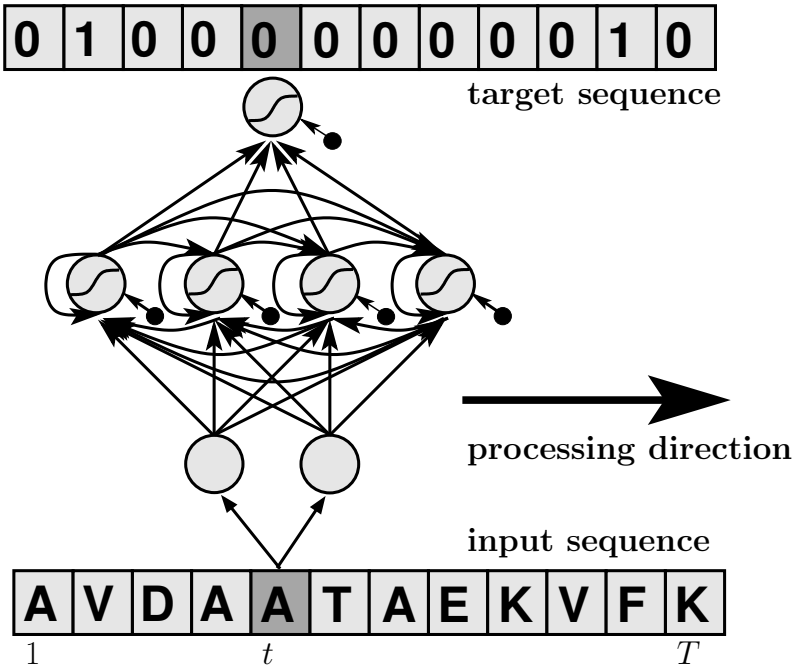


Figure 6.23: The processing of a sequence with a recurrent neural network. At each step the current input element is fed to the input units and the output unit should supply the target value.

The next subsections discuss how recurrent networks can be learned.

6.6.2 Real-Time Recurrent Learning

For performing gradient descent we have to compute

$$\frac{\partial}{\partial w_{uv}} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \tag{6.118}$$

for all w_{uv}.

Using the chain rule this can be expanded to

$$\begin{aligned} &\frac{\partial}{\partial w_{uv}} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \; = \\ &\sum_{k, k \text{ output unit}} \frac{\partial}{\partial a_k(t)} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \; \frac{\partial a_k(t)}{\partial w_{uv}} \; . \end{aligned} \tag{6.119}$$

For all units k we can compute

$$\frac{\partial a_k(t+1)}{\partial w_{uv}} \; = \; f'(\mathrm{net}_k(t+1)) \; \left(\sum_l w_{kl} \frac{\partial a_l(t)}{\partial w_{uv}} \; + \; \delta_{ku} a_v(t) \right) \; , \tag{6.120}$$

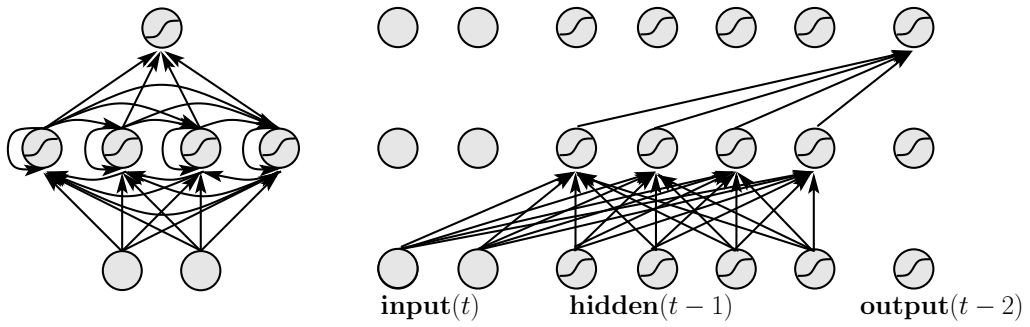


Figure 6.24: Left: A recurrent network. Right: the left network in feed-forward formalism, where all units have a copy (a clone) for each times step.

where δ is the Kronecker delta with $\delta_{ku} = 1$ for $k = u$ and $\delta_{ku} = 0$ otherwise.

If the values $\frac{\partial a_k(t)}{\partial w_{uv}}$ are computed during a forward pass then this is called *real time recurrent learning* (RTRL) [Werbos, 1981, Robinson and Fallside, 1987, Williams and Zipser, 1989, Gherrity, 1989].

RTRL has complexity of $O(W^2)$ for a fully connected network, where $W = N\,(N - I) = O(N^2)$ (each unit is connected to each other unit except the input units, which do not have ingoing connections).

Note that RTRL is independent of the length of the sequence, therefore RTRL is *local in time*.

6.6.3 Back-Propagation Through Time

A recurrent network can be transformed into a feed-forward network if for each time step a copy of all units is made. This procedure of *unfolding in time* is depicted in Fig. 6.24 for one step and the network unfolded over the whole time is depicted in Fig. 6.25.

After re-indexing the output by adding two time steps and re-indexing the hidden units by adding one time step we obtain the network from Fig. 6.26.

To the network of Fig. 6.26 can now be trained by standard back-propagation. This method is called *back-propagation through time* (BPTT) [Williams and Zipser, 1992, Werbos, 1988, Pearlmutter, 1989, Werbos, 1990].

We have

$$L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\{\boldsymbol{x}^i\}; \boldsymbol{w}\right)\right) \;=\; \sum_{t=1}^{T} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \tag{6.121}$$

for which we write for short

$$L \;=\; \sum_{t=1}^{T} L(t)\;. \tag{6.122}$$

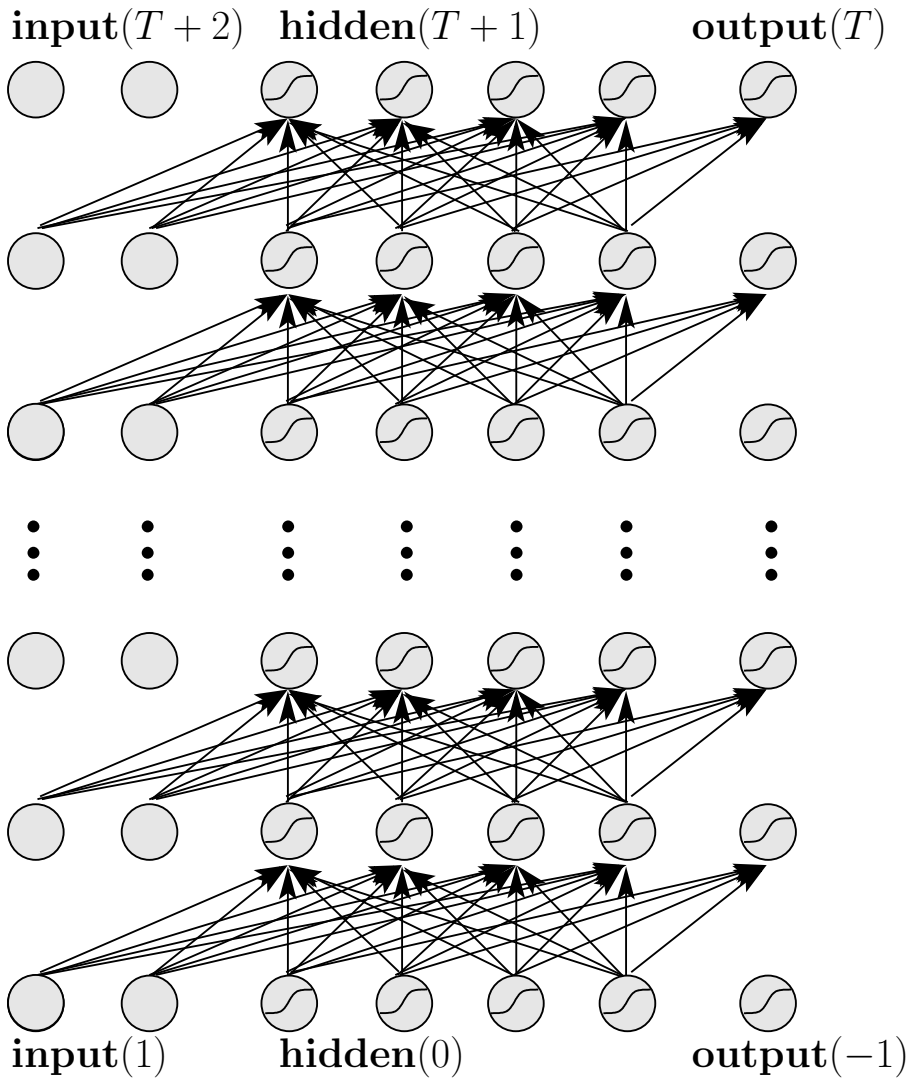


Figure 6.25: The recurrent network from Fig. 6.24 left unfolded in time. Dummy inputs at time $(t+1)$ and $(t+2)$ and dummy outputs at $(t-2)$ and $(t-1)$ are used. The input is shifted two time steps against the output to allow the input information be propagated through the network.

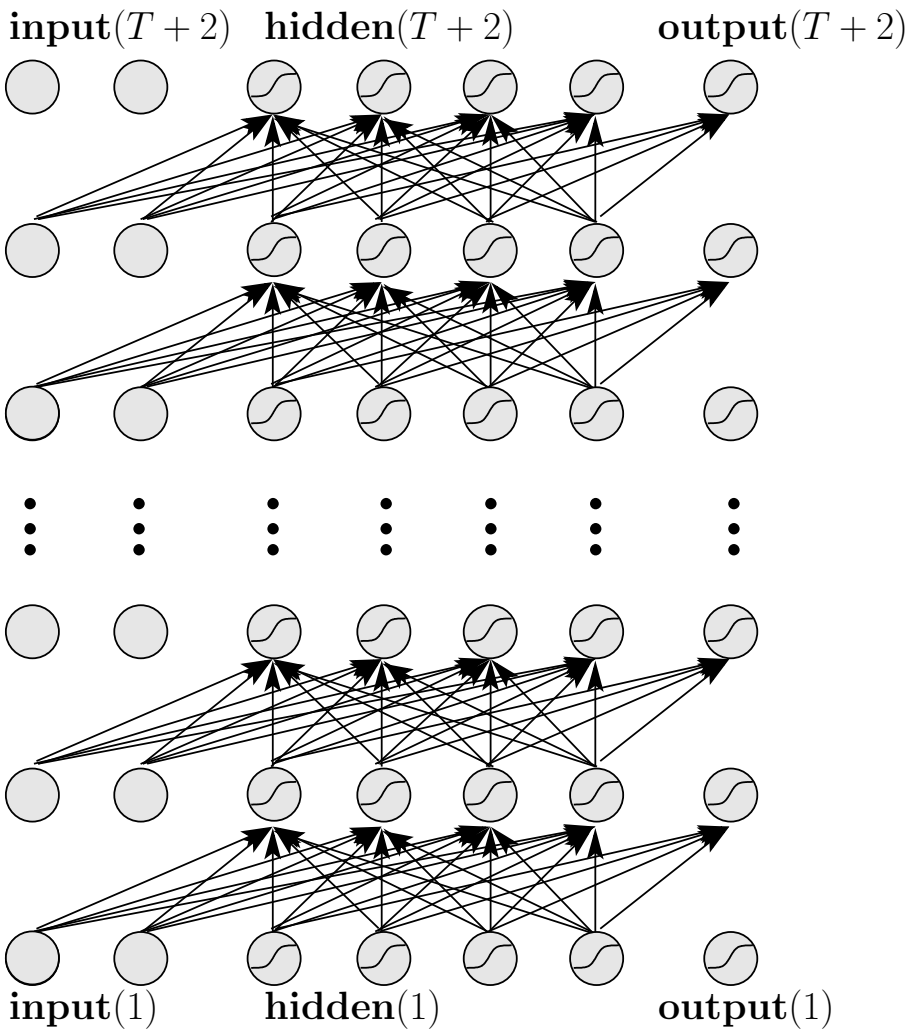


Figure 6.26: The recurrent network from Fig. 6.25 after re-indexing the hidden and output.

The delta error is

$$\delta_j(t) \;=\; -\,\frac{\partial L}{\partial(net_j(t))} = -\,\frac{\partial\left(\sum_{\tau=t}^{T} L(\tau)\right)}{\partial(net_j(t))} \tag{6.123}$$

The back-propagation algorithm starts with

$$\delta_j(T) \;=\; f'(\mathrm{net}_j(T))\,\frac{\partial L(T)}{\partial a_j(T)}\,, \tag{6.124}$$

where in above reformulation T has to be replaced by $(T+2)$.

For $t = T-1$ to $t = 1$:

$$\delta_j(t) \;=\; f'_j(\mathrm{net}_j(t))\,\left(\frac{\partial L(t)}{\partial a_j(t)} \;+\; \sum_l w_{lj}\delta_l(t+1)\right)\,, \tag{6.125}$$

where $\frac{\partial L(t)}{\partial a_j(t)}$ accounts for the immediate error and the sum for the back-propagated error. Both types of error occur if output units have outgoing connections. In out architecture in Fig. 6.26 these errors are separated.

The derivative of the loss with respect to a weight in a layer t of the unfolded network in Fig. 6.26 is

$$-\;\frac{\partial L}{\partial w_{jl}(t)} \;=\; -\,\frac{\partial L}{\partial(\mathrm{net}_j(t))}\,\frac{\mathrm{net}_j(t)}{\partial w_{jl}(t)} \;=\; \delta_j(t)\;a_l(t-1)\;. \tag{6.126}$$

Because the corresponding weights in different layers are identical, the derivative of the loss with respect to a weight is

$$\frac{\partial L}{\partial w_{jl}} \;=\; \sum_{t=1}^{T}\frac{\partial L}{\partial w_{jl}(t)} \;=\; -\sum_{t=1}^{T}\left(\delta_j(t)\;a_l(t-1)\right)\;. \tag{6.127}$$

The on-line weight update is

$$w_{jl}^{\mathrm{new}} \;=\; w_{jl}^{\mathrm{old}} \;-\; \eta\,\frac{\partial L}{\partial w_{jl}}\,, \tag{6.128}$$

where η is the learning rate.

The complexity of BPTT is $O(T\,W)$. BPTT is *local in space* because its complexity per time step and weight is independent of the number of weights.

A special case of BPTT is truncated BPTT called BPTT(n), where only n steps is propagated back. For example $n = 10$ is sufficient because information further back in time cannot be learned to store and to process (see Subsection 6.6.5).

Therefore BPTT is in most cases faster than RTRL without loss of performance.

6.6.4 Other Approaches

We did not consider continuous time networks and did not consider attractor networks like the Hopfield model or Boltzmann machines.

These approaches are currently not relevant for bioinformatics. Continuous time networks may be useful for modeling that is for systems biology.

In the review [Pearlmutter, 1995] a nice overview over recurrent network approaches is given. The first approach were attractor networks for which gradient based methods were developed [Almeida, 1987, Pineda, 1987].

Other recurrent network architectures are

- Networks with context units which use special units, the context units, to store old activations. "Elman networks" [Elman, 1988] use as context units the old activations of hidden units. "Jordan networks" [Jordan, 1986] use as context units old activations of the output units.
- Special context units with time constants allow fast computation or to extract special information in the past. The "focused back-propagation" method [Mozer, 1989] and the method of [Gori et al., 1989] introduce delay factors for the context units and lead to fast learning methods.
- Other networks directly feed the output back for the next time step [Narendra and Parthasarathy, 1990] or NARX networks [Lin et al., 1996].
- Local recurrent networks use only local feedback loops to store information [Frasconi et al., 1992]
- Networks can be build on systems known from control theory like "Finite Impulse Response Filter" (FIR) networks [Wan, 1990] where at the ingoing weights a FIR filter is placed. The architecture in [Back and Tsoi, 1991] uses "Infinite Impulse Response Filter" (IIR) filter instead of FIR filter.
- Other networks use "Auto Regressive Moving Average" (ARMA) units.
- "Time Delay Neural Networks" (TDNNS) [Bodenhausen, 1990, Bodenhausen and Waibel, 1991] use connections with time delays, that means the signal is delayed as it gets transported over the connection. Therefore old inputs can be delayed until they are needed for processing.
- The "Gamma Memory" model [de Vries and Principe, 1991] is a combination of TDNN and time constant based methods.
- Recurrent network training can be based on the (extended) Kalman filter estimation [Matthews, 1990, Williams, 1992, Puskorius and Feldkamp, 1994].

A variant of RTRL is "Teacher Forcing-RTRL" if the output units also have outgoing connections. With "Teacher Forcing-RTRL" the activation of the output units is replaced by the target values.

RTRL and BPTT can be combined to a hybrid algorithm [Schmidhuber, 1992a] which has complexity $O(W\ N)$. Here BPTT is made for N time steps having complexity $O(W\ N)$ whereafter a single RTRL update is made which as complexity $O(W^2/N)\ =\ O(W\ N)$.

6.6.5 Vanishing Gradient

The advantage of recurrent network over feed-forward networks with a window is that they can in principle use all information in the sequence which was so far processed.

However there is problem in storing the relevant information over time. If recurrent networks are not able to store information over time then there great advantage over other approaches is lost.

Consider an error signal $\delta_u(t)$ which is computed at time t at unit u and which gets propagated back over q time steps to unit v. The effect of $\delta_u(t)$ on $\delta_v(t-q)$ can be computed as

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} \ = \ \begin{cases} f'\left(\mathrm{net}_v(t-1)\right)\ w_{uv} & q=1 \\ f'\left(\mathrm{net}_v(t-q)\right)\ \sum_{l=I}^{N} \frac{\partial \delta_l(t-q+1)}{\partial \delta_u(t)} w_{lv} & q>1 \end{cases} . \tag{6.129}$$

If we set $l_q = v$ and $l_0 = u$ then we obtain

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} \ = \ \sum_{l_1=I}^{N} \cdots \sum_{l_{q-1}=I}^{N} \prod_{r=1}^{q} f'\left(\mathrm{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}} . \tag{6.130}$$

Because of

$$\delta_v(t-q) \ = \ -\frac{\partial L}{\partial \mathrm{net}_v(t-q)} \ = \ -\sum_l \frac{\partial L}{\partial \mathrm{net}_l(t)} \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_v(t-q)} \ = \tag{6.131}$$

$$\sum_l \delta_l(t)\ \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_v(t-q)}$$

holds

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} \ = \ \frac{\partial \mathrm{net}_u(t)}{\partial \mathrm{net}_v(t-q)} \tag{6.132}$$

true.

This leads to

$$\frac{\partial L(t)}{\partial w_{ij}(t-q)} \ = \tag{6.133}$$

$$\sum_{l:\text{output unit}} f'(\mathrm{net}_u(t))\ \frac{\partial L(t)}{\partial a_j(t)}\ \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_i(t-q)} \frac{\partial \mathrm{net}_i(t-q)}{\partial w_{ij}(t-q)} \ =$$

$$\sum_{l:\text{output unit}} f'(\mathrm{net}_u(t))\ \frac{\partial L(t)}{\partial a_j(t)}\ \frac{\partial \vartheta_i(t-q)}{\partial \vartheta_l(t)} a_j(t-q)$$

which shows that $\frac{\partial L(t)}{\partial w_{ij}(t-q)}$ is governed by the factor in eq. (6.130).

The eq. (6.130) contains N^{q-1} terms of the form

$$\prod_{r=1}^{q} f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ . \tag{6.134}$$

If the multipliers

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ >\ 1\ , \tag{6.135}$$

the learning is instable because derivatives grow over time and the weight update are too large.

If the multipliers

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ <\ 1\ , \tag{6.136}$$

then we have the case of *vanishing gradient* which means that $\frac{\partial L(t)}{\partial w_{ij}(t-q)}$ decreases with q exponentially to zero.

That means inputs which are far in the past do not influence the current loss and will not be used to improve the network. Therefore the network is not able to learn to store information in the past which can help to reduce the current loss.

6.6.6 Long Short-Term Memory

From previous considerations we know that to avoid both instable learning and the vanishing gradient, we have to enforce

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ =\ 1\ . \tag{6.137}$$

For simplicity we consider only one unit j with a self-recurrent connection w_{jj} and obtain

$$f'\left(\text{net}_{j}(t-r)\right)\ w_{jj}\ =\ 1\ . \tag{6.138}$$

Solving this differential equation gives:

$$f(x)\ =\ \frac{1}{w_{jj}}\ x\ . \tag{6.139}$$

Because f depends on the self-recurrent connection we can set $w_{jj} = 1$ to fix f and obtain

$$f(x)\ =\ x\ . \tag{6.140}$$

Fig. 6.27 shows the single unit which ensures that the vanishing gradient is avoided. Because this unit represents the identity it is immediately clear that information is stored over time through the architecture.

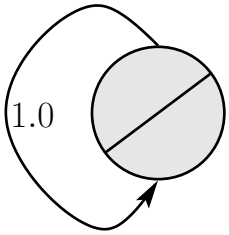


Figure 6.27: A single unit with self-recurrent connection which avoids the vanishing gradient.

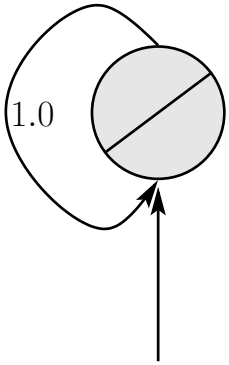


Figure 6.28: A single unit with self-recurrent connection which avoids the vanishing gradient and which has an input.

However it is not enough to avoid the vanishing gradient and therefore ensure keeping of information. A signal must be stored in the unit before it is kept. Fig. 6.28 shows the single unit with an additional input.

However a new problem appears: all information flowing over the input connection is stored. All information which is stored gets superimposed and the single signals cannot be accessed. More serious, not only relevant but also all irrelevant information is stored.

Only relevant information should be stored. In order to realize that a *input gate* a_{in} (basically a sigma-pi unit) is used. If we further assume that the network input net_{c} to the single unit is squashed the we obtain following dynamics:

$$a(t+1) \;=\; a(t) \;+\; a_{\text{in}}(t)\, g\left(\text{net}_{\text{c}}(t)\right) \;, \tag{6.141}$$

If a_{in} is a sigmoid unit active in $[0, b]$ then in can serve as a gating unit.

Now we assume that the output from the storing unit is also squashed by a function h. Further we can control the access to the stored information by an *output gate* a_{out} which is also a sigmoid unit active in $[0, b]$.

The memory access dynamics is

$$a_{\text{c}}(t+1) \;=\; a_{\text{out}}(t)\, h\left(a(t+1)\right) \;. \tag{6.142}$$

The whole dynamics is

$$a_{\text{c}}(t+1) \;=\; a_{\text{out}}(t)\, h\left(a(t) \;+\; a_{\text{in}}(t)\, g\left(\text{net}_{\text{c}}(t)\right)\right) \;. \tag{6.143}$$

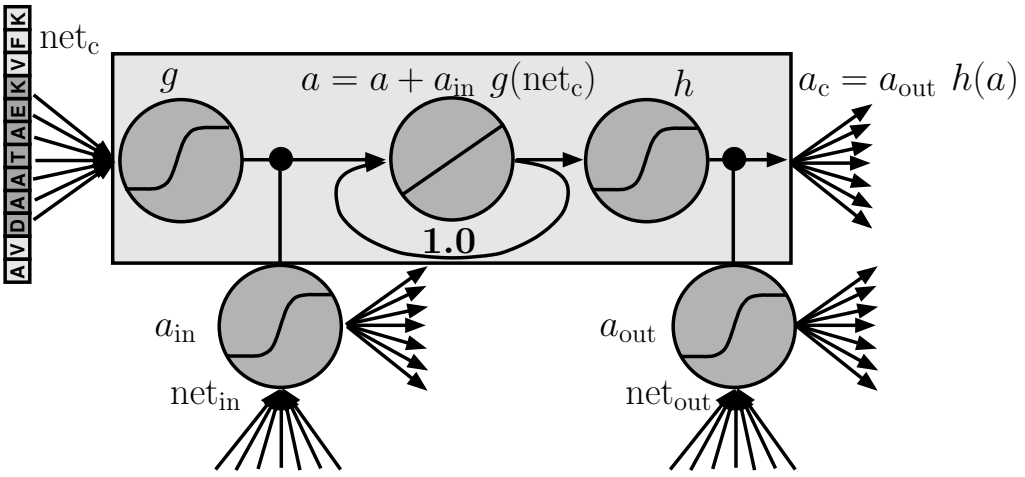


Figure 6.29: The LSTM memory cell. Arrows represent weighted connections of the neural network. Circles represent units (neurons), where the activation function (linear or sigmoid) is indicated in the circle.

The sub-architecture with the gate units a_{in} and a_{out} and the single unit a is called *memory cell* and depicted in Fig. 6.29.

The input to the storing unit is controlled by an "input gating" or attention unit (Fig. 6.29, unit marked "a_{in}") which blocks class-irrelevant information, so that only class-relevant information is stored in memory. The activation of attention units is bounded by 0 and b, i.e. the incoming information $\mathrm{net_c}(t)$ is squashed by a sigmoid function g. The output of the memory cell (Fig. 6.29, center) is bounded by the sigmoid function h (Fig. 6.29, unit labeled as "h"). Memory readout is controlled by an "output gate" (Fig. 6.29, unit labeled "a_{out}"). The cell's output a_{c} is then computed according to eq. (6.142) and eq. (6.143).

The recurrent network build from memory cells is called "Long Short-Term Memory" (LSTM, [Hochreiter and Schmidhuber, 1997b]) is an RNN with a designed memory sub-architecture called "memory cell" to store information.

Memory cells can in principle be integrated into any neural network architecture. The LSTM recurrent network structure as depicted in Fig. 6.30.

The LSTM network is well suited to protein and DNA analysis. For example it excelled in protein classification. Here the input can be not a single amino acid or nucleotide but a whole window over the current position. In this case LSTM is able to learn profiles as depicted in Fig. 6.31.

Using these profiles LSTM was able to generate new motives which were unknown to the PROSITE motif data base.

Currently LSTM is used for alternative splice site detection, nucleosome position detection, protein classification, etc.

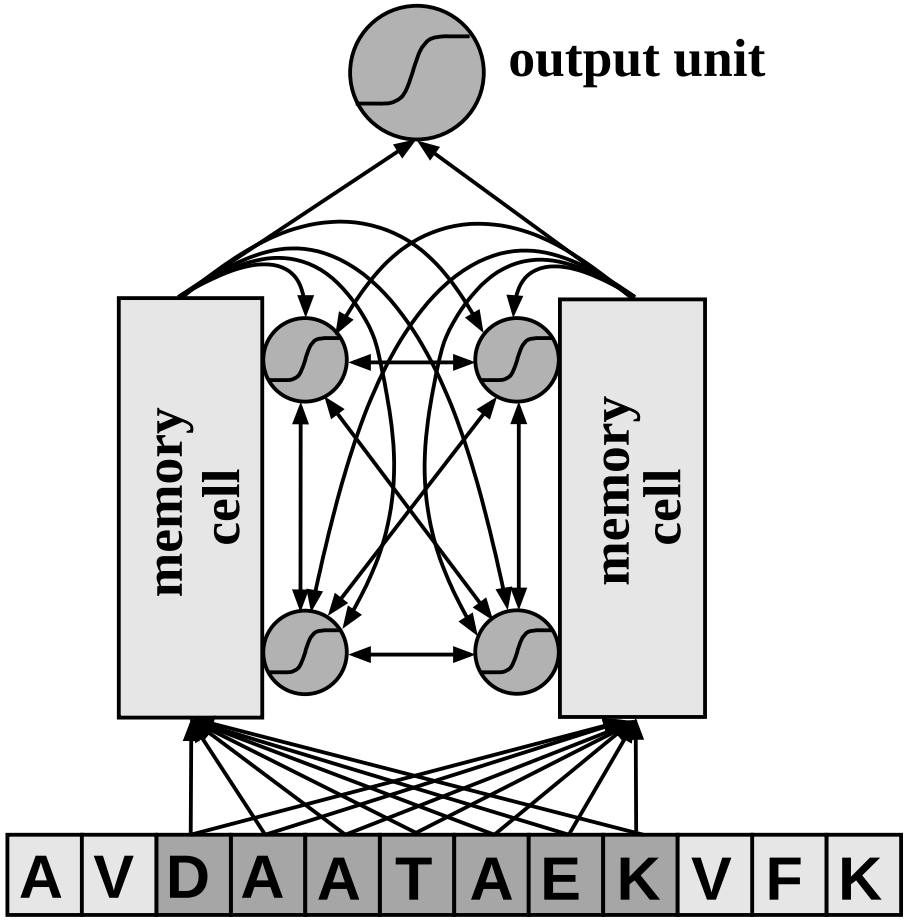


Figure 6.30: LSTM network with three layers: input layer (window of the amino acid sequence – shaded elements), hidden layer (with memory cells – see Fig. 6.29), and output layer. Arrows represent weighted connections; the hidden layer is fully connected. The subnetworks denoted by "memory cell" can store information and receive three kinds of inputs, which act as pattern detectors (input → hidden), as storage control (recurrent connections), and as retrieval control (recurrent connections). The stored information is combined at the output. The amino acid sequence is processed by scanning the sequence step by step from the beginning to the end.

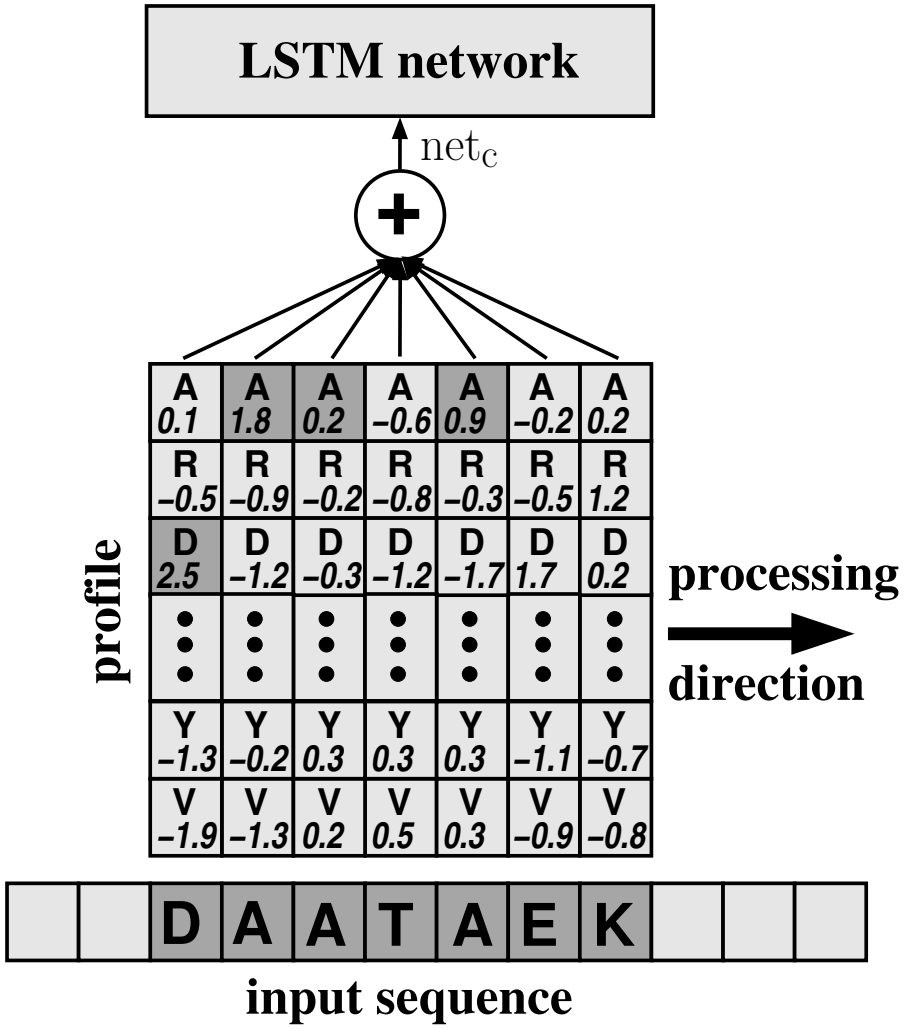


Figure 6.31: A profile as input to the LSTM network which scans the input from left to right. Amino acids are represented by the one letter code. Shaded squares in the matrix match the input and contribute to the sum $\mathrm{net_c}$.

Chapter 7

Bayes Techniques

In this chapter we introduce a probabilistic framework for the empirical error and regularization.

Especially the Bayes framework will be applied to neural networks but it can also be applied to other models.

The Bayes framework gives tools for dealing with the hyper-parameters like the value λ in eq. (6.89) for neural networks or the value C in the optimization problem for the C-SVM eq. (4.32). These hyper-parameters trade-off the empirical error with the complexity term. However an optimal value for these parameters can so far only be found by cross-validation on the training set. The Bayes framework helps to formally treat these parameters. Especially the case if many hyper-parameters are needed then their combination cannot be tested by cross-validation and a formal treatment is necessary.

Another important issue is that Bayes methods allow to introduce error bars and confidence intervals for the model outputs.

Bayes approaches also help to compare quite different models like different neural networks architectures.

Bayes techniques can be used to select relevant features. Feature selection will be discussed in detail in Chapter 8.

Bayes methods can be used to build averages and committees of models.

Summarizing, Bayes techniques allow

- to introduce a probabilistic framework
- to deal with hyper-parameters
- to supply error bars and confidence intervals for the model output
- to compare different models
- to select relevant features
- to make averages and committees.

7.1 Likelihood, Prior, Posterior, Evidence

As in Section 3.2.1 we have the training data $\{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\}$ ($\boldsymbol{z}^i = (\boldsymbol{x}^i, y^i)$), the matrix of feature vectors $\boldsymbol{X} = \left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\right)^T$, the vector of labels $\boldsymbol{y} = \left(y^1, \ldots, y^l\right)^T$, and the training data matrix $\boldsymbol{Z} = \left(\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\right)$. Further we define the training data as

$$\{\boldsymbol{z}\} = \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^l\} . \tag{7.1}$$

In Section 3.4.4 the likelihood $\mathcal{L}$ was defined as

$$\mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) = p(\{\boldsymbol{z}\}; \boldsymbol{w}) , \tag{7.2}$$

i.e. the probability of the model $p(\boldsymbol{z}; \boldsymbol{w})$ to produce the data set. We found that for iid data sampling the likelihood is

$$\mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) = p(\{\boldsymbol{z}\}; \boldsymbol{w}) = \prod_{i=1}^{l} p(\boldsymbol{z}^i; \boldsymbol{w}) . \tag{7.3}$$

In supervised learning we can write

$$p(\boldsymbol{z}; \boldsymbol{w}) = p(\boldsymbol{x})\ p(y \mid \boldsymbol{x}; \boldsymbol{w}) \tag{7.4}$$

and

$$\mathcal{L}(\{\boldsymbol{z}\}; \boldsymbol{w}) = \prod_{i=1}^{l} p(\boldsymbol{x}^i) \prod_{i=1}^{l} p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) . \tag{7.5}$$

Because $\prod_{i=1}^{l} p(\boldsymbol{x}^i)$ is independent of the parameters, it is sufficient to maximize the conditional likelihood

$$\mathcal{L}(\{y\} \mid \{\boldsymbol{x}\}; \boldsymbol{w}) = \prod_{i=1}^{l} p(y^i \mid \boldsymbol{x}^i; \boldsymbol{w}) . \tag{7.6}$$

The likelihood or the negative log-likelihood can be treated as any error term.

For the likelihood in this chapter the parameter vector $\boldsymbol{w}$ is not used to parameterize the likelihood but the likelihood is conditioned on $\boldsymbol{w}$.

The likelihood is

$$p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) . \tag{7.7}$$

However we found that only minimizing the likelihood would lead to overfitting if the model is complex enough. In the most extreme case the model would only produce the training examples

with equal probability and other data with probability zero. That means $p(\boldsymbol{z}; \boldsymbol{w})$ is the sum of Dirac delta-distributions.

To avoid overfitting we can assume that certain $\boldsymbol{w}$ are more probable to be observed in the real world than other. That means some models are more likely in the world.

The fact that some models are more likely can be expressed by a distribution $p(\boldsymbol{w})$, the *prior distribution*. The information in $p(\boldsymbol{w})$ stems from prior knowledge about the problem. This is knowledge without seeing the data, that means we would choose a model according to $p(\boldsymbol{w})$ if we do not have data available.

Now we can use *Bayes formula*:

$$p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \;=\; \frac{p(\{\boldsymbol{z}\} \mid \boldsymbol{w})\, p(\boldsymbol{w})}{p_{\boldsymbol{w}}(\{\boldsymbol{z}\})} \tag{7.8}$$

where

$$p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \tag{7.9}$$

is called the *posterior distribution* and the normalization constant

$$p_{\boldsymbol{w}}(\{\boldsymbol{z}\}) \;=\; \int_W p(\{\boldsymbol{z}\} \mid \boldsymbol{w})\, p(\boldsymbol{w})\, d\boldsymbol{w} \tag{7.10}$$

is called the *evidence* for a class of models parameterized by $\boldsymbol{w}$, however it is also called *accessible volume of the configuration space* (from statistical mechanics), *partition function* (from statistical mechanics), or *error moment generating function*.

Bayes formula is

$$\text{posterior} \;=\; \frac{\text{likelihood} \;\times\; \text{prior}}{\text{evidence}}\,. \tag{7.11}$$

Note that only if $p(\boldsymbol{w})$ is indeed the distribution of model parameters in the real world then

$$p_{\boldsymbol{w}}(\{\boldsymbol{z}\}) \;=\; p(\{\boldsymbol{z}\})\,. \tag{7.12}$$

That means if the real data is indeed produced by first choosing $\boldsymbol{w}$ according to $p(\boldsymbol{w})$ and then generating $\{\boldsymbol{z}\}$ through $p(\{\boldsymbol{z}\} \mid \boldsymbol{w})$ then $p_{\boldsymbol{w}}(\{\boldsymbol{z}\})$ is the probability of observing data $\{\boldsymbol{z}\}$.

However in general the data in real world is not produced according to some mathematical models and therefore $p_{\boldsymbol{w}}(\{\boldsymbol{z}\})$ is not the distribution of occurrence of data $\{\boldsymbol{z}\}$ in the real world.

However $p_{\boldsymbol{w}}(\{\boldsymbol{z}\})$ gives the probability of observing data $\{\boldsymbol{z}\}$ with the model class which is parameterized by $\boldsymbol{w}$.

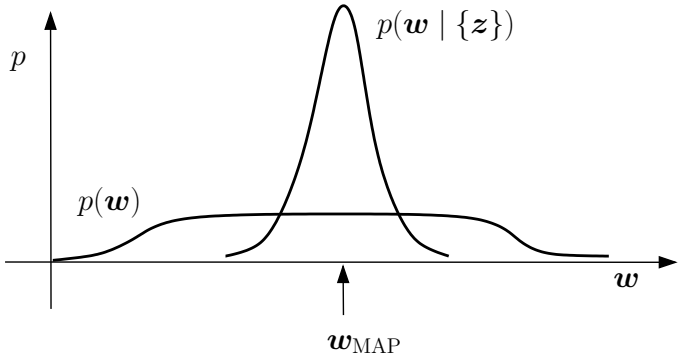


Figure 7.1: The maximum a posteriori estimator $\boldsymbol{w}_{\mathrm{MAP}}$ is the weight vector which maximizes the posterior $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$. The prior distribution $p(\boldsymbol{w})$ is also shown.

7.2 Maximum A Posteriori Approach

The Maximum A Posteriori Approach (MAP) search for the maximal posterior $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ Fig. 7.1 shows the maximum a posteriori estimator $\boldsymbol{w}_{\mathrm{MAP}}$ which maximizes the posterior.

For applying the MAP approach the prior $p(\boldsymbol{w})$ must be defined.

For neural networks we introduced the weight decay method in Section 6.4.5.4, where the simplest term was

$$\Omega(\boldsymbol{w}) \;=\; \|\boldsymbol{w}\|^2 \;=\; \boldsymbol{w}^T\boldsymbol{w} \;=\; \sum_{ij} w_{ij}^2 \,. \tag{7.13}$$

This can be expressed through a Gaussian weight prior

$$\begin{aligned} p(\boldsymbol{w}) \;&=\; \frac{1}{Z_w(\alpha)} \;\exp\left(-\frac{1}{2}\,\alpha\,\|\boldsymbol{w}\|^2\right) \\ Z_w(\alpha) \;&=\; \int_W \exp\left(-\frac{1}{2}\,\alpha\,\|\boldsymbol{w}\|^2\right)\,d\boldsymbol{w} \;=\; \left(\frac{2\,\pi}{\alpha}\right)^{W/2} \,. \end{aligned} \tag{7.14}$$

The parameter α is a hyper-parameter which trades in the $\log$-posterior the error term against the complexity term and is here correlated with the allowed variance of the weights.

The other weight decay terms in Section 6.4.5.4 give either a Laplace distribution ($\Omega(\boldsymbol{w}) \;=\; \|\boldsymbol{w}\|_1$)

$$\begin{aligned} p(\boldsymbol{w}) \;&=\; \frac{1}{Z_w(\alpha)} \;\exp\left(-\frac{1}{2}\,\alpha\,\|\boldsymbol{w}\|_1\right) \\ Z_w(\alpha) \;&=\; \int_W \exp\left(-\frac{1}{2}\,\alpha\,\|\boldsymbol{w}\|_1\right)\,d\boldsymbol{w} \end{aligned} \tag{7.15}$$

or for compact weight sets a Cauchy distribution ($\Omega(\boldsymbol{w}) \; = \; \log\left(1 \; + \; \|\boldsymbol{w}\|^2\right)$):

$$
\begin{aligned}
p(\boldsymbol{w}) \; &= \; \frac{1}{Z_w(\alpha)} \; \exp\left(-\frac{1}{2}\,\alpha\right) \; \left(1 \; + \; \|\boldsymbol{w}\|^2\right) \\
Z_w(\alpha) \; &= \; \int_W \; \exp\left(-\frac{1}{2}\,\alpha\right) \; \left(1 \; + \; \|\boldsymbol{w}\|^2\right) d\boldsymbol{w} \; .
\end{aligned}
\tag{7.16}
$$

For Gaussian noise models from Section 3.5.1 we have

$$
\begin{aligned}
&p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \; = \\
&\frac{1}{(2\,\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \; \exp\left(-\frac{1}{2}\,(\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})\right) \; p(\{\boldsymbol{x}\})
\end{aligned}
\tag{7.17}
$$

and for $\boldsymbol{\Sigma} \; = \; \sigma^2\,\boldsymbol{I}$

$$
\begin{aligned}
&p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \; = \\
&\frac{1}{(2\,\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \; \exp\left(-\frac{1}{2\,\sigma^2}\,(\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})^T\,(\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})\right) \; p(\{\boldsymbol{x}\}) \; .
\end{aligned}
\tag{7.18}
$$

The term $R_{\mathrm{emp}} \; = \; (\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})^T\,(\boldsymbol{y} \; - \; \boldsymbol{X}\,\boldsymbol{w})$ is only the mean squared error.

The negative log-posterior is

$$
-\; \log p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \; = \; -\; \log p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \; - \; \log p(\boldsymbol{w}) \; + \; \log p_{\boldsymbol{w}}(\{\boldsymbol{z}\}) \; ,
\tag{7.19}
$$

where $p_{\boldsymbol{w}}(\{\boldsymbol{z}\})$ does not depend on $\boldsymbol{w}$.

For maximum a posteriori estimation only $-\; \log p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \; - \; \log p(\boldsymbol{w})$ must be minimized which results in the terms

$$
\tilde{R}(\boldsymbol{w}) \; = \; \frac{1}{2\,\sigma^2} R_{\mathrm{emp}} \; + \; \frac{1}{2}\alpha\,\Omega(\boldsymbol{w}) \; = \; \frac{1}{2}\,\beta\,R_{\mathrm{emp}} \; + \; \frac{1}{2}\,\alpha\,\Omega(\boldsymbol{w}) \; ,
\tag{7.20}
$$

where $\beta^{-1} = \sigma^2$. If we set $R(\boldsymbol{w}) \; = \; \tilde{R}(\boldsymbol{w})\;2\;\sigma^2$ and setting $\lambda \; = \; \sigma^2\;\alpha$ we have to minimize

$$
R(\boldsymbol{w}) \; = \; R_{\mathrm{emp}} \; + \; \lambda\,\Omega(\boldsymbol{w}) \; .
\tag{7.21}
$$

This is exactly eq. (6.89).

Therefore minimizing error terms consisting of the empirical error plus a complexity term can be viewed in most cases as maximum a posteriori estimation.

Note that the likelihood is the exponential function with empirical error as argument

$$
p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \; = \; \frac{1}{Z_R(\beta)} \exp\left(-\frac{1}{2}\,\beta\,R_{\mathrm{emp}}\right)
\tag{7.22}
$$

and the prior is an exponential function of the complexity

$$p(\boldsymbol{w}) \;=\; \frac{1}{Z_w(\alpha)} \exp\left(-\frac{1}{2}\,\alpha\,\Omega(\boldsymbol{w})\right) \tag{7.23}$$

and the posterior is

$$p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \;=\; \frac{1}{Z(\alpha,\beta)} \exp\left(-\frac{1}{2}\,(\alpha\,\Omega(\boldsymbol{w}) \;+\; \beta\,R_{\mathrm{emp}})\right)\;, \tag{7.24}$$

where

$$Z(\alpha,\beta) \;=\; \int_W \exp\left(-\frac{1}{2}\,(\alpha\,\Omega(\boldsymbol{w}) \;+\; \beta\,R_{\mathrm{emp}})\right)\,d\boldsymbol{w}\;. \tag{7.25}$$

7.3 Posterior Approximation

In order to approximate the posterior a Gaussian assumption is made.

First we make a Taylor expansion of $R(\boldsymbol{w})$ around its minimum $\boldsymbol{w}_{\mathrm{MAP}}$:

$$\tilde{R}(\boldsymbol{w}) \;=\; \tilde{R}(\boldsymbol{w}_{\mathrm{MAP}}) \;+\; \frac{1}{2}\;(\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})^T\,\boldsymbol{H}\;(\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})\;, \tag{7.26}$$

where the first order derivatives vanish at the minimum and $\boldsymbol{H}$ is the Hessian of $\tilde{R}(\boldsymbol{w})$ at $\boldsymbol{w}_{\mathrm{MAP}}$.

The posterior is now a Gaussian

$$\begin{aligned} &p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \;=\; \frac{1}{Z}\;\exp(-\,\tilde{R}(\boldsymbol{w})) \;= \\ &\frac{1}{Z}\;\exp\left(-\,\tilde{R}(\boldsymbol{w}_{\mathrm{MAP}}) \;-\; \frac{1}{2}\;(\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})^T\,\boldsymbol{H}\;(\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})\right)\;, \end{aligned} \tag{7.27}$$

where Z is normalization constant.

The Hessian for weight decay given by

$$\boldsymbol{H} \;=\; \frac{1}{\sigma^2}\,\boldsymbol{H}_{\mathrm{emp}} \;+\; \alpha\,\boldsymbol{I} \;=\; \beta\,\boldsymbol{H}_{\mathrm{emp}} \;+\; \alpha\,\boldsymbol{I}\;, \tag{7.28}$$

where $\boldsymbol{H}_{\mathrm{emp}}$ is the Hessian of the empirical error and can for neural networks be computed as described in Section 6.4.4.

The normalization constant is

$$Z(\alpha,\beta) \;=\; \exp\left(\tilde{R}(\boldsymbol{w}_{\mathrm{MAP}})\right)\;(2\,\pi)^{-W/2}\;|\boldsymbol{H}|^{-1/2}\;. \tag{7.29}$$

7.4 Error Bars and Confidence Intervals

We now want to derive confidence intervals for model outputs. A user is not only interested in the best prediction but also wants to know how reliable the prediction is.

The distribution for the outputs is

$$p(y \mid \boldsymbol{x}, \{\boldsymbol{z}\}) \;=\; \int_W p(y \mid \boldsymbol{x}, \boldsymbol{w})\; p(\boldsymbol{w} \mid \{\boldsymbol{z}\})\; d\boldsymbol{w}\;, \tag{7.30}$$

where we used the posterior distribution $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ and a noise model $p(y \mid \boldsymbol{x}, \boldsymbol{w})$.

The Gaussian noise model for one dimension is

$$p(y \mid \boldsymbol{x}, \boldsymbol{w}) \;=\; \frac{1}{Z_R(\beta)}\; \exp\left(-\frac{\beta}{2}\; (g(\boldsymbol{x}; \boldsymbol{w}) \;-\; y)^2\right)\;, \tag{7.31}$$

where we again used $\beta \;=\; \frac{1}{\sigma^2}$ and

$$Z_R(\beta) \;=\; \left(\frac{2\,\pi}{\beta}\right)^{l/2}\;. \tag{7.32}$$

We now approximate $g(\boldsymbol{x}; \boldsymbol{w})$ linearly around $\boldsymbol{w}_{\mathrm{MAP}}$:

$$g(\boldsymbol{x}; \boldsymbol{w}) \;=\; g(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{MAP}}) \;+\; \boldsymbol{g}^T\; (\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})\;, \tag{7.33}$$

where $\boldsymbol{g}$ is the gradient of $g(\boldsymbol{x}; \boldsymbol{w})$ evaluated at $\boldsymbol{w}_{\mathrm{MAP}}$. This approximation together with the approximation for the posterior gives us

$$p(y \mid \boldsymbol{x}, \{\boldsymbol{z}\}) \;\propto\; \int_W \exp\Bigg(-\frac{\beta}{2}\left(y \;-\; g(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{MAP}}) \;+\; \boldsymbol{g}^T\; (\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})\right)^2 - \frac{1}{2}\; (\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})^T\, \boldsymbol{H}\; (\boldsymbol{w} \;-\; \boldsymbol{w}_{\mathrm{MAP}})\Bigg)\, d\boldsymbol{w}\;. \tag{7.34}$$

This integral can be computed and results in

$$p(y \mid \boldsymbol{x}, \{\boldsymbol{z}\}) \;=\; \frac{1}{\sqrt{2\,\pi}\;\sigma_y}\; \exp\left(-\frac{1}{2\;\sigma_y^2}\; (y \;-\; g(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{MAP}}))^2\right)\;, \tag{7.35}$$

where

$$\sigma_y^2 \;=\; \frac{1}{\beta} \;+\; \boldsymbol{g}^T \boldsymbol{H}^{-1}\, \boldsymbol{g} \;=\; \sigma^2 \;+\; \boldsymbol{g}^T \boldsymbol{H}^{-1}\, \boldsymbol{g}\;. \tag{7.36}$$

The output variance is the inherent data noise variance σ^2 plus the approximation uncertainty $\boldsymbol{g}^T \boldsymbol{H}^{-1}\, \boldsymbol{g}$.

Fig. 7.2 shows error bars which are large because of a high inherent error $\beta^{-1} = \sigma^2$. Fig. 7.3 shows error bars which are large because of a not very precisely chosen $\boldsymbol{w}_{\mathrm{MAP}}$. The later can be if few training data are available or if the data contradicts the prior.

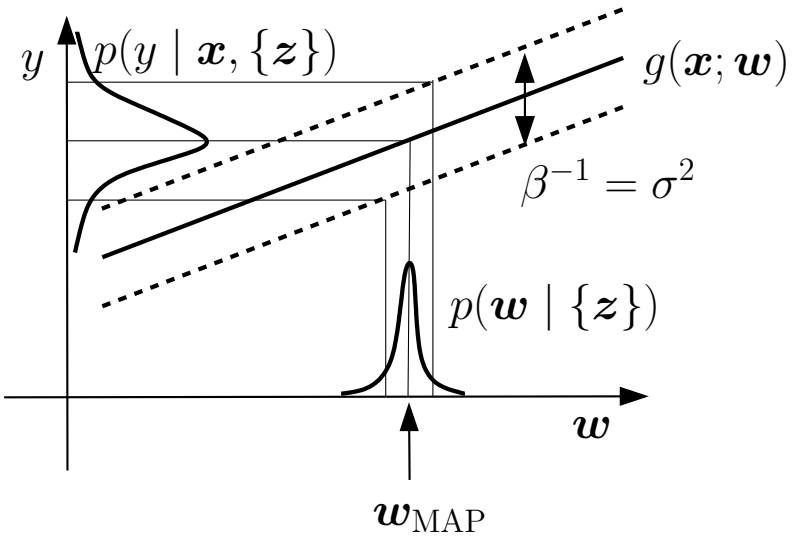


Figure 7.2: Error bars obtained by Bayes technique. On the y-axis the error bars are given as quantiles of the distribution $p(y \mid \boldsymbol{x}; \{\boldsymbol{z}\})$. The large error bars result from the high inherent error $\beta^{-1} = \sigma^2$ of the data. The parameter $\boldsymbol{w}_{\mathrm{MAP}}$ has been chosen very precisely (e.g. if many training data points were available).

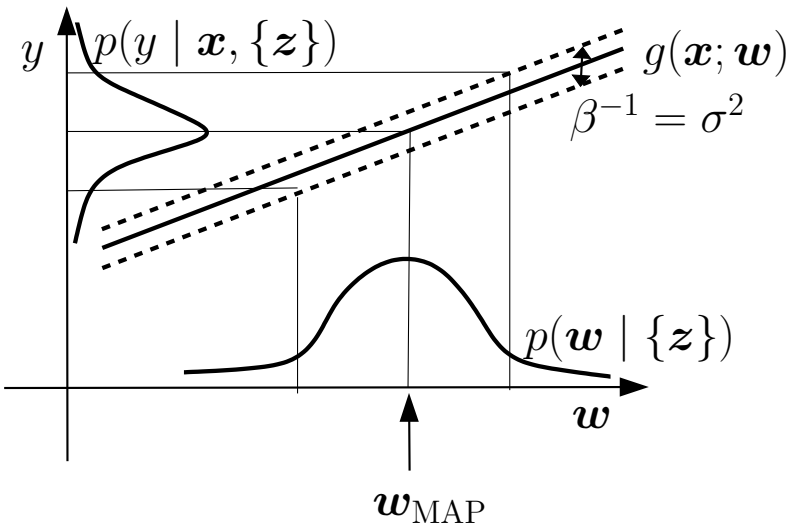


Figure 7.3: Error bars obtained by Bayes technique. On the y-axis the error bars are given as quantiles of the distribution $p(y \mid \boldsymbol{x}; \{\boldsymbol{z}\})$. The large error bars result from the broad posterior, that means the parameter $\boldsymbol{w}_{\mathrm{MAP}}$ has not been chosen very precisely (few data points or prior and data were not compatible).

7.5 Hyper-parameter Selection: Evidence Framework

We are focusing on the hyper-parameters α and β from the objective eq. (7.20). β is an assumption on the noise in the data. α is an assumption on the optimal network complexity relative to the empirical error.

The posterior can be expressed by integrating out α and β which is called *marginalization*:

$$\begin{aligned} & p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \; = \\ & \int_{S_\alpha} \int_{S_\beta} p(\boldsymbol{w}, \alpha, \beta \mid \{\boldsymbol{z}\}) \; d\alpha \; d\beta \; = \\ & \int_{S_\alpha} \int_{S_\beta} p(\boldsymbol{w} \mid \alpha, \beta, \{\boldsymbol{z}\}) \; p(\alpha, \beta \mid \{\boldsymbol{z}\}) \; d\alpha \; d\beta \; . \end{aligned} \tag{7.37}$$

To compute the integrals will be considered in Section 7.6.

Here we first consider to approximate the posterior. We assume that the posterior $p(\alpha, \beta \mid \{\boldsymbol{z}\})$ is sharply peaked around the maximal values α_{MAP} and β_{MAP}. That means around high values of $p(\alpha, \beta \mid \{\boldsymbol{z}\})$ the $p(\boldsymbol{w} \mid \alpha, \beta, \{\boldsymbol{z}\})$ is constant $p(\boldsymbol{w} \mid \alpha_{\mathrm{MAP}}, \beta_{\mathrm{MAP}}, \{\boldsymbol{z}\})$. We obtain

$$\begin{aligned} & p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \; = \; p(\boldsymbol{w} \mid \alpha_{\mathrm{MAP}}, \beta_{\mathrm{MAP}}, \{\boldsymbol{z}\}) \int_{S_\alpha} \int_{S_\beta} p(\alpha, \beta \mid \{\boldsymbol{z}\}) \; d\alpha \; d\beta \; = \\ & p(\boldsymbol{w} \mid \alpha_{\mathrm{MAP}}, \beta_{\mathrm{MAP}}, \{\boldsymbol{z}\}) \; . \end{aligned} \tag{7.38}$$

Using this approximation we are searching for the hyper-parameters which maximize the posterior. We will try to express the posterior with the variables α and β and then to search for the variables which maximize the posterior.

The posterior of α and β is

$$p(\alpha, \beta \mid \{\boldsymbol{z}\}) \; = \; \frac{p(\{\boldsymbol{z}\} \mid \alpha, \beta) \; p(\alpha, \beta)}{p_{\alpha,\beta}(\{\boldsymbol{z}\})} \; . \tag{7.39}$$

Here the prior for α and β, $p(\alpha, \beta)$ must be chosen. For example *non-informative priors* which give equal probability to all values are a popular choice.

Note that from objective eq. (7.20). we see that $\beta = \frac{1}{\sigma^2}$ is only present in the $\boldsymbol{w}$-likelihood because it determines the noise in the data. In contrast α is only present in $\boldsymbol{w}$-prior as a weighting factor for the $\boldsymbol{w}$-prior which scales the complexity against the error. We express the (α, β)-likelihood through marginalization over $\boldsymbol{w}$:

$$\begin{aligned} & p(\{\boldsymbol{z}\} \mid \alpha, \beta) \; = \; \int_W p(\{\boldsymbol{z}\} \mid \boldsymbol{w}, \alpha, \beta) \; p(\boldsymbol{w} \mid \alpha, \beta) \; d\boldsymbol{w} \; = \\ & \int_W p(\{\boldsymbol{z}\} \mid \boldsymbol{w}, \beta) \; p(\boldsymbol{w} \mid \alpha) \; d\boldsymbol{w} \; . \end{aligned} \tag{7.40}$$

Using eq. (7.22), eq. (7.23), eq. (7.24), and eq. (7.25) we obtain

$$p(\{\boldsymbol{z}\} \mid \alpha, \beta) \; = \; \frac{Z(\alpha, \beta)}{Z_w(\alpha) \; Z_R(\beta)} \; . \tag{7.41}$$

Especially,

$$p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \;=\; \frac{Z(\alpha,\beta)}{Z_R(\beta)\; Z_w(\alpha)}\; p(\{\boldsymbol{z}\} \mid \boldsymbol{w})\; p(\boldsymbol{w})\;. \tag{7.42}$$

Example.

We will show this on an example with concrete empirical error and prior term.

For example if we use the mean squared error and as regularization a weight decay term we already computed $Z(\alpha,\beta)$ in eq. (7.29) as

$$Z(\alpha,\beta) \;=\; \exp\Big(-\;\tilde{R}(\boldsymbol{w}_{\rm MAP})\Big)\;(2\,\pi)^{W/2}\;|\boldsymbol{H}|^{-1/2}\;, \tag{7.43}$$

where

$$\tilde{R}(\boldsymbol{w}_{\rm MAP}) \;=\; \frac{1}{2}\,\beta R_{\rm emp} \;+\; \frac{1}{2}\alpha\,\Omega(\boldsymbol{w})\;. \tag{7.44}$$

According to eq. (7.32)

$$Z_R(\beta) \;=\; \left(\frac{2\,\pi}{\beta}\right)^{l/2} \tag{7.45}$$

and according to eq. (7.14)

$$Z_w(\beta) \;=\; \left(\frac{2\,\pi}{\alpha}\right)^{W/2}\;. \tag{7.46}$$

$$\begin{aligned}\ln p(\{\boldsymbol{z}\} \mid \alpha,\beta) \;=\; &-\,\alpha\,\Omega(\boldsymbol{w}_{\rm MAP}) \;-\; \beta\,R_{\rm emp} \;-\; \frac{1}{2}\;\ln|\boldsymbol{H}| \;+\\ &\frac{W}{2}\;\ln\alpha \;+\; \frac{l}{2}\;\ln\beta \;-\; \frac{W+l}{2}\ln(2\pi)\;,\end{aligned} \tag{7.47}$$

where according to eq. (7.28)

$$\boldsymbol{H} \;=\; \beta\,\boldsymbol{H}_{\rm emp} \;+\; \alpha\boldsymbol{I}\;. \tag{7.48}$$

Assume we already computed the eigenvalues λ_j of $\boldsymbol{H}_{\rm emp}$ then

$$\begin{aligned}&\frac{\partial}{\partial\alpha}\ln|\boldsymbol{H}| \;=\; \frac{\partial}{\partial\alpha}\ln\prod_{j=1}^{W}(\beta\,\lambda_j \;+\; \alpha) \;=\\ &\frac{\partial}{\partial\alpha}\sum_{j=1}^{W}\ln(\beta\,\lambda_j \;+\; \alpha) \;=\; \sum_{j=1}^{W}\frac{1}{\beta\,\lambda_j \;+\; \alpha} \;=\; {\rm Tr}\boldsymbol{H}^{-1}\;,\end{aligned} \tag{7.49}$$

where we assumed that λ_j do not depend on α. However the Hessian $\boldsymbol{H}$ was evaluated at $\boldsymbol{w}_{\mathrm{MAP}}$ which depends on α, therefore terms in $\frac{\partial \lambda_j}{\partial \alpha}$ were neglected.

Setting the derivative of the negative log-posterior (for α and β) with respect to α to zero gives

$$\begin{aligned} &\frac{\partial}{\partial \alpha} \ln p(\{\boldsymbol{z}\} \mid \alpha, \beta) \; = \\ &- \Omega(\boldsymbol{w}_{\mathrm{MAP}}) \; - \; \frac{1}{2} \sum_{j=1}^{W} \frac{1}{\beta \, \lambda_j \; + \; \alpha} \; + \; \frac{1}{2} \, W \, \frac{1}{\alpha} \; = \; 0 \, , \end{aligned} \tag{7.50}$$

which gives

$$\begin{aligned} &2 \, \alpha \, \Omega(\boldsymbol{w}_{\mathrm{MAP}}) \; = \\ &-\sum_{j=1}^{W} \frac{\alpha}{\beta \, \lambda_j \; + \; \alpha} \; + \; W \;\; = \sum_{j=1}^{W} \frac{\beta \, \lambda_j}{\beta \, \lambda_j \; + \; \alpha} \; = \; \gamma \, . \end{aligned} \tag{7.51}$$

If $\Omega(\boldsymbol{w}_{\mathrm{MAP}}) \; = \; 0$ then the weight vector is zero, so $\Omega(\boldsymbol{w}_{\mathrm{MAP}})$ shows how far the weights are pushed away from their prior value of zero by the data.

The term $\frac{\beta \, \lambda_j}{\beta \, \lambda_j \; + \; \alpha}$ is in $[0; 1]$ and if it is close to one then the data governs this term and terms close to zero a driven by the prior.

The term γ measures the effective number of weights which are driven by the data.

Note, however that the Hessian is not evaluated at the minimum of R_{emp} but at $\boldsymbol{w}_{\mathrm{MAP}}$, therefore the eigenvalues λ_j of $\boldsymbol{H}_{\mathrm{emp}}$ are not guaranteed to be positive. Therefore terms $\frac{\beta \, \lambda_j}{\beta \, \lambda_j \; + \; \alpha}$ may be negative because $(\beta \, \lambda_j \; + \; \alpha)$ is positive.

Now we compute the derivative of the negative log-posterior (for α and β) with respect to β.

The derivative of the log of the absolute Hessian determinant with respect to β is

$$\begin{aligned} &\frac{\partial}{\partial \beta} \ln |\boldsymbol{H}| \; = \; \frac{\partial}{\partial \beta} \sum_{j=1}^{W} \ln \left(\beta \, \lambda_j \; + \; \alpha \right) \; = \\ &\sum_{j=1}^{W} \frac{\lambda_j}{\beta \, \lambda_j \; + \; \alpha} \, . \end{aligned} \tag{7.52}$$

Setting the derivative of the negative log-posterior with respect to β to zero gives

$$2 \, \beta \, R_{\mathrm{emp}} \; = \; l \; - \; \sum_{j=1}^{W} \frac{\lambda_j}{\beta \, \lambda_j \; + \; \alpha} \; = \; l \; - \; \gamma \, . \tag{7.53}$$

The updates for the hyper-parameters are

$$\begin{aligned} \alpha^{\mathrm{new}} \; &= \; \frac{\gamma}{2 \, \Omega(\boldsymbol{w}_{\mathrm{MAP}})} \\ \beta^{\mathrm{new}} \; &= \; \frac{l \; - \; \gamma}{2 \, R_{\mathrm{emp}}(\boldsymbol{w}_{\mathrm{MAP}})} \, . \end{aligned} \tag{7.54}$$

Now with these new hyper-parameters the new values of $\boldsymbol{w}_{\mathrm{MAP}}$ can be estimated through gradient based methods. Then again the hyper-parameters α and β can be updated and so forth.

If all parameters are well defined $\gamma = W$ and if much more training examples than weights are present $l >> W$ then one can use as an approximation for the update formulae

$$\begin{aligned} \alpha^{\mathrm{new}} &= \frac{W}{2\,\Omega(\boldsymbol{w}_{\mathrm{MAP}})} \\ \beta^{\mathrm{new}} &= \frac{l}{2\,R_{\mathrm{emp}}(\boldsymbol{w}_{\mathrm{MAP}})}\,. \end{aligned} \tag{7.55}$$

7.6 Hyper-parameter Selection: Integrate Out

In previous section we started with the posterior which was obtained by integrating out α and β.

$$\begin{aligned} &p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) = \int_{S_\alpha}\int_{S_\beta} p(\boldsymbol{w},\alpha,\beta \mid \{\boldsymbol{z}\})\,d\alpha\,d\beta = \\ &\int_{S_\alpha}\int_{S_\beta} p(\boldsymbol{w} \mid \alpha,\beta,\{\boldsymbol{z}\})\,p(\alpha,\beta \mid \{\boldsymbol{z}\})\,d\alpha\,d\beta = \\ &\frac{1}{p_w(\{\boldsymbol{z}\})}\int_{S_\alpha}\int_{S_\beta} p(\{\boldsymbol{z}\} \mid \boldsymbol{w},\beta)\,p(\boldsymbol{w} \mid \alpha)\,p(\alpha)\,p(\beta)\,d\alpha\,d\beta\,. \end{aligned} \tag{7.56}$$

Here we used that the hyper-parameters α and β are independent from one another do not depend on the data: $p(\alpha,\beta \mid \{\boldsymbol{z}\}) = p(\alpha)\,p(\beta)$. The $\boldsymbol{w}$-posterior $p(\boldsymbol{w} \mid \alpha,\beta,\{\boldsymbol{z}\})$ was expressed through the Bayes formula

$$p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) = \frac{p(\{\boldsymbol{z}\} \mid \boldsymbol{w})\,p(\boldsymbol{w})}{p_{\boldsymbol{w}}(\{\boldsymbol{z}\})} \tag{7.57}$$

and then the hyper-parameters are removed from densities where the variable is independent from the hyper-parameter . That is $p(\{\boldsymbol{z}\} \mid \boldsymbol{w},\alpha,\beta) = p(\{\boldsymbol{z}\} \mid \boldsymbol{w},\beta)$ and $p(\boldsymbol{w} \mid \alpha,\beta) = p(\boldsymbol{w} \mid \alpha)$.

The parameters α and β are scaling parameters. If target and output range is increased then β should re-scale the empirical error.

Similar hold for weight scaling. If the activation functions say $\frac{1}{1+\exp(-\rho\ \mathrm{net}}$ change their slopes ρ then the same network functions are obtained by re-scaling the weights and, therefore, net. That means different weight ranges may implement the same function.

Such parameter as the standard deviation σ for the Gaussians are scale parameters. For scale parameters the prior is often chosen to be non-informative (uniformly) on a logarithmic scale, that is $p(\ln(\alpha))$ and $p(\ln(\beta))$ are constant.

From this follows (note, $p_x(\boldsymbol{x}) = p_g(g(\boldsymbol{x}))\,\left|\frac{\partial g}{\partial x}\right|$):

$$p(\alpha) = \frac{1}{\alpha} \tag{7.58}$$

$$p(\beta) = \frac{1}{\beta}\,. \tag{7.59}$$

We first consider the prior over the weights

$$p(\boldsymbol{w}) \;=\; \int_0^\infty p(\boldsymbol{w} \mid \alpha)\, p(\alpha)\, d\alpha \;= \tag{7.60}$$
$$\int_0^\infty \frac{1}{Z_w(\alpha)}\, \exp(-\,\alpha\, \Omega(\boldsymbol{w}))\, \frac{1}{\alpha}\, d\alpha \;=$$
$$(2\,\pi)^{-W/2} \int_0^\infty \exp(-\,\alpha\, \Omega(\boldsymbol{w}))\, \alpha^{W/2-1}\, d\alpha \;=\; \frac{\Gamma(W/2)}{(2\,\pi\,\Omega(\boldsymbol{w}))^{W/2}}\;,$$

where Γ is the gamma function.

Analog we obtain

$$p(\{\boldsymbol{z}\} \mid \boldsymbol{w}) \;=\; \frac{\Gamma(l/2)}{(2\,\pi\,R_{\mathrm{emp}})^{l/2}}\;. \tag{7.61}$$

From these two values and the Bayes formula we can compute the negative log-posterior as

$$-\;\ln p(\boldsymbol{w} \mid \{\boldsymbol{z}\}) \;=\; \frac{l}{2}\, R_{\mathrm{emp}} \;+\; \frac{W}{2}\, \Omega(\boldsymbol{w}) \;+\; \mathrm{const}\;. \tag{7.62}$$

Comparing last equation with eq. (7.24) and noting that we used $\boldsymbol{w}_{\mathrm{MAP}}$, gives the update rules from eq. (7.55):

$$\alpha^{\mathrm{new}} \;=\; \frac{W}{2\,\Omega(\boldsymbol{w}_{\mathrm{MAP}})} \tag{7.63}$$
$$\beta^{\mathrm{new}} \;=\; \frac{l}{2\,R_{\mathrm{emp}}(\boldsymbol{w}_{\mathrm{MAP}})}\;.$$

Again an iterative methods first uses the actual α and β to find $\boldsymbol{w}_{\mathrm{MAP}}$ through gradient descent. And then α and β are updated whereafter again the new $\boldsymbol{w}_{\mathrm{MAP}}$ is estimated and so on.

7.7 Model Comparison

Using the Bayes formula we can compare model classes $\mathcal{M}$.

Bayes formula gives

$$p(\mathcal{M} \mid \{\boldsymbol{z}\}) \;=\; \frac{p(\{\boldsymbol{z}\} \mid \mathcal{M})\, p(\mathcal{M})}{p_{\mathcal{M}}(\{\boldsymbol{z}\})}\;, \tag{7.64}$$

where $p(\{\boldsymbol{z}\} \mid \mathcal{M})$ is here the likelihood of the data given a model class but is at the same time the evidence introduced in Section 7.1 for model selection.

The evidence for model selection was defined in eq. (7.10) as

$$p(\{\boldsymbol{z}\} \mid \mathcal{M}) \;=\; \int_W p(\{\boldsymbol{z}\} \mid \boldsymbol{w}, \mathcal{M})\, p(\boldsymbol{w} \mid \mathcal{M})\, d\boldsymbol{w}\;, \tag{7.65}$$

where we only made all probabilities conditioned on the model class $\mathcal{M}$.

If the posterior $p(\boldsymbol{w} \mid \{\boldsymbol{z}\}, \mathcal{M})$ (or according to the Bayes formula equivalently $p(\{\boldsymbol{z}\} \mid \boldsymbol{w}, \mathcal{M})\ p(\boldsymbol{w} \mid \mathcal{M})$) is peaked in weight space then we can approximate the posterior by a box around the maximum a posteriori value $\boldsymbol{w}_{\mathrm{MAP}}$:

$$p(\{\boldsymbol{z}\} \mid \mathcal{M}) \approx p(\{\boldsymbol{z}\} \mid \boldsymbol{w}_{\mathrm{MAP}}, \mathcal{M})\, p(\boldsymbol{w}_{\mathrm{MAP}} \mid \mathcal{M})\, \Delta \boldsymbol{w}_{\mathrm{MAP}} \,. \tag{7.66}$$

If we assume a Gaussian distribution of the posterior then $\Delta \boldsymbol{w}_{\mathrm{MAP}}$ can be estimated from the Hessian $\boldsymbol{H}$.

We can also use the eq. (7.47)

$$\begin{aligned} \ln p(\{\boldsymbol{z}\} \mid \alpha, \beta) \;=\; & -\,\alpha\, \Omega(\boldsymbol{w}_{\mathrm{MAP}}) \;-\; \beta\, R_{\mathrm{emp}} \;-\; \frac{1}{2}\, \ln |\boldsymbol{H}| \;+ \\ & \frac{W}{2}\, \ln \alpha \;+\; \frac{l}{2}\, \ln \beta \;-\; \frac{W+l}{2} \ln(2\pi) \,, \end{aligned} \tag{7.67}$$

where we insert α_{MAP} and β_{MAP} for α and β.

It can be shown (e.g. [Bishop, 1995] page pp 419) that the more exact term is

$$\begin{aligned} \ln p(\{\boldsymbol{z}\} \mid \mathcal{M}) \;=\; & -\,\alpha_{\mathrm{MAP}}\, \Omega(\boldsymbol{w}_{\mathrm{MAP}}) \;-\; \beta_{\mathrm{MAP}}\, R_{\mathrm{emp}} \;-\; \frac{1}{2}\, \ln |\boldsymbol{H}| \;+ \\ & \frac{W}{2}\, \ln \alpha_{\mathrm{MAP}} \;+\; \frac{l}{2}\, \ln \beta_{\mathrm{MAP}} \;+\; \ln H! \;+\; 2\, \ln H \;+\; \frac{1}{2}\, \ln\left(\frac{2}{\gamma}\right) \;+ \\ & \frac{1}{2}\, \ln\left(\frac{2}{l \;-\; \gamma}\right) \,. \end{aligned} \tag{7.68}$$

Here the terms in H, the number of hidden units in the network, appears because the posterior is locally approximated but there are equivalent regions in weights space.

As mentioned in Section 6.4.1 the networks are symmetric so that the signs of ingoing and outgoing weights to a hidden units can be flipped. This gives 2^H weight vectors representing the same function. The hidden units can also be reordered which gives $H!$ orderings. Together we obtain a factor of $(H!\, 2^H)$ equivalent representations through weight vectors of the same function.

7.8 Posterior Sampling

In order to compute the integrals like

$$A(f) \;=\; \int_W f(\boldsymbol{w})\, p(\boldsymbol{w} \mid \{\boldsymbol{z}\})\, d\boldsymbol{w} \tag{7.69}$$

we can sample weight vectors $\boldsymbol{w}_i$ to estimate

$$A(f) \;\approx\; \frac{1}{L} \sum_{i=1}^{L} f(\boldsymbol{w}_i) \,, \tag{7.70}$$

where the $\boldsymbol{w}_i$ are sampled according to $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$.

Because we cannot easily sample from $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ we use a simpler distribution $q(\boldsymbol{w})$ where we can sample from. We obtain

$$A(f) \;=\; \int_W f(\boldsymbol{w}) \, \frac{p(\boldsymbol{w} \mid \{\boldsymbol{z}\})}{q(\boldsymbol{w})} \, q(\boldsymbol{w}) \, d\boldsymbol{w} \tag{7.71}$$

which is an expectation in $q(\boldsymbol{w})$. This expectation can be approximated by

$$A(f) \;\approx\; \frac{1}{L} \sum_{i=1}^{L} f(\boldsymbol{w}_i) \frac{p(\boldsymbol{w}_i \mid \{\boldsymbol{z}\})}{q(\boldsymbol{w}_i)} \;, \tag{7.72}$$

where now the $\boldsymbol{w}_i$ are sampled according to $q(\boldsymbol{w})$.

To avoid the normalization of $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ which also includes integrations which are difficult to perform, the following term can be used

$$A(f) \;\approx\; \frac{\sum_{i=1}^{L} f(\boldsymbol{w}_i) \; \tilde{p}(\boldsymbol{w}_i \mid \{\boldsymbol{z}\}) \;/\; q(\boldsymbol{w}_i)}{\sum_{i=1}^{L} \tilde{p}(\boldsymbol{w}_i \mid \{\boldsymbol{z}\}) \;/\; q(\boldsymbol{w}_i)} \;, \tag{7.73}$$

where $\tilde{p}(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ is the unnormalized posterior, i.e. the product of the likelihood and the prior.

This approach is called *importance sampling*.

Because $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ is in general very small we must guarantee to sample in regions with large probability mass. This can be done by using *Markov Chain Monte Carlo* methods where regions with large mass are only left with low probability.

On method which improves random walk in a way that regions with large $p(\boldsymbol{w} \mid \{\boldsymbol{z}\})$ are sampled is called *Metropolis* algorithm. The Metropolis algorithm can be characterized as follows

$$\begin{aligned} &\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{candidate}} \text{ with probability} \\ &\begin{cases} 1 & \text{if} \quad p(\boldsymbol{w}^{\text{candidate}} \mid \{\boldsymbol{z}\}) \;>\; p(\boldsymbol{w}^{\text{old}} \mid \{\boldsymbol{z}\}) \\ \frac{p(\boldsymbol{w}^{\text{candidate}} \mid \{\boldsymbol{z}\})}{p(\boldsymbol{w}^{\text{old}} \mid \{\boldsymbol{z}\})} & \text{if} \quad p(\boldsymbol{w}^{\text{candidate}} \mid \{\boldsymbol{z}\}) \;<\; p(\boldsymbol{w}^{\text{old}} \mid \{\boldsymbol{z}\}) \end{cases} . \end{aligned} \tag{7.74}$$

Also *simulated annealing* can be used to estimate the expectation under the posterior which is similar to the Metropolis algorithm.

These sampling methods are discussed in [Neal, 1996] which is recommended for further reading.

Chapter 8

Feature Selection

Feature selection is important

1. to reduce the effect of the "curse of dimensionality" [Bellman, 1961] when predicting the target in a subsequent step,
2. to identify features which allow to understand the data
3. build models of the data generating process,
4. to reduce costs for future measurements, data analysis, or prediction, and
5. to identify targets for drugs, process improvement, failures.

Feature selection chooses a subset of the input components which are required for solving a task. In contrast feature construction methods compute new features from the original ones.

We focus on feature selection, where only components of the input vector are selected.

8.1 Feature Selection in Bioinformatics

Feature selection is an important topic in bioinformatics as high throughput methods measure many components at the same time. However for most tasks only few of these components are of interest. Extracting the relevant components allows to construct better (more robust and higher precision) classifiers or regression models.

Another important fact is that extracting relevant components gives insights into how nature is working. The extracted features can be used for building better models, can be used for improving or designing experiments, and can be used for identifying key components of systems which in turn can be used for drug design or switching off subsystems.

Medical doctors are changing their attention to effective screening and diagnostic involving high resolution, high throughput methods in order to early detect diseases. Early detection increases the chances of curing the disease.

8.1.1 Mass Spectrometry

One example is the high resolution mass spectrometry data which comes with many measurement values which are noisy.

In order to apply machine learning methods and obtain sufficient classification accuracy, dimensionality reduction is necessary – preferably through feature selection methods.

Another objective is to identify biomarkers which indicate the presents or the state of specific diseases. The best known biomarker is the prostate specific antigen (PSA) for prostate cancer. Nowadays it is commonly believed that for many diseases not a single biomarker exists but a pattern of biomarkers [Conrads et al., 2003].

Ovarian Cancer

In ovarian cancer studies [Petricoin et al., 2002a, Conrads et al., 2003] on mass spectrometry data high classification rates (95%) in distinguishing healthy persons from persons with cancer. However for clinical relevance a specificity of 99.6% is required. Note that specificity is "true negative / (true negative + false positive)" (performance on the negative class); sensitivity is "true positive / (true positive + false negative)" (performance on the positive class), accuracy is "true positive + true negative / all data" (the performance on all data), the balanced accuracy is the mean of specificity and sensitivity. For achieving this mark techniques with higher resolution and, therefore, techniques which produce more data have been developed, e.g. surface-enhanced laser desorption/ionization time-of-flight mass spectrometry (SELDI TOF MS) [Conrads et al., 2003].

Dimensionality reduction through principal component analysis (PCA – see later in unsupervised techniques) achieved 100% accuracy [Lilien et al., 2003]. In [Wu et al., 2003] feature selection techniques were applied and with 15 to 25 features an accuracy of 92% was obtained. The advantage of the features selection technique is that from the features it is possible to infer biomarkers which is more difficult form PCA components. [Tibshirani et al., 2003] achieved 72.5% balanced accuracy with only seven features.

Prostate Cancer

With SELDI TOF MS data to classify healthy and cancerous persons, decision trees achieved 81% sensitivity and a 97% specificity, thus 89& balanced accuracy [Adam et al., 2002].

This result was improved using AdaBoost in [Qu et al., 2002] to average sensitivity of 98.5% and a specificity of 97.9% (balanced accuracy 98%). However, with feature selection the results were not as good. In [Lilien et al., 2003] with PCA as dimensionality reduction method and LDA as classifier an average accuracy of 88% was achieved. Feature selection was used in [Petricoin et al., 2002b, Wulfkuhle et al., 2003] which resulted in 95% specificity and 83% sensitivity (balanced accuracy of 83%).

These results show varying results if using feature selection which was discussed in [Diamandis, 2003].

8.1.2 Protein Sequences

The most classical task of bioinformatics is sequences processing and comparing new sequences with known sequences in order to detect evolutionary relationships. The is called "homology

detection" and is by far the most used method to determine structure or function of new genes or their products the proteins.

Traditionally homology detection is done by alignment methods [Needleman and Wunsch, 1970, Smith and Waterman, 1981, Pearson, 1990, Altschul et al., 1990, 1997]. Other methods use Hidden Markov Models (HMMS – [Krogh et al., 1994a, Eddy and Durbin, 1995, Sonnhammer et al., 1997, 1998]) or profiles [Gribskov et al., 1987, 1990]. However still about 40% of the sequenced human genes have not been classified by their function through homology methods [Lander et al., 2001, Venter et al., 2001].

The methods considered in this subsection are based on *motifs*, that are highly conserved patterns of the protein sequence which allow to classify the sequences [Falquet et al., 2002, Nevill-Manning et al., 5865-5871, Huang and Brutlag, 2001]. Here each motif of a motif database is considered as a feature describing the sequence. If we assume that evolution used building blocks to construct new proteins then these building blocks may be identified. The motifs as building blocks are responsible for catalytic sites, binding sites, or structural patterns (used for folding or stability). Assumption is that also proteins can be classified if there is no known homolog but other proteins which share the same building blocks. Feature selection focuses now on the task to find the best suited motifs for a specific task [Ben-Hur and Brutlag, 2003, 2004, Logan et al., 2001].

SVM Kernels as Feature Selection Methods for Motifs

Note that the kernels like string kernel, spectrum kernel, mismatch kernel, etc. identify important motifs of the positive class by selecting appropriate support vectors.

The vector $\boldsymbol{w}$ of the linear support vector machine in the space of subsequences weights motifs (pattern) which are indicative for the positive class highly positive and motifs which are indicative for the negative class negative.

For example the motifs can explicitly represented by the spectrum kernel if explicitly computing $\boldsymbol{w}$.

Basically $\boldsymbol{w}$ only selects the most indicative motifs for the classification task at hand.

8.1.3 Microarray Data

Gene expression profiles obtained by the microarray technique provide a snapshot of the expression values of some thousand up to some ten thousand genes in a particular tissue sample or a particular sample from other organisms (bacteria, plants, etc.). The advantage of the microarray method – namely to monitor a large number of variables of a cell's (or a piece of tissue's) state, however, often turns out to be difficult to exploit. The number of samples is small and the level of noise is high which makes it difficult to detect the small number of genes relevant to the task at hand. Therefore, specific gene selection methods must be designed in order to reliably extract relevant genes. Here the features are the genes, therefore feature selection is equal to gene selection in the microarray case. However in principle also tissue extraction is possible, i.e. select the tissues which are correlated to certain genes or certain gene clusters.

The microarray technique [Southern, 1988, Lysov et al., 1988, Drmanac et al., 1989, Bains and Smith, 1988] is a recent technique which allows to monitor the concentration of many kinds of messenger RNA (mRNA) simultaneously in cells of a tissue sample and provides a snapshot of the pattern of gene expression at the time of preparation [Wang et al., 1998, Gerhold et al.,

1999]. The so-called DNA microarrays allow for the first time the simultaneous measurement of several 1000 or several 10,000 expression levels providing valuable information about whole genetic networks. DNA microarrays allow to search for genes related to certain properties of the sample tissue and to extract related genes via dependencies in their expression pattern.

Fig. 8.1 depicts the microarray procedure. Messenger RNA is extracted from the samples (Step 1) and reversely transcribed to cDNA (Step 2). This "target" cDNA is then coupled to a fluorescent dye (Step 3). The target cDNA is then hybridized with a large number of probes of immobilized DNA (steps 4 and 5) which had been synthesized and fixed to different locations of the DNA chip during fabrication. The cDNA from the samples binds to their corresponding probes on the chip (Step 5). After cleaning, the chip is scanned with a confocal microscope and the strength of the fluorescent light is recorded (Step 6). Genes which are predominantly expressed in the sample give rise to bright spots of strong fluorescent light. No expression is indicated by weak fluorescent light. After segmentation of the stained locations on the chip and a correction for background intensity, intensity values are transformed to real numbers for every location (Step 7). After processing, the data from several experiments with different samples are collected and represented in matrix form, where columns correspond to tissue samples, rows correspond to genes, and matrix entries describe the result of a measurement of how strong a particular gene was expressed in a particular sample.

Expression values as measured by the DNA microarray technique are noisy. Firstly, there exists biological noise, because samples do not show the same "expression state" and exactly the same levels of mRNA even if they belong to the same class or the same experimental condition. Then there is noise introduced by the microarray measurement technique. Sources of noise include tolerances in chip properties which originate from the fabrication process, different efficiencies for the mRNA extraction and the reverse transcription process, variations in background intensities, nonuniform labeling of the cDNA targets (the dye may bind multiple times and with different efficiencies), variations in the dye concentration during labeling, pipette errors, temperature fluctuations and variations in the efficiency of hybridization, and scanner deviations. The effect of measurement noise can be reduced by averaging over multiple measurements using the same sample but usually remains large. Measurement noise is not always Gaussian. [Hartemink et al., 2001] for example found that the measurement noise distribution of the logarithmic expression values has heavy tails.

Gene Selection for Microarrays

As already mentioned feature selection is in most microarray applications equal to gene selection. In most cases the tissue samples are labeled according to conditions like healthy tissue or tissue associated with a disease or like plant samples from certain growth periods.

Gene selection aims at three goals:

- (a) data preprocessing in order to improve the prediction quality of machine learning approaches,
- (b) identification of indicator genes (this would aid the interpretation and understanding of the data), and
- (c) reducing costs, if microarray data are used for example for diagnostic purposes.

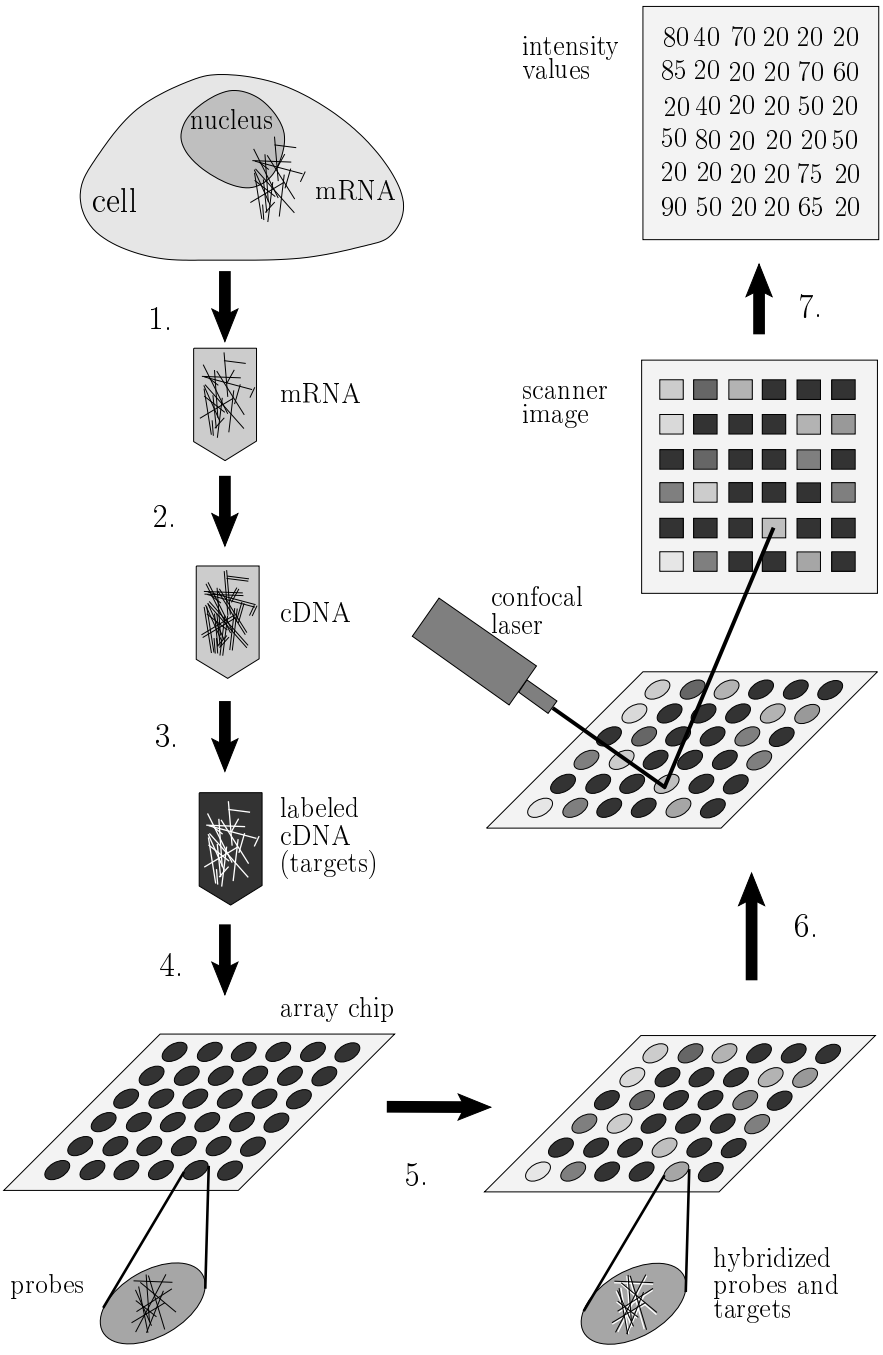


Figure 8.1: The microarray technique (see text for explanation).

Item (a) is an important issue in machine learning if the input dimension is larger than the number of samples. The reduction in performance for data sets with many attributes is known as the "curse of dimensionality" [Bellman, 1961]. According to [Stone, 1980], the number of training examples has to increase exponentially with the number of dimensions in order to ensure that an estimator also performs well for higher dimensional data. Otherwise overfitting (high variance in model selection) occurs, that is fitting of the selected model to noise in the training data. On the other hand, if the model class is chosen to be smooth so that the variance of model selection is restricted (low overfitting), then underfitting (high bias of model selection) occurs, that is the training data is not approximated well enough. The later is shown by [Friedman, 1997] who demonstrated for K-nearest neighbor classifiers that the curse of dimensionality leads to large bias. Practical applications confirm the theory: many input dimensions lead to poor generalization performance. Fewer features on the other hand should improve generalization for equal training error.

For microarray data the situation is especially difficult because the number of features (genes) is often more than 10 times larger than the number of examples (tissue samples). The high level of noise additionally complicates the selection of relevant genes of microarray data. Both facts, the large number of genes and the presence of noise, led [Guyon et al., 2002] to state that "the features selected matter more than the classifier used" for DNA microarray data classification, a fact which will be confirmed by our analysis later on.

Item (b) refers to the identification of genes whose expression values change with the sample class. Genes which show different expression values in a control condition when compared to the condition to analyze are useful to differentiate between these conditions and should be extracted (see [Jäger et al., 2003]). The knowledge of the relevant genes can then be exploited in two ways. Firstly, cellular mechanisms can be understood and active pathways may be identified. Secondly, target genes or target proteins for causing or avoiding conditions can be detected. In medical applications both kinds of information are highly relevant for diagnosis and drug design. Note, however, that the selection of genes for prediction and the selection of genes whose expression levels are correlated lead to different sets. Redundant sets of genes, which are the outcome of the latter task, may lead to a reduced performance of the former task. On the other hand, genes selected for the purpose of prediction may not include genes strongly correlated to each other in order to keep the number of features small.

Item (c) refers to the costs of large scale microarray studies, for example, for diagnostic purposes. Small gene ensembles lead to cheaper chips (fewer probes on a chip), to savings in manpower (fewer experiments), and to easier interpretable experiments [Jäger et al., 2003].

8.2 Feature Selection Methods

In principle feature selection is possible for supervised and unsupervised methods. Here we focus on feature selection for supervised methods. Unsupervised feature selection methods are mostly related to unsupervised techniques, e.g. to select components which are most informative or which are not redundant etc.

For simplicity let us consider a classification task where the objects to classify are described by vectors with a fixed number of components (the features). The training set consists of vectors which are labeled by whether the according object belongs to a class or not and we assume that

there are only two classes. Given the training data, a classifier should be selected which assigns correct class labels to the feature vectors. The goal of machine learning methods is not only to select a classifier which performs well on the training set, but which also correctly classifies new examples, that is which correctly predicts events of the future.

There are two classes of preprocessing methods which are commonly used to improve machine learning techniques: *feature selection* and *feature construction* methods. Feature construction methods compute new features as a combination of the original ones and are often used for dimensionality reduction which will be treated later in the unsupervised learning chapter.

Feature selection methods, in contrast to feature construction, choose a subset of the input components which are supposed to be relevant for solving a task and leave it to a subsequent stage of processing to combine their values in a proper way. In the following we focus on feature selection, that is on the task of choosing a subset of "informative" input components, that is components which are relevant for predicting the class labels. The classifier is then selected using the reduced feature vectors as the objects' description. Therefore, only feature selection techniques address items (b) and (c) from the previous section, that is the extraction of indicator genes and reducing costs. Note that item (b) may not be fully addressed by feature selection approaches because redundant features are avoided and not all indicator genes are extracted. However, the missing genes can be extracted by correlation analysis in a subsequent step.

Review articles on feature selection have been published in a "special issue on relevance" of the journal *Artificial Intelligence* [Kohavi and John, 1997, Blum and Langley, 1997] and a "special issue on variable and feature selection" of the *Journal of Machine Learning Research* [Guyon and Elisseeff, 2003] to which we refer the reader for more details. The book of [Liu and Motoda, 1998] also gives an overview on feature selection.

The best and most comprehensive overview with the newest methods can be found in [Guyon et al., 2006].

Feature selection methods perform either *feature ranking* or *subset selection*. In feature ranking an importance value is assigned to every feature while subset selection attempts at constructing an optimal subset of features. While some feature ranking methods do not consider dependencies between features, subset selection methods usually do and may even include features which have low correlations with the class label if justified by a good classification performance. The latter usually happens if dependencies between features (and not between class label and a certain feature) are important for prediction. In those cases the selection of interacting features is important, but it is also difficult to achieve (see [Turney, 1993a,b]).

Feature selection methods fall into one of two categories [Langley, 1994, Kohavi and John, 1997, John et al., 1994, Das, 2001, Liu and Motoda, 1998, Liu and Setiono, 1996]:

- (a) *filter methods*: do not use an explicit regression or classification method;
- (b) *wrapper methods*: use a regression or classification method as the objective function for the evaluation of a subset of features.

8.2.1 Filter Methods

Filter methods extract features whose values show dependencies with class labels without explicitly relying on a regression or classification method.

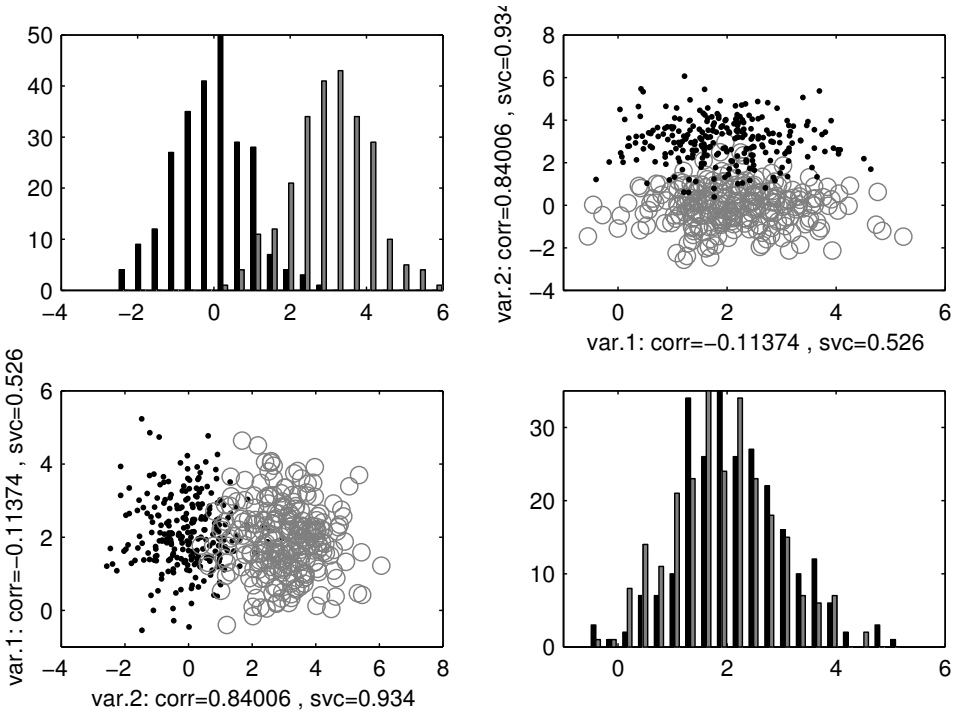


Figure 8.2: Simple two feature classification problem, where feature 1 (var. 1) is noise and feature 2 (var. 2) is correlated to the classes. In the upper right figure and lower left figure only the axis are exchanged. The upper left figure gives the class histogram along feature 2 whereas the lower right figure gives the histogram along feature 1. The correlation to the class (corr) and the performance of the single variable classifier (svc) is given. Copyright © 2006 Springer-Verlag Berlin Heidelberg.

One example are statistical methods which compute the statistical dependencies between class labels and features and select features where the dependencies are strong.

Remember Fig. 2.14 (below Fig. 8.2). Statistical methods can distinguish between feature 1 and feature2 because feature 1 separates the classes.

The calculation of dependencies is based on Pearson's correlation coefficient, Wilcoxon statistics, t-statistics, Fisher's criterion or signal-to-noise ratios [Hastie et al., 2001, Golub et al., 1999, Furey et al., 2000, Tusher et al., 2001]. Statistical methods are fast and robust, but assume certain data or class distributions and cannot recognize dependencies between features. In principle, statistical methods can serve as subset selection methods if the dependency between a whole feature subset and the class label is computed. For example, the mutual information between feature sets and class labels has been considered in [Koller and Sahami, 1996]. However the number of possible subsets increases exponentially with the number of features which makes these approaches unattractive. Therefore, the method in [Koller and Sahami, 1996] is only tractable if approximations are made.

We want to give some simple statistical feature selection methods. In the following the target is y and the j-th feature is x_j. The statistical methods measure the correlation between the j-th

feature and the label y. We define

$$\bar{y} \;=\; \frac{1}{l}\sum_{i=1}^{l} y^i \tag{8.1}$$

$$\bar{x}_j \;=\; \frac{1}{l}\sum_{i=1}^{l} x_j^i \tag{8.2}$$

$$\sigma_y^2 \;=\; \frac{1}{l}\sum_{i=1}^{l} \left(y^i \;-\; \bar{y}\right)^2 \tag{8.3}$$

$$\sigma_j^2 \;=\; \frac{1}{l}\sum_{i=1}^{l} \left(x_j^i \;-\; \bar{x}_j\right)^2 \;. \tag{8.4}$$

For classification we have $y \in \{-1, 1\}$ and assume we have l^+ class $+1$ and l^- class -1 examples then we define

$$\bar{x}_j^+ \;=\; \frac{1}{l^+}\sum_{i=1;y^i=1}^{l} x_j^i \tag{8.5}$$

$$\bar{x}_j^- \;=\; \frac{1}{l^-}\sum_{i=1;y^i=-1}^{l} x_j^i \tag{8.6}$$

$$\left(\sigma_j^+\right)^2 \;=\; \frac{1}{l^+}\sum_{i=1;y^i=1}^{l} \left(x_j^i \;-\; \bar{x}_j^+\right)^2 \tag{8.7}$$

$$\left(\sigma_j^-\right)^2 \;=\; \frac{1}{l^-}\sum_{i=1;y^i=1}^{l} \left(x_j^i \;-\; \bar{x}_j^-\right)^2 \;. \tag{8.8}$$

Pearson's correlation coefficient is

$$\mathrm{corr} \;=\; \frac{\sum_{i=1}^{l}\left(x_j^i \;-\; \bar{x}_j\right)\;\left(y^i \;-\; \bar{y}\right)}{\sigma_y\;\sigma_j} \tag{8.9}$$

and the test for correlation can be performed using the paired t-test (assumptions: normal distribution and both distributions have same variance – only test for equal mean).

The "signal-to-noise ratio" is

$$\frac{\bar{x}_j^+ \;-\; \bar{x}_j^-}{\sigma_j^+ \;+\; \sigma_j^-} \tag{8.10}$$

which is also known as Golub's criterion.

The Fisher criterion is

$$\frac{\left(\bar{x}_j^+ - \bar{x}_j^-\right)^2}{\left(\sigma_j^+\right)^2 + \left(\sigma_j^-\right)^2} . \tag{8.11}$$

The two-sample t-test uses

$$\frac{\bar{x}_j^+ - \bar{x}_j^-}{\sqrt{\left(\sigma_j^+\right)^2 / l^+ + \left(\sigma_j^-\right)^2 / l^+}} . \tag{8.12}$$

The "relief" methods [Kira and Rendell, 1992, Kononenko, 1994, Robnik-Sikonja and Kononenko, 1997] are another approach which assign relevance values to features. Values are assigned according to the average separation of data vectors belonging to different classes minus the average separation of data points belonging to the same class. The averages are computed by randomly selecting a data point and determining its nearest data points from the same class and the opposite class. The "relief" methods are fast, can detect feature dependencies but – again – do not remove redundant features.

Combinatorial search procedures are able to remove redundant features from the selected set. These methods exhaustively test all feature subsets for their ability to separate the classes, that is whether two training vectors have the same values on the selected feature subset but different class labels. After testing, the minimal subset necessary to predict the class label is chosen. For example such methods are FOCUS [Almuallim and Dietterich, 1991] or the probabilistic approach in [Liu and Setiono, 1996]. Combinatorial search methods, however, suffer from high computational costs and can only be applied to a small number of features. They are prone to overfitting through noise but on the other hand they will find the best solution in the noiseless case. Another feature subset selection which – like FOCUS – searches for a minimal necessary feature subset to separate the classes is based on decision trees [Cardie, 1993]. The decision tree is used for separating the classes but not as a classifier. This method, however, is not applicable for small training sets because only $\log_2 m$ features are selected if m training examples are available. However many bioinformatics task like microarrays, the sample size is usually below 100, only $\log_2 100 \approx 7$ genes are typically selected. These are too few genes for decision trees.

The problem of selecting relevant features can be more difficult as an be achieved by statistical methods. Remember Fig. 2.16 which is below given as Fig. 8.3 which shows an example where the correlation of features with the class is zero. Statistical methods which take only the mean (or center) of the classes may fail to extract the best features as it was shown in Fig. 2.17 which is below shown again as Fig. 8.4. In this figures the shape of the clusters of the classes is important. In the left and right subfigure the feature's mean values and variances are equal for each class. However, the direction of the variance differs in the subfigures leading to different performance in classification.

We have also shown in Tab. 2.1 (below Tab. 8.1) where features which have no correlation with the target should be selected. The table shown also an example where the feature with the largest correlation with the target should not be selected. Reason is that correlation between features can

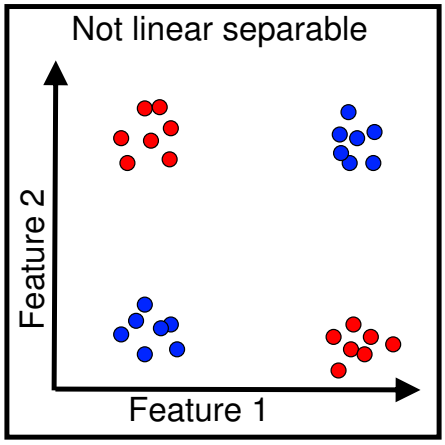


Figure 8.3: An XOR problem of two features, where each single feature is neither correlated to the problem nor helpful for classification. Only both features together help.

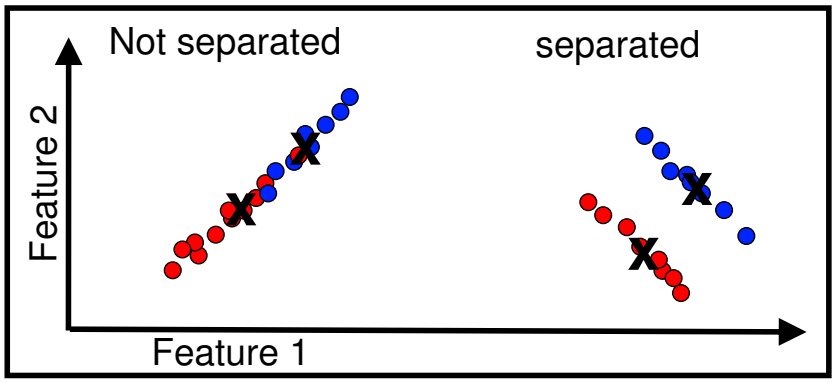


Figure 8.4: The left and right subfigure each show two classes where the features mean value and variance for each class is equal. However, the direction of the variance differs in the subfigures leading to different performance in classification.

f_1	f_2	t	f_1	f_2	f_3	t
-2	3	1	0	-1	0	-1
2	-3	-1	1	1	0	1
-2	1	-1	-1	0	-1	-1
2	-1	1	1	0	1	1

Table 8.1: Left hand side: the target t is computed from two features f_1 and f_2 as $t = f_1 + f_2$. No correlation between t and f_1.

help for prediction like if features are correlated by noise then one feature can be used to remove the noise from another feature.

In Tab. 8.1, the target t is computed from two features f_1 and f_2 as $t = f_1 + f_2$. All values have mean zero and the correlation coefficient between t and f_1 is zero. In this case f_1 should be selected because it has negative correlation with f_2. The top ranked feature may not be correlated to the target, e.g. if it contains target-independent information which can be removed from other features. The right hand side of Tab. 8.1 depicts another situation, where $t = f_2 + f_3$. f_1, the feature which has highest correlation coefficient with the target (0.9 compared to 0.71 of the other features) should not be selected because it is correlated to all other features.

8.2.2 Wrapper Methods

Wrapper methods [Kohavi and John, 1997, John et al., 1994] use a classifier as the objective function for the evaluation of a subset of features. The classifier is obtained through a model selection (training) method which minimizes the classification error on the training data. The classifier is then used to compute the prediction error on a validation set. Typical classifiers are decision trees, for example ID3 [Quinlan, 1986], CART [Breiman et al., 1984], and C4.5 [Quinlan, 1993], or instance-based classifiers like k-nearest neighbor.

Well known wrapper methods are the nested subset methods which are based on greedy strategies like hill-climbing (for example SLASH [Caruana and Freitag, 1994] and the random mutation hill climbing – random mutation of feature presence map – described in [Skalak, 1994]). Nested subset methods perform either "forward selection" [Cover and Campenhout, 1977] or "backward elimination" [Marill and Green, 1963].

Forward vs. Backward Selection.

Forward selection works in the underfitting regime. It starts from an empty set of features and adds features step by step which lead to the largest reduction of the generalization error. Backward elimination, on the other hand, works in the overfitting regime. It starts with the set of all features and removes unimportant features step by step in order to maximally reduce the generalization error. The major shortcoming of these methods is that they do not consider all possible combinations of features [Cover and Campenhout, 1977]. If, for example, only the XOR of two features is important, these features would not be recognized by a forward selection procedure which adds only a single feature at a time.

The backward selection procedure suffers from a similar problem. Assume that one feature conveys the information of two other features and vice versa. The best strategy would be to

remove these two features to obtain a minimal set but backward selection may keep these two features and remove the single one which is represented by the other two. Another problem of backward selection is to determine good candidate features for deletion because overfitting makes it hard to distinguish between label noise fitting and true dependencies with class labels.

Other search strategies are computationally more expensive but explore more possible feature sets. Such search methods include beam and bidirectional search [Siedlecki and Sklansky, 1988], best-first search [Xu et al., 1989], and genetic algorithms [Vafaie and Jong, 1992, 1993, Bala et al., 1995].

8.2.3 Kernel Based Methods

Recently kernel based feature selection methods which use the support vector machine (SVM) approach have shown good performance for feature selection tasks. See for example the review in [Guyon and Elisseeff, 2003]. Kernel based feature selection methods are emphasized through this subsection since they are especially suited for microarray data due to the fact that they have shown good results in high dimensional data spaces and have been successfully applied to noisy data. These methods use either one of two feature selection technique, which were already known in the field of neural networks:

- (a) feature selection by pruning irrelevant features after a classifier has been learned or
- (b) adding a regularization term to the training error which penalizes the use of uninformative features during learning a classification task.

8.2.3.1 Feature selection after learning

[Guyon et al., 2002] proposed a feature selection method for support vector learning of linear classifiers where the features with the smallest squared weight values are pruned after learning is complete. This method is a special case of the "Optimal Brain Surgeon" (OBS,) or "Optimal Brain Damage" techniques given in Section 6.4.5.3 where input weights can be deleted. OBS handles dependent features or whereas OBD assumes independent feature under the assumption that feature values have variance one. Intuitively, the support vector method corresponds to projecting the normal vector of the separating hyperplane into the subspace perpendicular to the less important directions. The features for which these values are lowest are then deleted. [Guyon et al., 2002] also describe an iterative version of this feature selection procedure where the feature with the smallest absolute weight is removed of the model of a linear SVM after each SVM optimization step. This method is then called "Recursive Feature Elimination" (RFE) and is an example of backward elimination of features. It has recently been extended by [Rakotomamonjy, 2003] to nonlinear kernels. Note, however, that these methods which prune features after learning cannot detect important but redundant features and that they are sensitive to outliers.

8.2.3.2 Feature selection during learning

Regularization techniques have been proposed for support vector machines in order to improve prediction performance by selecting relevant features. The first set of techniques directly favors

SVMs with sparse weight vectors. This can be done by using the 1-norm in the SVM objective function, a technique, which is known as the linear programming (LP) machine [Schölkopf and Smola, 2002, Smola et al., 1999, Frieß and Harrison, 1998]. This approach leads to many zero components of the weight vector and to the removal of the corresponding features. In [Bradley and Mangasarian, 1998, Bi et al., 2003] these methods are utilized together with backward elimination. In [Bradley and Mangasarian, 1998] the 0-norm of the weight vector is considered as an objective to select a classifier. The 0-norm counts the non-zero components of the weight vector which leads to a discrete and NP-hard optimization problem. Approximations can be made but they are sensitive to the choice of parameters (see [Weston et al., 2003]) and the optimization is still computationally complex in high dimensions. [Weston et al., 2003] propose an improved approximation of the 0-norm, which reduces to a method which iteratively solves 1-norm SVMs and adjusts scaling factors for the different features. In [Perkins et al., 2003] both the 0-norm and the 1- or 2-norm are used for feature selection, where the 1- or 2-norm serves for regularization and the 0-norm selects features.

The second set of techniques is based on the proper choice of scaling factors for the different features. [Weston et al., 2000] applies scaling factors to the 2-norm SVM approach ("R2W2"). Two phases are performed iteratively. First the SVM is optimized and a bound for the generalization error is computed. Secondly, the scaling factors are optimized by a gradient descent method in order to minimize the bound. This method has the advantage, that it can be extended to non-linear kernels, where the scaling factors are put into the kernel function. On the other hand, this method is computationally expensive because two optimization problems (SVM solution and error bound) have to be solved for every iteration and the kernel matrix must be evaluated for every step. Additionally, the gradient based optimization suffers from convergence to local optima.

In bioinformatics so far the most common choice are statistical methods e.g. for selecting relevant genes from microarray data like in [Pomeroy et al., 2002]. However, support vector machine based methods have recently been applied with good success [Shipp et al., 2002].

8.2.3.3 P-SVM Feature Selection

The Potential Support Vector Machine (P-SVM) from Section 4.14 is suitable for feature selection.

The P-SVM selects the optimal features in Tab. 8.1 because it takes into account the correlation between features.

The P-SVM was introduced as a method for selecting a classification function, where the classification boundary depends on a small number of "relevant" features. The method can be used for feature selection, and it can also be used for the subsequent prediction of class labels in a classification task. The optimal P-SVM classifier is a classifier with minimal empirical risk but with the largest margin after sphering the data. Feature selection can be interpreted as removing features from the optimal classifier but bounding the increase in mean squared error through the value ϵ.

The first observation is that optimization (that is the selection of the proper values for $\boldsymbol{\alpha}$ and b) only involves the measurement matrix $\boldsymbol{X}$ and the label vector $\boldsymbol{y}$.

For example, in order to apply the P-SVM method to the analysis of microarray data, we identify the objects with samples, the features with genes, and the matrix $\boldsymbol{X}$ with the matrix of expression values. Due to the term $\boldsymbol{X}\ \boldsymbol{X}^T$ in the dual objective, the optimization problem is

well defined for measurement matrices $\boldsymbol{X}$ of expression values. From a conceptual point of view, however, it is advantageous to interpret the matrix $\boldsymbol{X}$ of observations (here: expression values) itself as a dot product matrix whose values emerge as a result of the application of a measurement kernel.

The second observation is that an evaluation of the classifier for new samples $\boldsymbol{x}$ only involves the measurement of its expression values $(\boldsymbol{x})_j$ for the selected "support" genes j. The number of "support genes" depends on the value of a noise parameter, the correlation threshold ϵ. If the value of ϵ is large during learning, only a few "most informative" genes are kept. The other genes are discarded because of being too noisy or not conveying information about the class labels.

The set of all genes for which the weighting coefficients α_j are non-zero (the set of support genes) is the selected feature set. The size of this set is controlled by the value of ϵ, and if the P-SVM is applied for feature selection, the value of ϵ should be large.

In the following the P-SVM for feature selection is given.

P-SVM feature selection

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \quad & \frac{1}{2}\left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^\top \boldsymbol{F}^\top \boldsymbol{F} \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right) \\ & - \boldsymbol{y}^\top \boldsymbol{F} \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right) + \epsilon\, \mathbf{1}^\top \left(\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-\right) \\ \text{subject to} \quad & \mathbf{0} \leq \boldsymbol{\alpha}^+ \,,\ \mathbf{0} \leq \boldsymbol{\alpha}^- \,. \end{aligned} \tag{8.13}$$

- ϵ: parameter to determine the number of features, large ϵ means few features
- $\alpha_j = \alpha_j^+ - \alpha_j^-$: relevance value for complex feature vector $\boldsymbol{z}_j$, $\alpha_j \neq 0$ means that vector no. j is selected, positive α_j means class 1 indicative vector $\boldsymbol{z}_j$ and negative α_j means class -1 indicative
- $\boldsymbol{F} = \boldsymbol{X}^\top \boldsymbol{Z}$ with data matrix $\boldsymbol{X}$ and the matrix of complex features vectors $\boldsymbol{Z}$ (variable selection: $\boldsymbol{F} = \boldsymbol{X}$)
- $\boldsymbol{y}$: vector of labels

8.2.4 Automatic Relevance Determination

Automatic Relevance Determination (ARD) is a special case of weight decay from Section 6.4.5.4 and Section 7.2, where weight decay is applied to the input hidden weights and the maximum posterior is determined [Neal, 1996, MacKay, 1993].

Here a Gaussian prior is used as in the example in Section 7.5. However the Gaussian is has as covariance matrix a diagonal matrix with entries β_j^{-1}.

If $\boldsymbol{H}$ is the Hessian of the log-posterior evaluated around the maximum $\boldsymbol{w}_{\mathrm{MAP}}$, the β_j can be estimated as

$$\beta_j = \frac{1 - \beta_j H_{jj}^{-1}}{(w_i)^2_{\mathrm{MAP}}} \,. \tag{8.14}$$

Using this update of β, the evidence framework in Section 7.5 can be used. The MAP estimate and the value of β_i are iteratively updated.

Note that the "R2W2" method which was considered in previous section is similar to the ARD approach.

8.3 Microarray Gene Selection Protocol

In this section we describe the protocol for extracting meaningful genes from a given set of expression values for the purpose of predicting labels of the sample classes. The protocol includes data preprocessing, the proper normalization of the expression values, the feature selection and ranking steps, and the final construction of the predictor.

Note that our feature selection protocol requires class labels which must be supplied together with the expression values of the microarray experiment. For the following, however, we assume that the task is to select features for classification and that l labeled samples are given for training the classifier.

8.3.1 Description of the Protocol

1. **Expression values vs. log-ratios.** Before data analysis starts it is necessary to choose an appropriate representation of the data. Common representations are based on the ratio $T_j = \frac{R_j}{G_j}$ of expression values between the value R_j (red) of a gene j in the sample to analyze and the value G_j (green) in the control sample, and the log ratio $L_j = \log_2(T_j)$.

   For arrays like the Affymetrix chips only one kind of expression level which indicates the concentration of the according mRNA in the sample. Here also the log expression value is a common choice for representing the expression levels.

2. **Normalization and Summarization.** Different normalization methods exist to make the measurements from different arrays comparable. The most famous ones are "Quantile normalization" and "Cyclic Loess".

   Some techniques like the Affymetrix GeneChip makes more than one measurement for each gene. A so-called probe set makes 11 to 21 measurements for each gene. To obtain a single expression value these measurements must be "summarized". Here also different summarization methods like RMA, GCRMA, MBEI, MAS5.0, or FARMS exist. Some methods like FARMS are able to supply a present call, which is discussed in next item.

3. **Present call.** The present call is usually the first step in the analysis of microarray data. During this step genes are identified for which the confidence is high, that they are actually expressed in at least one of the samples. Genes for which this confidence is low are excluded from further processing in order to suppress noise.

   For this purpose an error model has to be constructed for the expression values or their ratios (sometimes before, sometimes after averaging across multiple measurements of the same sample — see [Tseng et al., 2001, Schuchhardt and Beule, 2000, Kerr et al., 2000, Hartemink et al., 2001]. This error model accounts for both measurement specific noise (for

example background fluctuations), which affects all expression values in a similar way, and gene specific noise (for example the binding efficiency of the dye), which affects expression values for different genes in a different way. Using this error model one assigns a p-value, which gives the probability that the observed measurement is produced by noise, to every measurement of an expression level. If the P-value is smaller than a threshold q_1 (typical values are 5%, 2%, or 1%), the expression level is marked "reliable". If this happens for a minimum number q_2 (typical values range from 3 to 20) of samples, the corresponding gene is selected and a so-called present call has been made.

4. **Standardization.** Before further processing, the expression values are normalized to mean zero and unit variance across all training samples and separately for every gene. Standardization accounts for the fact that expression values may differ by orders of magnitudes between genes and allows to assess the importance of genes also for genes with small expression values.

   Some summarization methods may already account for comparable variance and zero mean – in this case standardization is not necessary.

5. **Gene ranking and gene selection.** Here we assume that a feature selection method has been chosen where the size of the set of selected genes is controlled by a hyperparameter which we call ϵ in the following (according to the P-SVM).

   In this step we perform two loops: An "inner loop" and an "outer loop" (the leave-one-out loop). The inner loop serves two purposes. It ranks features if only a subset method like the P-SVM is available and it makes feature selection more robust against variations due to initial conditions of the selection method. The outer loop also serves also two purposes. It makes the selection robust against outlier samples and allows to determine the optimal number of selected genes together with the optimal values of hyperparameters for the later prediction of class labels. In order to do this, a predictor must be constructed. Here we suggest to use a ν-SVM where the value of ν is optimized by the outer loop. In order to implement the outer (leave-one-out) loop, l different sets of samples of size $l-1$ are constructed by leaving out one sample for validation. For each of the l sets of reduced size, we perform gene selection and ranking using the following "inner loop".

   Inner loop. The subset selection method is applied multiple times to every reduced set of samples for different values of ϵ. For every set of samples multiple sets of genes of different size are obtained, one for every value of ϵ. If the value of ϵ is large, the number of selected genes is small and vice versa. The inner loop starts with values of ϵ which are fairly large in order to obtain few genes only. Gradually the value is reduced to obtain more genes per run. Genes obtained for the largest value of ϵ obtain the highest rank, the second highest rank is given to genes which additionally appear for the second largest value of ϵ, etc. The values of ϵ are constant across sample sets. The minimal value should be chosen, such that the number of extracted genes is approximately the total number l of samples. The maximal value should be chosen such that approximately five to ten genes are selected. The other values are distributed uniformly between these extreme values.

   Outer loop. The results of the inner loops are then combined across the l different sets of samples. A final ranking of genes is obtained according to how often genes are selected in the l leave-one-out runs of the inner loop. If a gene is selected in many leave-one-out runs, it is ranked high, else it is ranked low. Genes which are selected equally often are ranked

according to the average of their rank determined by the inner loops. The advantage of the leave-one-out procedure is that a high correlation between expression values and class labels induced by a single sample is scaled down if the according sample is removed. This makes the procedure more robust against outliers.

The outer loop is also used for selecting an optimal number of genes and other hyperparameters. For this purpose, ν-SVMs are trained on each of the l sets of samples for different values of the hyperparameter ν and the number F of high ranking genes (ranking is obtained by the inner loop). Then the average error is calculated on the left out samples. Since the leave-one-out error as a function of the number F of selected genes is noisy, the leave-one-out error for F is replaced by the average of the leave-one-out errors for F, $F + a$, and $F - a$. Then the values of the hyperparameter ν and the number of genes F which give rise to the lowest error are selected. This completes the feature selection procedure.

8.3.2 Comments on the Protocol and on Gene Selection

Normalization and Summarization of new arrays. If a new array has to be analyzed then all known arrays together with the new array must be normalized and used for summarization. Thereafter machine learning methods can be applied to the training set and the new array can be classified.

Corrections to the outer, leave-one-out loop. The samples which were removed from the data in the outer loop when constructing the l reduced subsets for the gene ranking should not be considered for the present call and for determining the normalization parameters. Both steps should be done individually for each of the l sets of sample, otherwise feature or hyperparameter selection may not be optimal.

Computational Costs. The feature selection protocol requires $l \times n_\epsilon$ feature selection runs, where n_ϵ is the number of different values of the ϵ parameter. However the computational effort is justified by the increased robustness against correlation by chance (see next item) and the elimination of single sample correlations.

Correlations by chance. "Correlations by chance" refers to the fact, that noise induced spurious correlations between genes and class labels may appear for a small sample size if the level of noise is high. If the number of selected genes is small compared to the total number of probes (genes) on the chip, spurious correlations may become a serious problem. Monte-Carlo simulations of van't Veer et al. [2002] on randomly chosen expression values for a data set of 78 samples and 5000 genes resulted in 36 "genes" which had noise induced correlation coefficients larger than 0.3. In order to avoid large negative effects of above-mentioned spurious correlations the number of selected genes should not be too small, and one should extract a few tens of genes rather than a few genes only to decrease the influence of single spurious correlated genes. The random correlation effect can also be reduced, by increasing q_2, the minimum number of "reliable" expression values for making a present call. This avoids the selection of genes for which too few samples contribute to the correlation measure. However as explained in the next paragraph too many genes should be avoided as well.

Redundancy. Redundant sets of genes, that is sets of genes with correlated expression patterns should be avoided in order to obtain good machine learning results [Jäger et al., 2003]. Selection of too many genes with redundant information may lead to low generalization performance Another reason for avoiding redundancy is that not all causes which imply the conditions may be

recognized. This may happen if the set has to be kept small while redundant genes are included (redundant genes indicate the same cause). Reducing redundancy does not preclude the extraction of co-expressed clusters of genes: co-regulated genes can be extracted in a subsequent processing step, for example based on classical statistical analysis.

Finally, one may wonder why redundant information does not help to decrease the noise level of otherwise informative genes. Empirically one finds that non-redundant feature selection methods (P-SVM and R2W2) outperform feature selection methods which include redundant genes (Fisher correlation and RFE). It seems as if the detrimental effects of a larger number of features are stronger.

8.3.3 Classification of Samples

In order to construct a predictor for the class labels of new samples a classifier is trained on all the l samples using the optimal set of genes and the optimal value of the hyperparameter (here: ν, cf. Step 5). The generalization performance of the classifier can again be estimated using a cross-validation procedure. This procedure must involve performing the full gene selection procedure including all preprocessing steps (for example normalization and feature selection) separately on all l cross-validation subsets. Otherwise a bias is introduced in the estimate. Note that this also requires to perform the "inner loop" of Step 5 on sets of $(l-2)$ samples.

Before the classifier is applied to new data, the expression values for the new sample must be scaled according to the parameters derived from the training set. As a consequence we may observe expression values which are larger than the ones which occur in the training data. We set the expression values exceeding the maximal value in the training set to this maximal value. With this procedure we may underestimate certain expression levels but the robustness against unexpected deviations from the training data is increased.

Chapter 9

Hidden Markov Models

This is the first chapter devoted to unsupervised learning, especially to generative models. The most prominent generative model in bioinformatics is the hidden Markov model. It is well suited for analyzing protein or DNA sequences because of its discrete nature.

9.1 Hidden Markov Models in Bioinformatics

A hidden Markov model (HMM) is a generative approach for generating output sequences. The model is able to assign to each sequence a certain probability of being produced by the current model. The sequences of a class are used to build a model so that these sequences have high probability of being produced by the model. Thereafter the model can be utilized to search for sequences which also have high probability as being produces by the model. Therefore the new sequences with high probability are assumed to be similar to the sequences from which the model is build.

The HMM is able to model certain patterns in a sequences and if those patterns are detected, the probability of the sequence is increased.

HMMs for gene prediction.

The DNA is scanned and exons and introns are identified from which the coding region of the gene can be obtain. Translating the coding regions of the gene gives a protein sequences. HMMs are a standard tool for identifying genes in a genome. GENSCAN [Burge and Karlin, 1997] and other HMM approaches to gene prediction [Kulp et al., 1996, Krogh, 1997, Krogh et al., 1994b] have a base-pair specificity between 50% and 80%.

Profile HMMs.

Profile HMMs [Krogh et al., 1994a] are used to store a multiple alignment in a hidden Markov model. An HMM is better suited for storing the alignment than a consensus string because it is a generative model. Especially new sequences can be be evaluated according to their likelihood of being produced by the model. Also the likelihood can be fine tuned after storing the alignment.

If HMMs are build from unaligned sequences, they often stick in local likelihood maxima. Approaches exist which try to avoid them, e.g. deterministic annealing (“Userguide” to HMMER version 1.8). Because of the poor results with unaligned sequences despite annealing approaches, in the new version of HMMER the HMMs are only initialized by alignment results.

The most common software packages for profile HMMs are HMMER [Eddy, 1998] (`http://hmmer.wustl.edu/`) and SAM [Krogh et al., 1994a] (`http://www.cse.ucsc.edu/compbio/sam.html`).

However HMMs have drawbacks as Sean Eddy writes [Eddy, 2004]:

> "HMMs are reasonable models of linear sequence problems, but they don't deal well with correlations between residues or states, especially long-range correlations. HMMs assume that each residue depends only on one underlying state, and each state in the state path depends only on one previous state; otherwise, residues and states are independent of each other." ... "The state path of an HMM has no way of remembering what a distant state generated when a second state generates its residue."

Real valued protein characteristics like hydrophobic profiles cannot be represented by HMMs. HMMs cannot detect higher order correlations, cannot consider dependencies between regions in the sequence, cannot deal with correlations of elements within regions, cannot derive or process real valued protein characteristics, cannot take into account negative examples during model selection, and do not work sufficiently well for unaligned sequences.

Other HMMs Applications.

HMMs were used for remote homology detection [Park et al., 1998, Karplus et al., 1998, 1999] which is weak sequence homology [Sjölander et al., 1996].

HMMs were used for scoring [Barrett et al., 1997] and are combined with trees [Lio et al., 1999].

A whole data base is build on HMMs, namely the PFAM (protein family database) [Bateman et al., 2004, 2000]. Here protein families are classified by HMMs. Software exists for large data sets or proteins like SMART (simple modular architecture research tool) [Schultz et al., 2000].

9.2 Hidden Markov Model Basics

A hidden Markov model (HMM) is a graph of connected hidden states $u \in \{1, \ldots, S\}$, where each state produces a probabilistic output.

Fig. 9.1 shows a hidden Markov model with two state values 1 and 0 which are associated with "on" and "off". If the switch is "on" then it can remain on or going to the value "off". If the switch is "off" then it can remain off or going to the value "on". The state may be hidden in the sense that the position of switch cannot be observed.

The model evolves over time t (in bioinformatics time is replaced by sequence position). At each step the process jumps from the current state into another state or remains in the current state. The evolving of the state variable u over time can be expressed by introducing the variable u_t for each time point t. At each time t the variable u_t has a certain value $u_t \in \{1, \ldots, S\}$. Fig. 9.2 shows a hidden Markov model where the state variable evolves over time.

It is possible to present all possible sequences of values of the hidden state like in Fig. 9.3. Each path in the Figure from left to right is a possible sequence of state values. The probability of taking a certain value at a certain time (e.g. $u_5 = 3$) is the sum over all path' from the left to this value and time.

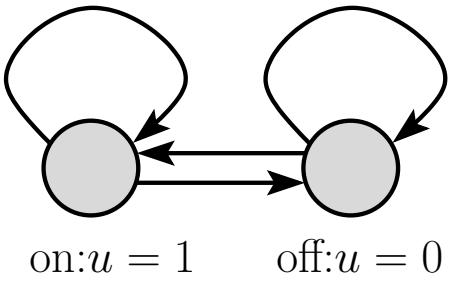


Figure 9.1: A simple hidden Markov model, where the state u can take on one of the two values 0 or 1. This model represents the switch for a light: it is either "on" or "off" and at every time point it can remain in its current state (reccurent connections) or go to the opposite state.

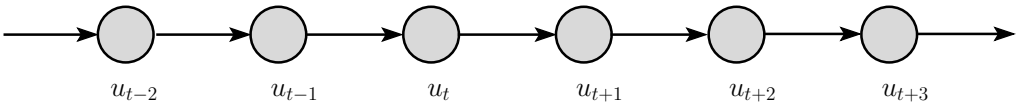


Figure 9.2: A simple hidden Markov model. The state u evolves over time and to each time t the state u takes on the value u_t.

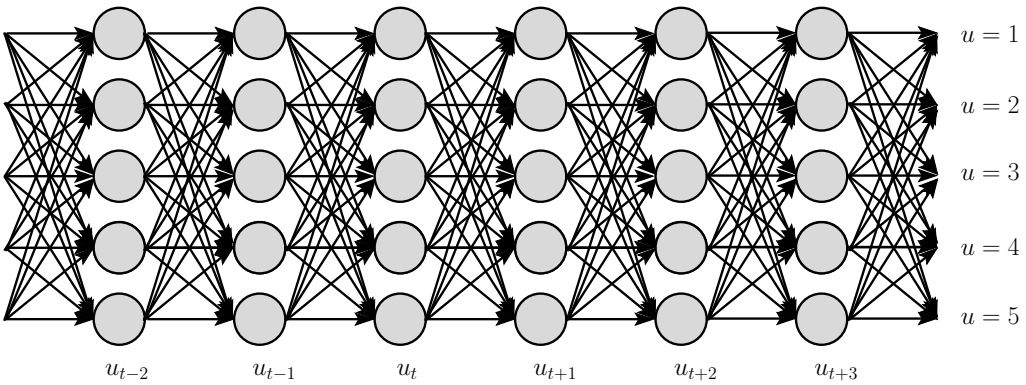


Figure 9.3: The hidden Markov model from Fig. 9.2 in more detail where also the state values $u = 1, \ldots, u = 5$ are given ($S = 5$). At each time t the state u_t takes on one of these values and if the state moves on the value may change. Each path from left to right has a certain probability and the probability of taking a certain value at a certain time is the sum over all path' from the left to this value and time.

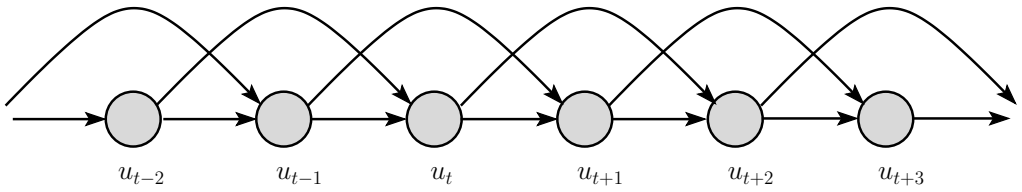


Figure 9.4: A second order hidden Markov model. The transition probability does not depend only on the current state value but also on the previous state value.

The hidden Markov model has transition probabilities $p(a \mid b)$, where $a, b \in \{1, \ldots, S\}$ and b is the current state and a the next state. Here the Markov assumption is that the next state only depends on the current state. Higher order hidden Markov models assume that the probability of going into the next state depends on the current state and previous states. For example in a second order Markov model the transition probability is $p(a \mid b, c)$, where $a, b, c \in \{1, \ldots, S\}$ and b is the current state, c the previous state, and a the next state. The second order Markov model is depicted in Fig. 9.4.

We will focus on a first order hidden Markov model, where the probability of going into a state depends only on the actual state.

At each time the state variable u takes on the value u_t and has a previous value given by u_{t-1}, therefore we observed the transition from u_{t-1} to u_t. This transition has probability of $p(u_t \mid u_{t-1})$,

Assume we have a certain start state probability $p_S(u_1)$ then the probability of observing the sequence $u^T = (u_1, u_2, u_3, \ldots, u_T)$ of length T is

$$p(u^T) \;=\; p_S(u_1) \prod_{t=2}^{T} p(u_t \mid u_{t-1}) \,. \tag{9.1}$$

For example a sequence may be $(u_1 = 3, u_2 = 5, u_3 = 2, \ldots, u_T = 4)$ that is the sequence $(3, 5, 2, \ldots, 4)$. Fig. 9.5 shows the hidden Markov model from Fig. 9.3 where now the transition probabilities are marked including the start state probability p_S.

Now we will consider Markov models which actually produce data, that means they are used as generative models. We assume that each state value has an emission probability $p_E(x_t \mid u_t)$ of emitting a certain output. Here u_t is a value of the state variable at time t (e.g. $u_t = 2$) and x_t is an element of the output alphabet of size P, for example $x_t \in \{A, T, C, G\}$. A specific emission probability may be $p_E(A \mid 2)$. Fig. 9.6 shows a hidden Markov model with output emission.

Fig. 9.7 shows a HMM where the output sequence is the Shine-Dalgarno pattern for ribosome binding regions.

Each output sequence has a certain probability of being produced by the current model. The joint probability of the output sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$ of length T and the hidden state value sequence $u^T = (u_1, u_2, u_3, \ldots, u_T)$ of length T is

$$p(u^T, x^T) \;=\; p_S(u_1) \prod_{t=2}^{T} p(u_t \mid u_{t-1}) \prod_{t=1}^{T} p_E(x_t \mid u_t) \,. \tag{9.2}$$

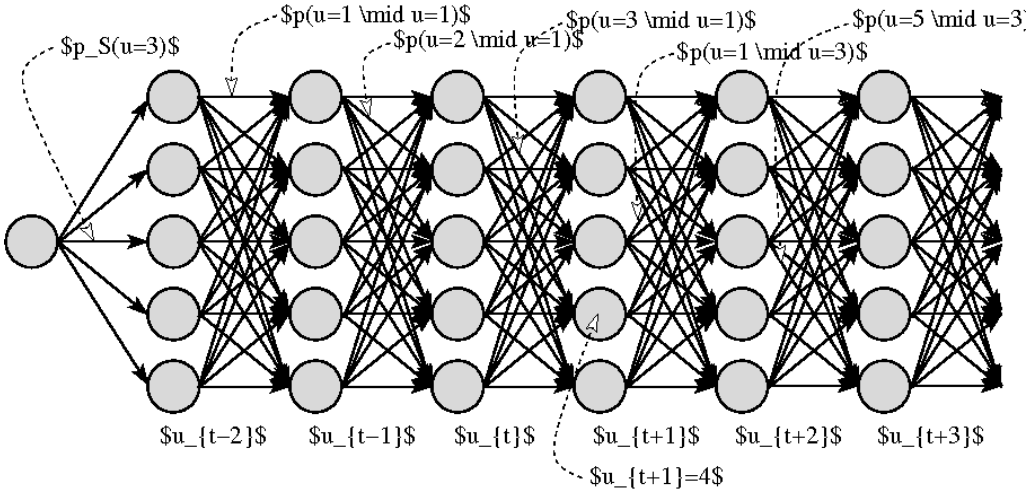


Figure 9.5: The hidden Markov model from Fig. 9.3 where now the transition probabilities are marked including the start state probability p_S. Also the state value $u_{t+1} = 4$ is marked.

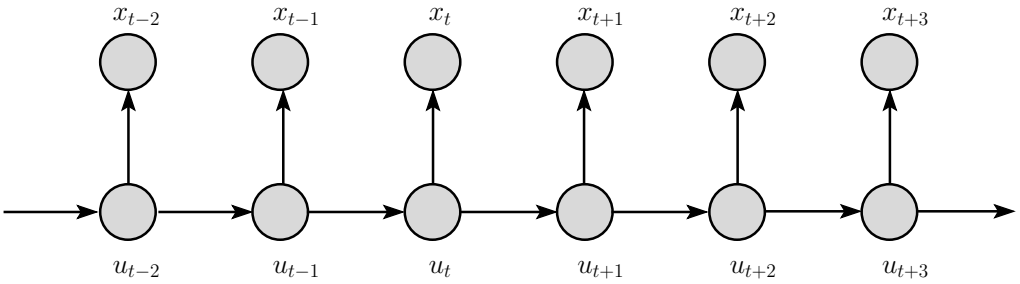


Figure 9.6: A simple hidden Markov model with output. At each time t the hidden state u_t has a certain probability of producing the output x_t.

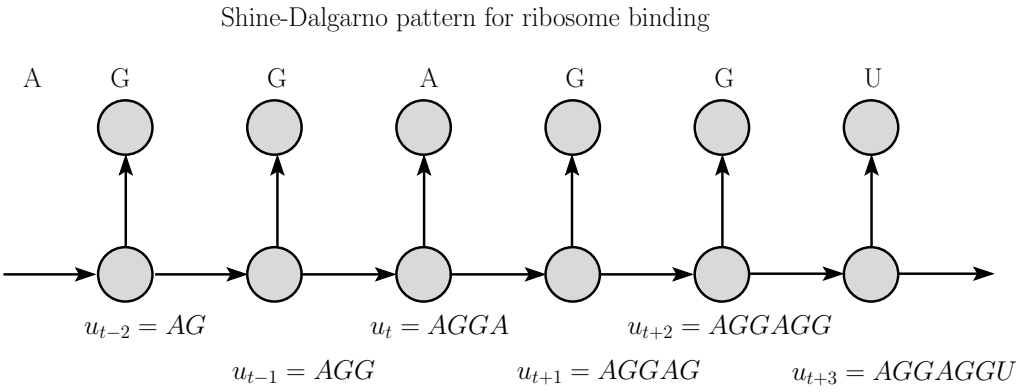


Figure 9.7: An HMM which supplies the Shine-Dalgarno pattern where the ribosome binds. Each state value is associated with a prefix sequence of the Shine-Dalgarno pattern. The state value for "no prefix" is omitted in the figure.

Through marginalization we obtain the probability of the output sequence x^T being produced by the HMM:

$$p(x^T) \;=\; \sum_{u^T} p(u^T, x^T) \;=\; \sum_{u^T} p_S(u_1) \prod_{t=2}^{T} p(u_t \mid u_{t-1}) \prod_{t=1}^{T} p_E(x_t \mid u_t) \;, \tag{9.3}$$

where $\sum_{u^T}$ denotes the sum over all possible sequences of length T of the values $\{1, \ldots, S\}$. The sum $\sum_{u^T}$ has S^T summands corresponding to different sequences (S values for u_1 multiplied by S values for u_2 etc.).

Fortunately, the (first order) Markov assumption allows to recursively compute above sum. We denote with $x^t = (x_1, x_2, x_3, \ldots, x_t)$ the prefix sequence of x^T of length t. We introduce the probability $p(x^t, u_t)$ of the model producing x^t and being in state u_t at the end.

$$\begin{aligned}
&p(x^t, u_t) \;=\; p(x_t \mid x^{t-1}, u_t)\, p(x^{t-1}, u_t) \;= \\
&p_E(x_t \mid u_t) \sum_{u_{t-1}} p(x^{t-1}, u_t, u_{t-1}) \;= \\
&p_E(x_t \mid u_t) \sum_{u_{t-1}} p(u_t \mid x^{t-1}, u_{t-1})\, p(x^{t-1}, u_{t-1}) \;= \\
&p_E(x_t \mid u_t) \sum_{u_{t-1}} p(u_t \mid u_{t-1})\, p(x^{t-1}, u_{t-1}) \;,
\end{aligned} \tag{9.4}$$

where the Markov assumptions $p(x_t \mid x^{t-1}, u_t) \;=\; p_E(x_t \mid u_t)$ on the output emission and $p(u_t \mid x^{t-1}, u_{t-1}) \;=\; p(u_t \mid u_{t-1})$ and on the transitions is used. Further marginalization $p(x^{t-1}, u_t) \;=\; \sum_{u_{t-1}} p(x^{t-1}, u_t, u_{t-1})$ and the definition of conditional probabilities $p(x^{t-1}, u_t, u_{t-1}) \;=\; p(u_t \mid x^{t-1}, u_{t-1})\, p(x^{t-1}, u_{t-1})$ was applied.

That means each recursion step needs only a sum over all u_{t-1} which is a sum over S values. However we have to do this for each value of u_t, therefore the recursion has complexity of $O(T\, S^2)$. The complexity can be reduced if transition probabilities are zero. The recursion starts with

$$p(x^1, u_1) \;=\; p_S(u_1)\, p_E(x_1 \mid u_1) \;. \tag{9.5}$$

The final probability of x^T can be computed as

$$p(x^T) \;=\; \sum_{u_T} p(x^T, u_T) \;. \tag{9.6}$$

This algorithm is called the "forward pass" or the "forward phase" and is used to compute the probability of x^T which is equal to the likelihood of x^T because we have discrete values. Alg. 9.1 shows the algorithm for the forward phase to compute the likelihood for one sequence for an HMM.

Algorithm 9.1 HMM Forward Pass

Given: sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$, state values $u \in \{1, \ldots, S\}$, start probabilities $p_S(u_1)$, transition probabilities $p(u_t \mid u_{t-1})$, and emission probabilities $p_E(x_t \mid u_t)$; Output: likelihood $p(x^T)$ and $p(x^t, u_t)$

BEGIN initialization

$$p(x^1, u_1) \;=\; p_S(u_1)\, p_E(x_1 \mid u_1)$$

END initialization
BEGIN Recursion
for $(t = 2\,;\; t \;\leq\; T\,;\; t++)$ **do**
for $(a = 1\,;\; a \;\leq\; S\,;\; a++)$ **do**

$$p(x^t, u_t = a) \;=\; p_E(x_t \mid u_t = a) \sum_{u_{t-1}=1}^{S} p(u_t = a \mid u_{t-1})\, p(x^{t-1}, u_{t-1})$$

end for
end for
END Recursion
BEGIN Compute Likelihood

$$p(x^T) \;=\; \sum_{a=1}^{S} p(x^T, u_T = a)$$

END Compute Likelihood

9.3 Expectation Maximization for HMM: Baum-Welch Algorithm

Now we focus on learning and parameter selection based on a training set.

The parameters of a hidden Markov model are the S start probabilities $p_S(u_1)$, the S^2 transitions probabilities $p(u_t \mid u_{t-1})$, and the $S\ P$ emission probabilities $p_E(x_t \mid u_t)$ (P is the number of output symbols).

If we have a set of training sequences $\{\boldsymbol{x}^i\}, 1 \leq i \leq l$, then the parameter can be optimized by maximizing the likelihood from Section 3.4.

Instead of gradient based methods from Chapter 5 we deduce a Expectation Maximization algorithm as shown in Subsection 3.4.6.

In Subsection 3.4.6 in eq. (3.77) we defined

$$\mathcal{F}(Q, \boldsymbol{w}) \ = \ \int_U Q(\boldsymbol{u} \mid \boldsymbol{x}) \ \ln p(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{w}) \ d\boldsymbol{u} \ - \tag{9.7}$$
$$\int_U Q(\boldsymbol{u} \mid \boldsymbol{x}) \ \ln Q(\boldsymbol{u} \mid \boldsymbol{x}) \ d\boldsymbol{u} \ ,$$

where $Q(\boldsymbol{u} \mid \boldsymbol{x})$ is an estimation for $p(\boldsymbol{u} \mid \boldsymbol{x}; \boldsymbol{w})$.

We have to adapt this formulation to discrete HMMs. For HMMs $\boldsymbol{u}$ is the sequence of hidden states, $\boldsymbol{x}$ the sequence of output states and $\boldsymbol{w}$ summarizes all probability parameters (start, transition, and emission) in the model. The integral $\int_U d\boldsymbol{u}$ can be replaced by a sum.

The estimation for the state sequence can be written as

$$Q(\boldsymbol{u} \mid \boldsymbol{x}) \ = \ p(u_1 = a_1, u_2 = a_2, \ldots, u_T = a_T \mid x^T; \boldsymbol{w}) \ . \tag{9.8}$$

We obtain

$$\mathcal{F}(Q, \boldsymbol{w}) \ = \ \sum_{a_1=1}^{S} \ldots \sum_{a_T=1}^{S} \tag{9.9}$$
$$p(u_1 = a_1, u_2 = a_2, \ldots, u_T = a_T \mid x^T; \boldsymbol{w}) \ \ln p(x^T, u^T; \boldsymbol{w}) \ -$$
$$\sum_{a_1=1}^{S} \ldots \sum_{a_T=1}^{S} p(u_1 = a_1, u_2 = a_2, \ldots, u_T = a_T \mid x^T; \boldsymbol{w})$$
$$\ln p(u_1 = a_1, u_2 = a_2, \ldots, u_T = a_T \mid x^T; \boldsymbol{w}) \ = \tag{9.10}$$
$$\sum_{a_1=1}^{S} \ldots \sum_{a_T=1}^{S} p(u_1 = a_1, u_2 = a_2, \ldots, u_T = a_T \mid x^T; \boldsymbol{w}) \ \ln p(x^T, u^T; \boldsymbol{w})$$
$$-c \ ,$$

where c is a constant independent of $\boldsymbol{w}$.

We have

$$\ln p(u^T, x^T; \boldsymbol{w}) \ = \ \ln p_S(u_1) \ + \ \sum_{t=2}^{T} \ln p(u_t \mid u_{t-1}) \ + \ \sum_{t=1}^{T} \ln p_E(x_t \mid u_t) \ . \tag{9.11}$$

Because most variables a_t can be summed out we obtain:

$$\begin{aligned}\mathcal{F}(Q, \boldsymbol{w}) \;=\; & \sum_{a=1}^{S} p(u_1 = a \mid x^T; \boldsymbol{w}) \;\ln p_S(u_1 = a) \;+ \\ & \sum_{t=1}^{T} \sum_{a=1}^{S} p(u_t = a \mid x^T; \boldsymbol{w}) \;\ln p_E(x_t \mid u_t = a) \;+ \\ & \sum_{t=2}^{T} \sum_{a=1}^{S} \sum_{b=1}^{S} p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w}) \;\ln p(u_t = a \mid u_{t-1} = b) \;- c\,.\end{aligned} \tag{9.12}$$

Note that the parameters $\boldsymbol{w}$ are the start probabilities $p_S(a)$, the emission probabilities $p_E(x \mid a)$, and the transition probabilities $p(a \mid b)$. We have as constraints $\sum_a p_S(a) \;=\; 1$, $\sum_x p_E(x \mid a) \;=\; 1$, and $\sum_a p(a \mid b) \;=\; 1$.

Consider the optimization problem

$$\begin{aligned}\min_{\boldsymbol{w}} \quad & \sum_t \sum_a c_{ta} \;\ln w_a \\ \text{s.t.} \quad & \sum_a w_a \;=\; 1\,.\end{aligned} \tag{9.13}$$

The Lagrangian is

$$L \;=\; \sum_t \sum_a c_{ta} \;\ln w_a \;-\; \lambda \left(\sum_a w_a \;-\; 1 \right)\,. \tag{9.14}$$

Optimality requires

$$\frac{\partial L}{\partial w_a} \;=\; \sum_t c_{ta} \, \frac{1}{w_a} \;-\; \lambda \;=\; 0 \tag{9.15}$$

therefore

$$\sum_t c_{ta} \;-\; \lambda \, w_a \;=\; 0 \tag{9.16}$$

and summing over a gives

$$\sum_a \sum_t c_{ta} \;=\; \lambda\,. \tag{9.17}$$

We obtain

$$w_a \;=\; \frac{\sum_t c_{ta}}{\sum_a \sum_t c_{ta}}\,. \tag{9.18}$$

The constraint maximization step (M-step) is therefore

$$p_S(a) = \frac{p(u_1 = a \mid x^T; \boldsymbol{w})}{\sum_{a'} p(u_1 = a' \mid x^T; \boldsymbol{w})} \tag{9.19}$$

$$p_E(x \mid a) = \frac{\sum_{t=1}^{T} \delta_{x_t=x}\, p(u_t = a \mid x^T; \boldsymbol{w})}{\sum_y \sum_{t=1}^{T} \delta_{x_t=y}\, p(u_t = a \mid x^T; \boldsymbol{w})} \tag{9.20}$$

$$p(a \mid b) = \frac{\sum_{t=2}^{T} p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w})}{\sum_{a'} \sum_{t=2}^{T} p(u_t = a', u_{t-1} = b \mid x^T; \boldsymbol{w})} \tag{9.21}$$

which is

$$p_S(a) = p(u_1 = a \mid x^T; \boldsymbol{w}) \tag{9.22}$$

$$p_E(x \mid a) = \frac{\sum_{t=1}^{T} \delta_{x_t=x}\, p(u_t = a \mid x^T; \boldsymbol{w})}{\sum_{t=1}^{T} p(u_t = a \mid x^T; \boldsymbol{w})} \tag{9.23}$$

$$p(a \mid b) = \frac{\sum_{t=2}^{T} p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w})}{\sum_{t=2}^{T} p(u_{t-1} = b \mid x^T; \boldsymbol{w})} \,. \tag{9.24}$$

We now consider the estimation step (E-step) in order to estimate $p(u_t = a \mid x^T; \boldsymbol{w})$ and $p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w})$. First we have to introduce the suffix sequence $x^{t \leftarrow T} = (x_t, x_{t+1}, \ldots, x_T)$. of length $T - t + 1$.

We use the probability $p(x^{t+1 \leftarrow T} \mid u_t = a)$ of the suffix sequence $x^{t+1 \leftarrow T} = (x_{t+1}, \ldots, x_T)$ being produced by the model if starting from $u_t = a$.

Now we can formulate an expression for $p(u_t = a \mid x^T; \boldsymbol{w})$

$$p(u_t = a \mid x^T; \boldsymbol{w}) = \frac{p(u_t = a, x^T; \boldsymbol{w})}{p(x^T)} = \tag{9.25}$$
$$\frac{p(x^t, u_t = a; \boldsymbol{w})\; p(x^{t+1 \leftarrow T} \mid u_t = a)}{p(x^T)} \,,$$

where the first "=" is the definition of conditional probability and the second "=" say that all path' of hidden values with have at time t the value a can be separated into a prefix path from start to time t ending in the value a and a suffix path starting at time t in a.

Similar we can formulate an expression for $p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w})$

$$p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w}) = \frac{p(u_t = a, u_{t-1} = b, x^T; \boldsymbol{w})}{p(x^T)} = \tag{9.26}$$
$$p(x^{t-1}, u_{t-1} = b; \boldsymbol{w})\; p(u_t = a \mid u_{t-1} = b)\; p_E(x_t \mid u_t = a)$$
$$p(x^{t+1 \leftarrow T} \mid u_t = a) \;/\; p(x^T) \,,$$

where again the first "=" is the conditional probability and the second "=" says all path' which are at time t in state value a and in time $(t-1)$ in state value b can be separated in a prefix path from start to time $(t-1)$ ending in b, a suffix path starting from t in value a to the end, the transition $b \leftarrow a$ with probability $p(u_t = a \mid u_{t-1} = b)$ and the emission of x_t given by $p_E(x_t \mid u_t = a)$.

Note that

$$p(u_t = a \mid x^T; \boldsymbol{w}) \;=\; \sum_{b=1}^{S} p(u_t = a, u_{t-1} = b \mid x^T; \boldsymbol{w}) \;. \tag{9.27}$$

Similar to eq. (9.4) we can derive a backward recursion for computing $p(x^{t+1\leftarrow T} \mid u_t = a)$ by using the Markov assumptions:

$$\begin{aligned} &p(x^{t+1\leftarrow T} \mid u_t = a) \;= \\ &\sum_{b=1}^{S} p_E(x_{t+1} \mid u_{t+1} = b)\, p(u_{t+1} = b \mid u_t = a)\, p(x^{t+2\leftarrow T} \mid u_{t+1} = b) \;. \end{aligned} \tag{9.28}$$

The starting conditions are

$$p(x^{T\leftarrow T} \mid u_{T-1} = a) \;=\; \sum_{b=1}^{S} p_E(x_T \mid u_T = b)\, p(u_T = b \mid u_{T-1} = a) \tag{9.29}$$

or, alternatively

$$\forall_a : \;\; p(x^{T+1\leftarrow T} \mid u_T = a) \;=\; 1 \tag{9.30}$$

In Alg. 9.2 an algorithm for the backward procedure for HMMs is given.

The EM algorithm for HMMs is given Alg. 9.3 which is based on the forward procedure Alg. 9.1 and the backward procedure Alg. 9.2.

9.4 Viterby Algorithm

In the forward (and also backward) pass we computed $p(x^T)$, the probability of producing x^T by the model, that is the likelihood of x^T. The likelihood of x^T is an integral – more exactly a sum – over all probabilities of possible sequences of hidden states multiplied by the probability that the hidden sequence emits x^T.

In many cases a specific hidden sequence $(u^T)^* = (u_1^*, u_2^*, u_3^*, \ldots, u_T^*)$ and its probability of emitting x^T dominates the above sum. More formally

$$(u^T)^* \;=\; \arg\max_{u^T} p(u^T \mid x^T) \;=\; \arg\max_{u^T} p(u^T, x^T) \;. \tag{9.31}$$

$(u^T)^*$ is of interest if the hidden states have a semantic meaning, then one want to extract $(u^T)^*$.

In bioinformatics the extraction of $(u^T)^*$ is important to make an alignment of a sequence with a multiple alignment stored in an HMM.

Algorithm 9.2 HMM Backward Pass

Given: sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$, state values $u \in \{1, \ldots, S\}$, start probabilities $p_S(u_1)$, transition probabilities $p(u_t \mid u_{t-1})$, and emission probabilities $p_E(x_t \mid u_t)$; Output: likelihood $p(x^T)$ and $p(x^{t+1 \leftarrow T} \mid u_t = a)$

BEGIN initialization

$$\forall_a : \; p(x^{T+1 \leftarrow T} \mid u_T = a) \; = \; 1$$

END initialization
BEGIN Recursion
for $(t = T - 1 \; ; \; t \; \geq \; 1 \; ; \; t - -)$ **do**
for $(a = 1 \; ; \; a \; \leq \; S \; ; \; a + +)$ **do**

$$p(x^{t+1 \leftarrow T} \mid u_t = a) \; = \; \sum_{b=1}^{S} p_E(x_{t+1} \mid u_{t+1} = b) \, p(u_{t+1} = b \mid u_t = a) \, p(x^{t+2 \leftarrow T} \mid u_{t+1} = b) \; .$$

end for
end for
END Recursion
BEGIN Compute Likelihood

$$p(x^T) \; = \; \sum_{a=1}^{S} p_S(u_1 = a) \; (p(x^{1 \leftarrow T} \mid u_1 = a)$$

END Compute Likelihood

Algorithm 9.3 HMM EM Algorithm

Given: l training sequences $(x^T)^i = (x_1^i, x_2^i, x_3^i, \ldots, x_T^i)$ for $1 \leq i \leq l$, state values $u \in \{1, \ldots, S\}$, start probabilities $p_S(u_1)$, transition probabilities $p(u_t \mid u_{t-1})$, and emission probabilities $p_E(x \mid u)$; Output: updated values of $p_S(u)$, $p_E(x \mid u)$, and $p(u_t \mid u_{t-1})$

BEGIN initialization

initialize start probabilities $p_S(u_1)$, transition probabilities $p(u_t \mid u_{t-1})$, and emission probabilities $p_E(x \mid u)$; Output: updated values of $p_S(u)$, $p_E(x \mid u)$, and $p(u_t \mid u_{t-1})$

END initialization

Stop=false
while Stop=false **do**
for $(i = 1 \,;\, i \geq l \,;\, i++)$ **do**

Forward Pass

forward pass for $(x^T)^i$ according to Alg. 9.1

Backward Pass

backward pass for $(x^T)^i$ according to Alg. 9.2

E-Step

for $(a = 1 \,;\, a \leq S \,;\, a++)$ **do**
for $(b = 1 \,;\, b \leq S \,;\, b++)$ **do**

$$p(u_t = a, u_{t-1} = b \mid (x^T)^i; \boldsymbol{w}) = p((x^{t-1})^i, u_{t-1} = b; \boldsymbol{w})\, p(u_t = a \mid u_{t-1} = b)\, p_E(x_t^i \mid u_t = a)\, p((x^{t+1 \leftarrow T})^i \mid u_t = a) \,/\, p((x^T)^i)$$

end for
end for
for $(a = 1 \,;\, a \leq S \,;\, a++)$ **do**

$$p(u_t = a \mid (x^T)^i; \boldsymbol{w}) = \sum_{b=1}^{S} p(u_t = a, u_{t-1} = b \mid (x^T)^i; \boldsymbol{w})$$

end for

M-Step

for $(a = 1 \,;\, a \leq S \,;\, a++)$ **do**

$$p_S(a) = p(u_1 = a \mid (x^T)^i; \boldsymbol{w})$$

end for
for $(a = 1 \,;\, a \leq S \,;\, a++)$ **do**
for $(x = 1 \,;\, x \leq P \,;\, x++)$ **do**

$$p_E(x \mid a) = \frac{\sum_{t=1}^{T} \delta_{x_t^i = x}\, p(u_t = a \mid (x^T)^i; \boldsymbol{w})}{\sum_{t=1}^{T} p(u_t = a \mid (x^T)^i; \boldsymbol{w})}$$

end for
end for
for $(a = 1 \,;\, a \leq S \,;\, a++)$ **do**
for $(b = 1 \,;\, b \leq S \,;\, b++)$ **do**

$$p(a \mid b) = \frac{\sum_{t=2}^{T} p(u_t = a, u_{t-1} = b \mid (x^T)^i; \boldsymbol{w})}{\sum_{t=2}^{T} p(u_{t-1} = b \mid (x^T)^i; \boldsymbol{w})}$$

end for
end for
end for
if stop criterion fulfilled **then**
Stop=true
end if
end while

Because $(u^T)^*$ can be viewed as an alignment, it is not surprising that it can be obtained through dynamic programming. The dynamic programming algorithm has to find a path in Fig. 9.5 from left to right. Towards this end the state values at certain time which are the circles in Fig. 9.5 are represented by a matrix $\boldsymbol{V}$. $V_{t,a}$ contains the maximal probability of a sequence of length t ending in state value a:

$$V_{t,a} \;=\; \max_{u^{t-1}} p(x^t, u^{t-1}, u_t = a)\;. \tag{9.32}$$

The Markov conditions allow now to formulate $V_{t,a}$ recursively:

$$V_{t,a} \;=\; p_E(x_t \mid u_t = a)\;\max_b p(u_t = a \mid u_{t-1} = b)\;V_{t-1,b} \tag{9.33}$$

with initialization

$$V_{1,a} \;=\; p_S(a) p_E(x_1 \mid u_1 = a) \tag{9.34}$$

and the result

$$\max_{u^T} p(u^T, x^T) \;=\; \max_a V_{T,a}\;. \tag{9.35}$$

The best sequence of hidden states can be found by back-tracing using

$$b(t,a) \;=\; \arg\max_b p(u_t = a \mid u_{t-1} = b)\;V_{t-1,b} \tag{9.36}$$

The complexity of the Viterby algorithm is $O(T\;S^2)$ because all $S\;T$ values $V_{t,a}$ must be computed and for computing them, the maximum over S terms must be determined.

The Viterby algorithm can be used to iteratively improve a multiple alignment:

1 initialize the HMM

2 align all sequences to the HMM via the Viterby algorithm

3 make frequency counts per column and compute the transition probabilities to update the HMM

4 if not converged go to step 2

9.5 Input Output Hidden Markov Models

Input Output Hidden Markov Models (IOHMMs) generates an output sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$ of length T conditioned on an input sequence $y^T = (y_1, y_2, y_3, \ldots, y_T)$ of length T.

The difference between standard HMMs and input output HMMs is that the probabilities are conditioned on the input. Start probabilities are $p_S(u_1 \mid y_1)$, the transition probabilities $p(u_t \mid y_t, u_{t-1})$, and the emission probabilities $p_E(x_t \mid y_t, u_t)$.

Algorithm 9.4 HMM Viterby

Given: sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$, state values $u \in \{1, \ldots, S\}$, start probabilities $p_S(u_1)$, transition probabilities $p(u_t \mid u_{t-1})$, and emission probabilities $p_E(x_t \mid u_t)$; Output: most likely sequence of hidden state values $(u^T)^*$ and its probability $p\left(x^T, (u^T)^*\right)$

BEGIN initialization

$$V_{1,a} = p_S(a) p_E(x_1 \mid u_1 = a)$$

END initialization
BEGIN Recursion
for $(t = 2\ ;\ t \leq T\ ;\ t++)$ **do**
for $(a = 1\ ;\ a \leq S\ ;\ a++)$ **do**

$$V_{t,a} = p_E(x_t \mid u_t = a) \max_b p(u_t = a \mid u_{t-1} = b)\, V_{t-1,b}$$

$$b(t, a) = \arg\max_b p(u_t = a \mid u_{t-1} = b)\, V_{t-1,b}$$

end for
end for
END Recursion
BEGIN Compute Probability

$$p\left(x^T, (u^T)^*\right) = \max_{a=1}^{S} V(T, a)$$

END Compute Probability
BEGIN Back-tracing

$$s = \arg\max_{a=1}^{S} V(T, a)$$

print s
for $(t = T\ ;\ t \geq 2\ ;\ t--)$ **do**

$$s = b(t, s)$$

print s
end for
END Back-tracing

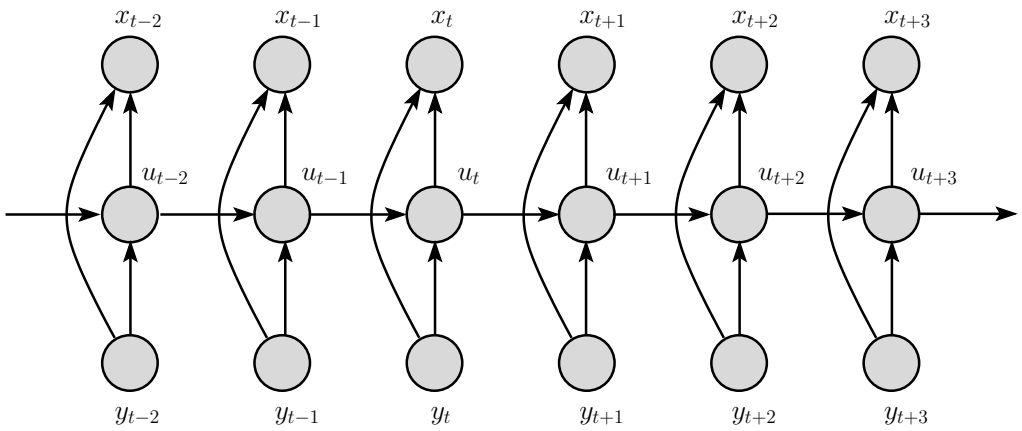


Figure 9.8: An input output HMM (IOHMM) where the output sequence $x^T = (x_1, x_2, x_3, \ldots, x_T)$ is conditioned on the input sequence $y^T = (y_1, y_2, y_3, \ldots, y_T)$.

Using IOMMs also negative examples can be used by setting for all y_t either a don't care or a fixed value and setting $y_T = 1$ for the positive class and $y_T = -1$ for the negative class. Whether a model for the negative class can be build is not clear but at least a subclass of the negative class which is very similar to the positive class can be better discriminated.

The number of parameters increase proportional to the number of input symbols, which may make it more difficult to estimate the probabilities if not enough data is available.

Learning via the likelihood is as with the standard HMM with the probabilities additionally conditioned on the input.

9.6 Factorial Hidden Markov Models

The HMM architecture Fig. 9.6 is extended to Fig. 9.9 where the hidden state is divided into more components u_i (three in the figure).

The transition probability of u_i is conditioned on all u_k with $k \leq i$ and the emission probability depends on all hidden states. The HMM architecture in Fig. 9.9 is meant that u_1 evolves very slowly, u_2 evolves faster, and u_3 evolves fastest of all hidden variables. Fast evolving variables do not influence slow evolving ones but slow evolving variables influence fast evolving variables.

If the factorial HMM has h hidden state variables u_i and each one of them can take on S values then the emission probability distribution consists of $P\ S^h$ emission probabilities. Therefore learning factorial HMMs is computational expensive. However approximative methods have been developed to speed up learning [Ghahramani and Jordan, 1996, 1997].

9.7 Memory Input Output Factorial Hidden Markov Models

Remember that we quoted Sean Eddy [Eddy, 2004]:

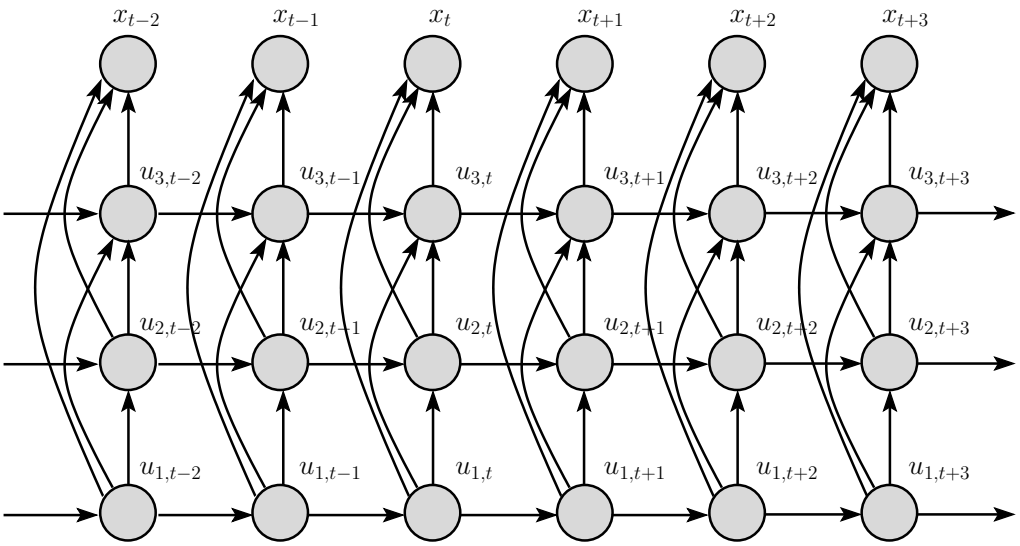


Figure 9.9: A factorial HMM with three hidden state variables u_1, u_2, and u_3. The transition probability of u_i is conditioned on all u_k with $k \leq i$ and the emission probability depends on all hidden states.

> "HMMs are reasonable models of linear sequence problems, but they don't deal well with correlations between residues or states, especially long-range correlations. HMMs assume that each residue depends only on one underlying state, and each state in the state path depends only on one previous state; otherwise, residues and states are independent of each other." ... "The state path of an HMM has no way of remembering what a distant state generated when a second state generates its residue."

The only way a HMM can store information over time is to go into a certain state value and don't change it any more. The state is fixed and the event which led the HMM enter the fixed state is memorized. Instead of a state a set of states can be entered from which the escape probability is zero.

To realize a state with a non-escaping value which can memorize past events is

$$p(u_t = a \mid u_{t-1} = a) \;=\; 1 \;.$$

That means if the state takes on the value a then the state will not take any other value.

In principle the storage of past events can be learned but the likelihood of storing decreases exponentially with the time of storage. Therefore learning to store is practically impossible because these small likelihood differences are tiny in comparison to local minima resulting from certain input / output pattern or input / output distribution.

Therefore memory is enforced by setting $p(u_t = a \mid u_{t-1} = a) \;=\; 1$ and to allowing this probability to change.

However after the storage process (taking on the value a) the model is fixed and neither future systems dynamics nor other events to memorize can be dealt with.

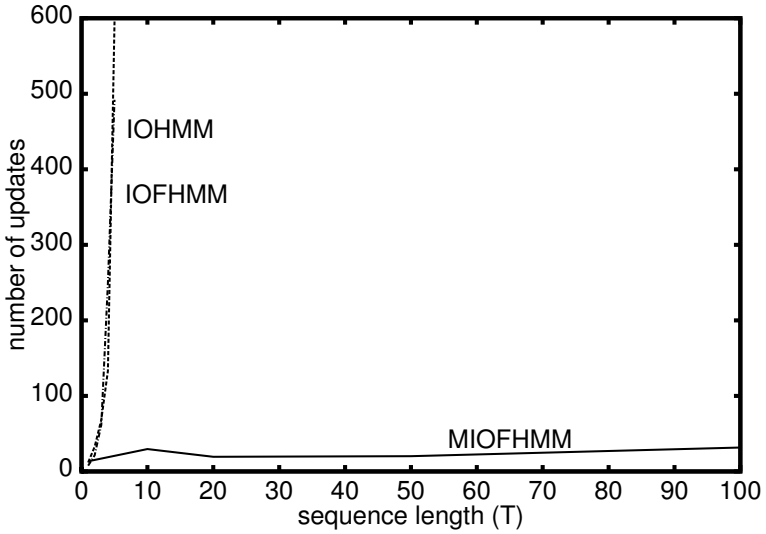


Figure 9.10: Number of updates required to learn to remember an input element until sequence end for three models: input output HMM (IOHMM), input output factorial HMM (IOFHMM), and "Memory-based Input-Output Factorial HMM" (MIOFHMM) as a function of the sequence length T.

To overcome this problem a factorial HMM can be used where some of the hidden state variables can store information and others extract the dynamics of the system to model.

Storing events is especially suited for input output HMMs where input events can be stored.

An architecture with memory state variable and using the input output architecture is the "Memory-based Input-Output Factorial HMM" (MIOFHMM, [Hochreiter and Mozer, 2001]).

Initially, all state variables have "uncommitted" values then various inputs can trigger the memory state variables to take on values from which the state variables cannot escape – they behave as a memory for the occurrence of an input event. Fig. 9.10 shows the number of updates required to train three models: input output HMM (IOHMM), input output factorial HMM (IOFHMM), and "Memory-based Input-Output Factorial HMM" (MIOFHMM) as a function of the sequence length T.

9.8 Tricks of the Trade

- Sometimes the HMM and its algorithms must be adjusted for bioinformatics applications for example to handle delete states which do not emit symbols in the forward pass.
- HMMs can be used for variable length of the sequences; however care must be take if comparing likelihoods because there are more longer sequences than shorter and the likelihood decreases exponentially with the length
- To deal with small likelihood and probability values is is recommended to compute the values in the log-space

- To avoid zero probabilities for certain sequences which makes certain minima unreachable all probabilities can be kept above a threshold ϵ.
- The EM-algorithm cannot reach probabilities which are exact zero, therefore, as an after-learning postprocessing all small probabilities $\leq \epsilon$ can be set to zero. This often helps to generalize from short to very long sequences.
- HMM are prone to local minima, for example if HMMs are build from unaligned sequences. Global optimization strategies try to avoid theses minima, e.g. deterministic annealing was suggested in the "Userguide" to HMMER version 1.8.

9.9 Profile Hidden Markov Models

Profile Hidden Markov Models code a multiple sequence alignment into a HMM as a position-specific scoring system which can be used to search databases for remote homologous sequences. Fig. 9.11 shows a HMM which can be used for homology search. The top row with states indicated with circles are a pattern. The diamond states are inserted strings. The bottom row with states indicated as squares are deletions, where a letter from the pattern is skipped.

To learn an HMM from a set of unaligned positive examples suffers from the problem of local minima. Therefore expensive global optimization strategies must be used to avoid theses minima, e.g. deterministic annealing was suggested in the "Userguide" to HMMER version 1.8. Therefore in most applications an HMM is at least initialized by a multiple alignment of the positive examples.

The use of profile HMMs was made very convenient by the free HMMER package by Sean Eddy [Eddy, 1998] which allows to to build and apply HMMs. HMMER supplies a log-odds likelihood of the model compared to a random model to access the significance of the score of a new sequence. Fig. 9.12 shows the architecture of the models used by HMMER. The states indicated by squares and denoted by "Mx" are the consensus string. The circled states denoted by "Dx" are deletion states (non-emitting states), where part of the consensus string can be skipped. The diamond states denoted by "Ix" are insertion states where a substring can be inserted into the consensus string

The other package which enabled a convenient use of HMMs for biological sequences is Sequence Alignment and Modeling system (SAM – `http://www.cse.ucsc.edu/research/compbio/sam.html`) which enables for creating, refining, and using HMMs for biological sequence analysis. Also the SAM models represent a refinement of a multiple alignment. Models can be used to both generate multiple alignments and search databases for new members of the family.

Also databases like Protein FAMily database (Pfam) are based on HMMs. 67% of proteins contain at least one Pfam profile HMM and 45% of residues in the protein database are covered in total by the HMMs.

Another HMM application which is not associated with profile HMMs is shown in Fig. 9.13, where the HMM is used for splice site detection.

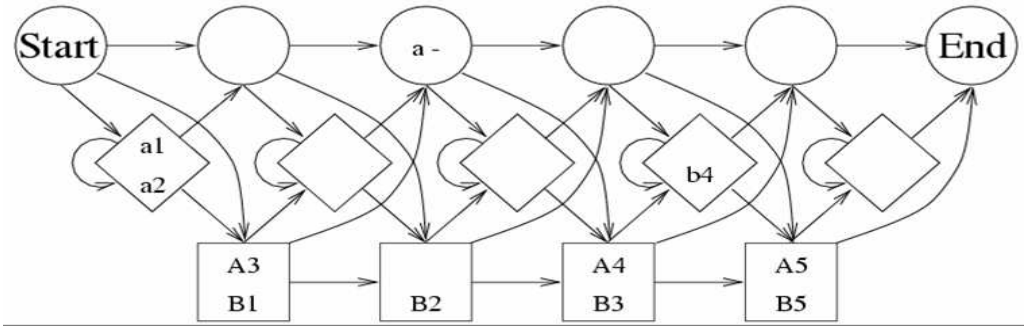


Figure 9.11: Hidden Markov model for homology search. The top row with states indicated with circles are a pattern. The diamond states are inserted strings. The bottom row with states indicated as squares are deletions, where a letter from the pattern is skipped.

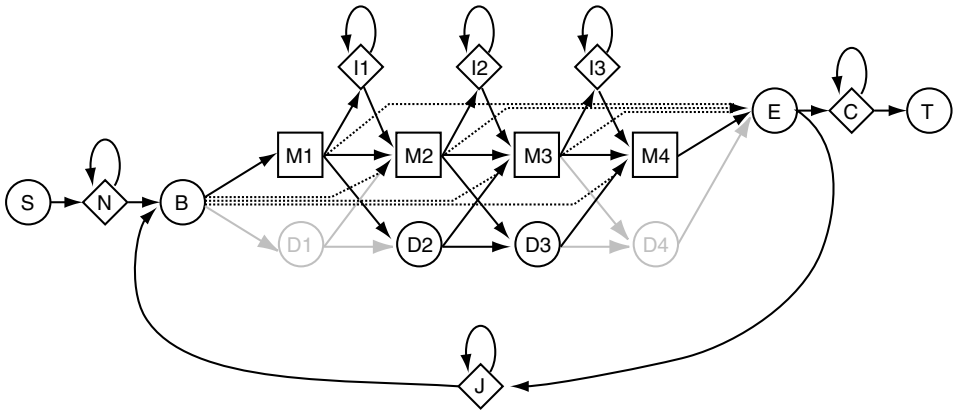


Figure 9.12: The HMMER hidden Markov architecture. The states indicated by squares and denoted by "Mx" form a pattern (consensus string). The circled states denoted by "Dx" are deletion states (non-emitting), where a letter from the pattern can be skipped. The diamond states denoted by "Ix" are insertion states where a substring between letters of the pattern has been inserted. "B" and "E" denote the begin and end state of the pattern, respectively.

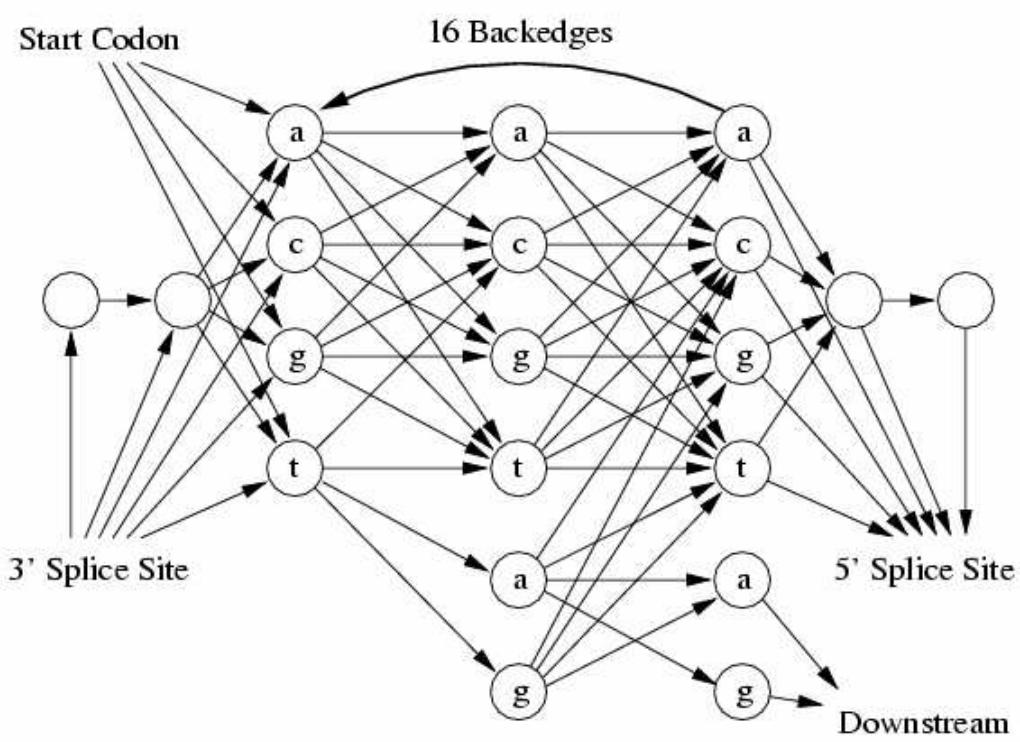


Figure 9.13: An HMM for splice site detection.

Chapter 10

Unsupervised Learning: Projection Methods and Clustering

10.1 Introduction

10.1.1 Unsupervised Learning in Bioinformatics

We have already seen an application of an unsupervised method in bioinformatics: the hidden Markov model. Hidden Markov models are used for sequence analysis.

One important topic in bioinformatics is to analyze microarray measurements with unsupervised techniques. Goals are to figure out which genes are expressed simultaneously (are part of the same pathway), what metabolic stages are present, etc. Also time series of microarray data must be analyzed e.g. by cluster analysis (e.g. [Eisen et al., 1998]). Fig. 10.1 and Fig. 10.2 show microarray dendrograms obtained through hierarchical clustering.

Often unsupervised methods are used in bioinformatics to visualize dependencies and clusters like the use of principal component analysis for Spellman's cell-cycle data in Fig. 10.3.

For visualization the data must be in general down-projected to show the data in a 2-dimensional or 3-dimensional space.

As we have already explained in Chapter 8 on feature selection, dimension reduction is an important step in preprocessing biological data. In Chapter 8 we reported that with mass spectrometry the data is huge and principal component analysis was used to reduce the input dimension. For example [Lilien et al., 2003] achieved best results on classification of healthy and cancerous persons of prostate Cancer on the basis surface-enhanced laser desorption/ionization time-of-flight mass spectrometry (SELDI TOF MS) date if PCA was used.

10.1.2 Unsupervised Learning Categories

Many unsupervised learning procedures can be viewed as trying to bring two probability distributions into alignment. Two well known classes of unsupervised procedures that can be cast in this manner are *generative* and *recoding* models.

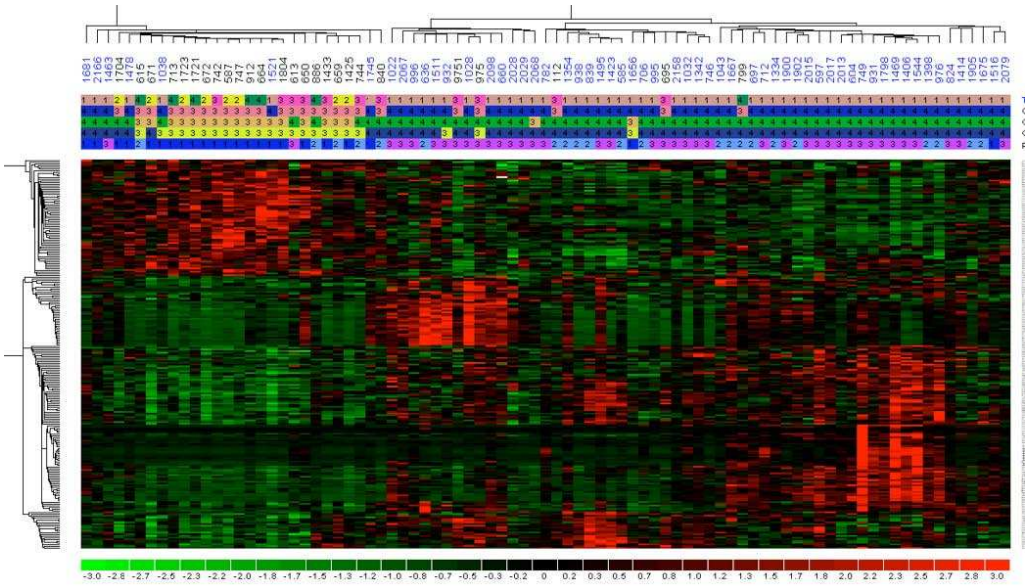


Figure 10.1: A microarray dendrogram obtained by hierarchical clustering.

10.1.2.1 Generative Framework

In a generative unsupervised framework (see Fig. 10.4), the environment generates training examples – which we will refer to as *observations* or training data – by sampling from one distribution; the other distribution is embodied in the model. In the generative framework we want to model or to simulate the real world by generating samples with the same underlying distribution as the real world samples.

The hidden Markov models from previous Chapter 9 fall into the generative framework. Other examples of the generative framework are factor analysis [Jöreskog, 1967, Everitt, 1984, Neal and Dayan, 1997], Boltzmann machines [Hinton and Sejnowski, 1986], or mixture models. In the generative framework we have to bring two distributions to match: the fixed distribution of observations and the model output distribution.

10.1.2.2 Recoding Framework

In the recoding unsupervised framework (see Fig. 10.5), the model transforms observations to an output space. The distribution of the transformed observations (the outputs) is compared either to a reference (target) distribution or whether target distribution features are present. These features are measured by an objective function which must be optimized. Target features are used to represent a whole class of target distributions. In many cases the objective function can be replaced by the distribution from the target distribution set which is closest to the output distribution.

Goal of the recoding framework is to represent the observations in an appropriate way, e.g. desired data density, low dimensional representation, high variance (large information), non-Gaussianity, or independent components.

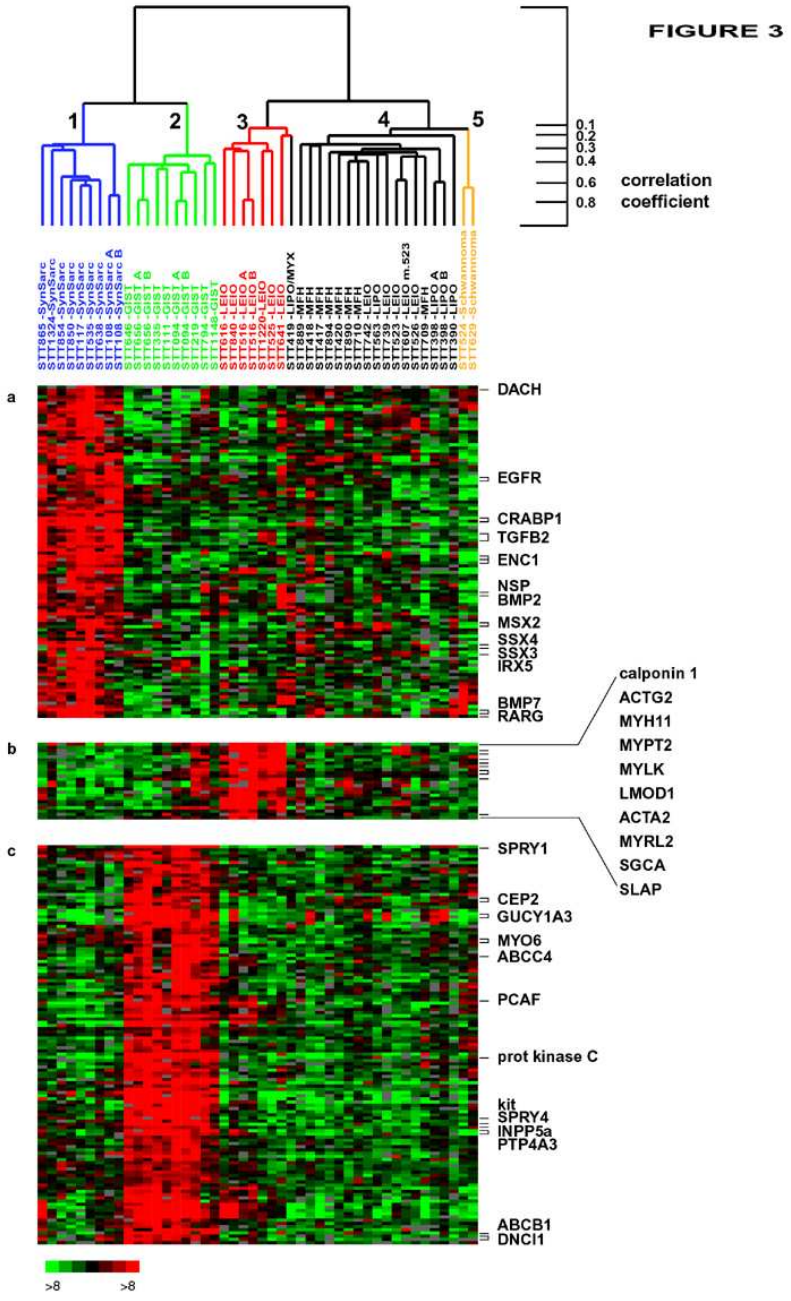


Figure 10.2: Another example of a microarray dendrogram obtained by hierarchical clustering.

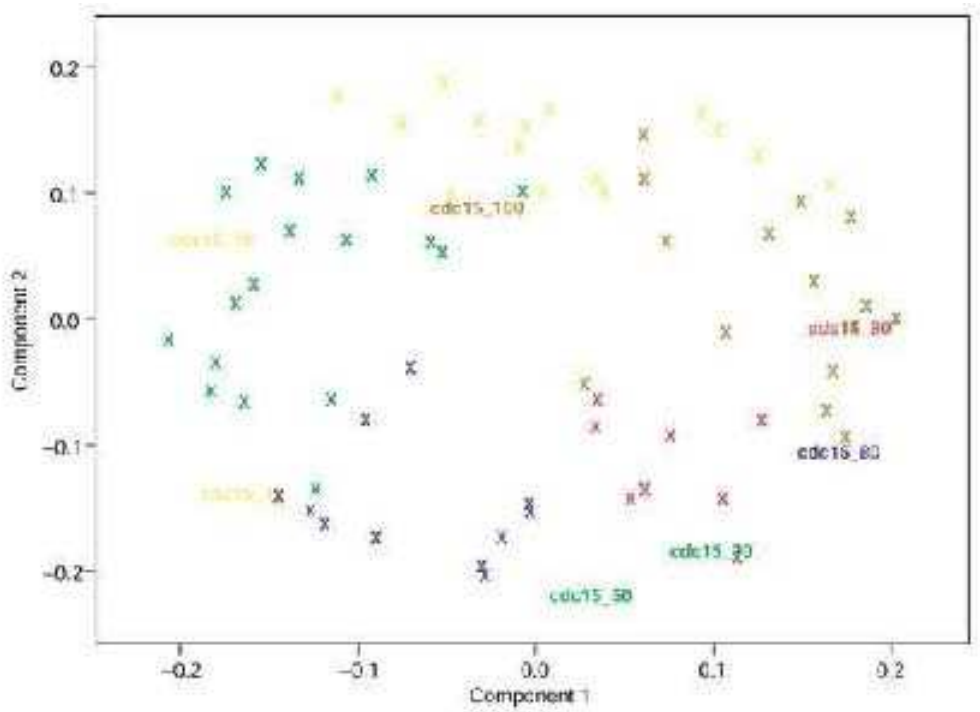


Figure 10.3: Spellman's cell-cycle data represented through the first principal components (Landgrebe et al., Genome Biology, 2002).

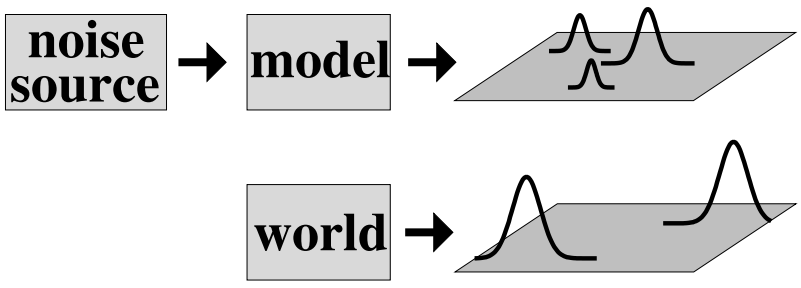


Figure 10.4: The generative framework is depicted. A noise source "drives" the model and produces an output distribution which should match the distribution observed in the world. Separation of model and noise source means that all adjustable parameters are contained in the model.

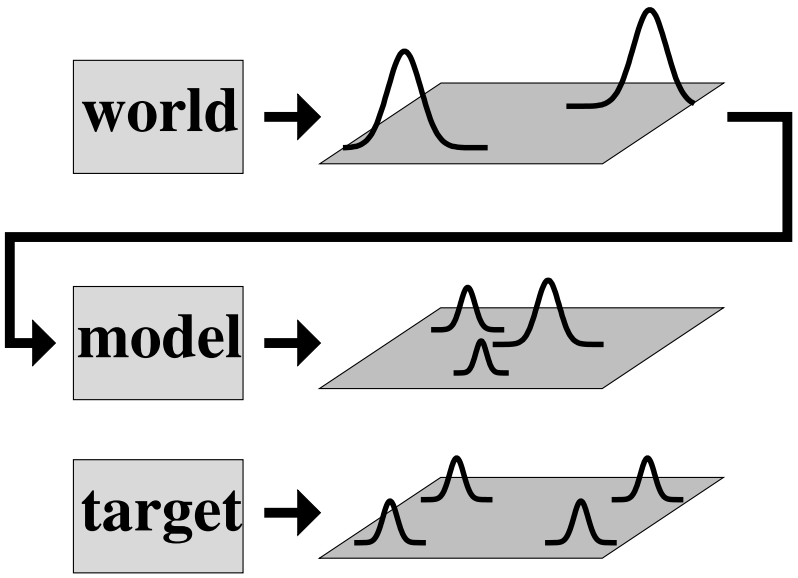


Figure 10.5: The recoding framework is depicted. The data from the world is mapped through a model to a model output distributions which should match a specified target distribution.

Recoding Example: Density Estimation.

Another special case of the recoding unsupervised framework is density estimation, where the reference (target) distribution is easy to calculate like an uniform distribution or a Gaussian distribution. A well known example is the mixtures of Gaussians (MoG) model [Pearson, 1894, Hasselblad, 1966, Duda and Hart, 1973]. The observation transformation must be locally invertible that is the Jacobian determinant must exists for the observation. That allows to compute the density of the observation. Note that these constraints are weaker than assuming to know the inverse of the transformation. For example with neural networks the Jacobian can be computed [Flake and Pearlmutter, 2000] and, therefore, it is local invertible but the inverse model is unknown.

Other Recoding Approaches.

Projection Pursuit and PCA. Other examples within the recoding framework are projection methods such as *projection pursuit* (e.g., [Friedman and Tukey, 1974, Friedman and Stuetzle, 1981, Huber, 1985, Friedman, 1987, Jones, 1987, Jones and Sibson, 1987, Zhao and Atkeson, 1996, Intrator, 1993]), *principal component analysis (PCA)* (e.g. [Oja, 1982, 1989, Jolliffe, 1986, Jackson, 1991]. Projection pursuit aims at an output distribution which is as non-Gaussian as possible where the non-Gaussianity is measured by the entropy. Note, that for a given variance the Gaussian distribution maximizes the entropy. PCA's objective for an one-dimensional projection is maximal variance. The projection is constraint to a linear mapping, the coefficient vector has unit length, and is orthogonal to previous projections.

Clustering and Coincidence Detection. An objective function can be given for *Clustering* methods and *Coincidence detection* approaches. Clustering methods can be interpreted as mixture models and describe the data by a multimodal distribution. Coincidence detection is based on the fact that the multidimensional distribution of observations has high density regions indicating correlated components in the observations. Therefore coincidence detection is closely related to independent component analysis.

Self-organizing maps (SOMs). For *self-organizing maps (SOMs)* [Kohonen, 1982, 1988, 1990, 1995, Ritter et al., 1992, 1991, Obermayer et al., 1992, Erwin et al., 1992] the objective function [**cannot** always be expressed as a single scalar function (like an energy or error function). Scalar objectives are important to derive learning algorithms based on optimizing this function and to compare models. The objective of SOMs is a scalar function for discrete input spaces and for discrete neighborhood functions otherwise the objective function must be expressed as a vector valued potential function [Kohonen, 1995, Cottrell et al., 1995, Ritter et al., 1992, 1991, Erwin et al., 1992]. The lack of a scalar objective function one of the major drawbacks of SOMs because models cannot be compared, overfitting not detected, and stopping of training is difficult to determine.

10.1.2.3 Recoding and Generative Framework Unified

If the recoding model has an unique inverse then the generative framework can be applied in the recoding context. The inverse model can use the observations as training examples which must be produced from some target distribution. Then the model maps the training examples to target distributions.

If the inverse model is obtained from the generative approach, then the recoding model is also available.

The objective function of principal component analysis (PCA) and independent component analysis (ICA) [Hinton and Sejnowski, 1999, Attias, 1999] attempt at keeping maximal information about the observations in the code while fulfilling certain constraints. In the generative view they are treated as generative models, which try to produce the data, whereas the constraints are coded into the model e.g. by using specific target distributions.

However, most recoding methods have higher input dimensionality than output dimensionality because the goal is to represent the input compactly and non-redundantly for visualization or for features extraction.

However, it is sometimes possible to formulate the recoding approach as a generative model even for a non-bijective model. The target distribution has to produce the observations by first generating a code which in turn generates the observations. Computing the likelihood required the computations of the probabilities of the codes corresponding to the observations. Density estimation, Projection pursuit and vector quantization can be treated is such a way [Hinton and Sejnowski, 1999, Attias, 1999]. The probabilities of the codes are the posterior of the code given the data, where we assume that an observation can be produced by different codes. For example, recoding methods which do not have bijective functions are *principal curves* [Mulier and Cherkassky, 1995, Ritter et al., 1992], which are a nonlinear generalization of principal components [Hastie and Stuetzle, 1989].

Example: Independent Component Analysis. An example for a recoding model treated as generative model and using objective functions is independent component analysis (ICA) [Schmidhuber, 1992b, Bell and Sejnowski, 1995, Deco and Brauer, 1995, Pearlmutter and Parra, 1997, Cardoso and Laheld, 1996, Cichocki et al., 1994, Jutten and Herault, 1991, Comon, 1994, Yang and Amari, 1997, Yang et al., 1996, Cardoso and Souloumiac, 1993, Hyvärinen, 1999], a method that discovers a representation of vector-valued observations in which the statistical dependence

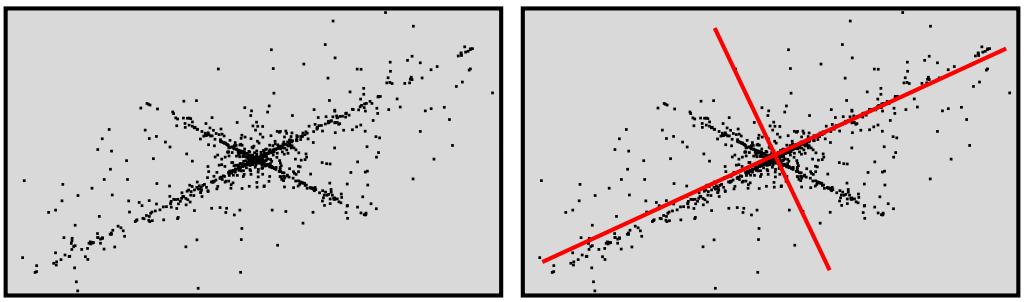


Figure 10.6: Principal component analysis for a two-dimensional data set. Left: the original data set. Right: The first principal component is shown as the line from lower left to upper right. The second principal component is orthogonal to the first component.

among the vector elements in the output space is minimized. With ICA, the model de-mixes observation vectors and the output should consist of statistically independent components. Towards this end the output distribution can be compared against a given factorial distribution. The generative approach assumes that the observations are produced by a model with independent hidden units [Cardoso, 1997, Moulines et al., 1997, Yang and Amari, 1997, Pearlmutter and Parra, 1997]. Alternatively, an objective or contrast function indicating statistically independent model output components can be used (see e.g. [Hyvärinen, 1999]).

Example: Density Estimation. The traditional density estimation framework supplies the inverse model and, therefore, can be viewed as a generative framework. For example in the mixture of Gaussians model each mixture component can be viewed as being generated by a Gaussian distribution with identity matrix as covariance matrix and thereafter transformed by a non-singular linear mapping. The posterior can be easily computed thus the model can serve data generation model.

10.2 Principal Component Analysis

Principal Component Analysis (PCA) also known as *Karhunen-Loève transform* (KTL) or as Hotelling transform makes a transformation of the coordinate system so that the first coordinate has the largest variance of the data, the second largest data variance is on the second coordinate, etc. The coordinates are called *principal components*. Fig. 10.6 shows the principal components of a two-dimensional data set and Fig. 10.7 shows how the first principal component is extracted from some data points.

Let $\boldsymbol{X} = \left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\right)^T \in \mathbb{R}^{l \times d}$ be the matrix of data, then the data can be centered at the origin by

$$\bar{\boldsymbol{X}} = \boldsymbol{X} - \frac{1}{l}\,\mathbf{1}\,\mathbf{1}^T\boldsymbol{X}\,. \tag{10.1}$$

Principal component analysis can be performed by computing the eigenvectors and eigenvalues of the covariance matrix

$$\boldsymbol{C} = \frac{1}{l}\,\bar{\boldsymbol{X}}^T\bar{\boldsymbol{X}} = \boldsymbol{W}\boldsymbol{\Lambda}^2\boldsymbol{W}^T\,, \tag{10.2}$$

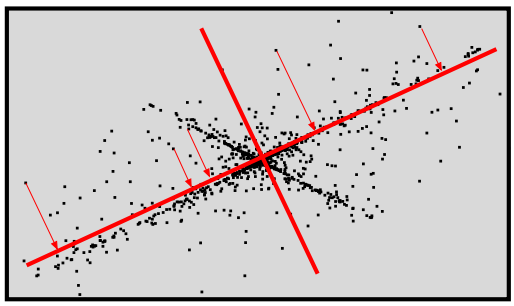


Figure 10.7: Principal component analysis for a two-dimensional data set. The first principal component is extracted for some data points by projecting them onto the first component and measuring the distance to the origin.

where $\mathbf{\Lambda}$ is the diagonal matrix of the roots of the eigenvalues and $\boldsymbol{W}$ is the matrix of principal components (the eigenvectors). The roots of the eigenvalues tell how much data variance is explained by the corresponding eigenvector.

The projection

$$\boldsymbol{x}^T \boldsymbol{w}_i \tag{10.3}$$

of a data vector $\boldsymbol{x}$ onto the i-th eigenvector $\boldsymbol{w}_i$ is the i-th principal component of the data vector $\boldsymbol{x}$.

If we normalize the data so that

$$\sum_{i=1}^{d} \lambda_i \;=\; 1 \tag{10.4}$$

and if the first k principal components are used to represent the data, then

$$\mathrm{E(MSE)} \;=\; \sum_{i=k+1}^{d} \lambda_i \tag{10.5}$$

is the expected mean squared reconstruction error between the back-projected representation and the original data.

Especially if the eigenvalues for the first principal components are large then most information in the data can be represented by these first eigenvalues. This is important for down-projecting the data for visualization.

The first principal component is described as

$$\boldsymbol{w}_1 \;=\; \arg\max_{\|\boldsymbol{w}\|=1} \mathrm{E}\left(\left(\boldsymbol{w}^T \boldsymbol{x}\right)^2\right) \tag{10.6}$$

and the k-th as

$$\boldsymbol{w}_k \;=\; \arg\max_{\|\boldsymbol{w}\|=1} \mathrm{E}\left(\left(\boldsymbol{w}^T \left(\boldsymbol{x} \;-\; \sum_{i=1}^{k-1} \boldsymbol{w}_i \, \boldsymbol{w}_i^T \boldsymbol{x}\right)\right)^2\right) \;. \tag{10.7}$$

Iterative methods for PCA are sometimes to prefer if the dimension d is large or if on-line methods should be implemented.

Most famous is Oja's rule [Oja, 1982]. First let us the projection define as

$$y \;=\; \boldsymbol{w}^T \boldsymbol{x}\;. \tag{10.8}$$

Oja's rule is

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w} \;+\; \eta\;\left(y\,\boldsymbol{x} \;-\; y^2\,\boldsymbol{w}\right)\;, \tag{10.9}$$

where η is the learning rate.

The eigenvectors of $\boldsymbol{C}$ are the fixed points of Oja's rule and only the eigenvector with largest eigenvalue is a stable fixed point.

10.3 Independent Component Analysis

Independent Component Analysis (ICA) attempts at finding a code for the observation where the components are mutually statistical independent.

The assumption for ICA is that the observations $\boldsymbol{x}$ are generated by mixing the sources $\boldsymbol{s}$, where both $\boldsymbol{x}$ and $\boldsymbol{s}$ are d-dimensional vectors:

$$\boldsymbol{x} \;=\; \boldsymbol{A}\,\boldsymbol{s}\;. \tag{10.10}$$

Goal is to find a matrix $\boldsymbol{W}$ with

$$\boldsymbol{s} \;=\; \boldsymbol{W}\,\boldsymbol{x}\;. \tag{10.11}$$

The main assumption says that the sources are statistically independent:

$$p(\boldsymbol{s}) \;=\; \prod_{j=1}^{d} p(s_j)\;. \tag{10.12}$$

Fig. 10.8 shows how two speaker (the sources $\boldsymbol{s}$) speak independently from each other. The microphones record the acoustic signals and are the observations $\boldsymbol{x}$.

In the optimal case we obtain $\boldsymbol{W} \;=\; \boldsymbol{A}^{-1}$. However

$$\boldsymbol{x} \;=\; \boldsymbol{A}\,\boldsymbol{P}\,\boldsymbol{P}^{-1}\,\boldsymbol{s} \tag{10.13}$$

holds true for all permutation matrices $\boldsymbol{P}$ and all diagonal (scaling) matrices $\boldsymbol{P}$. In this case $\bar{\boldsymbol{s}} \;=\; \boldsymbol{P}^{-1}\,\boldsymbol{s}$ are sources which are statistically independent and can be mixed to give $\boldsymbol{x}$ with mixing matrix $\boldsymbol{A}\,\boldsymbol{P}$.

This means the problem formulation gives only unique solutions up to permutations and scaling.

Fig. 10.9 shows the ICA solution of the data set of Fig. 10.6 and Fig. 10.10 compares the PCA and the ICA solution.

Assumptions which must be made are

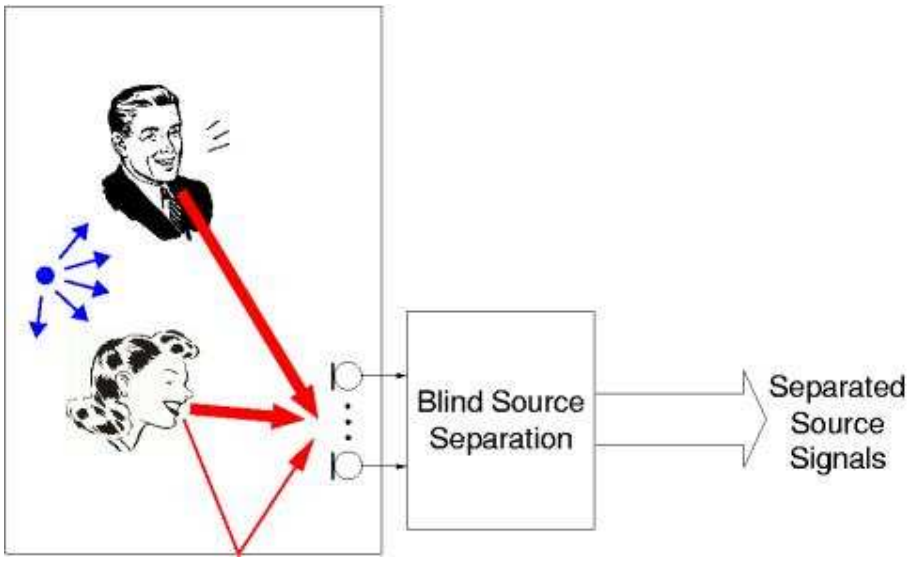


Figure 10.8: Two speakers recorded by two microphones. The speakers produce independent acoustic signals which can be separated by ICA.

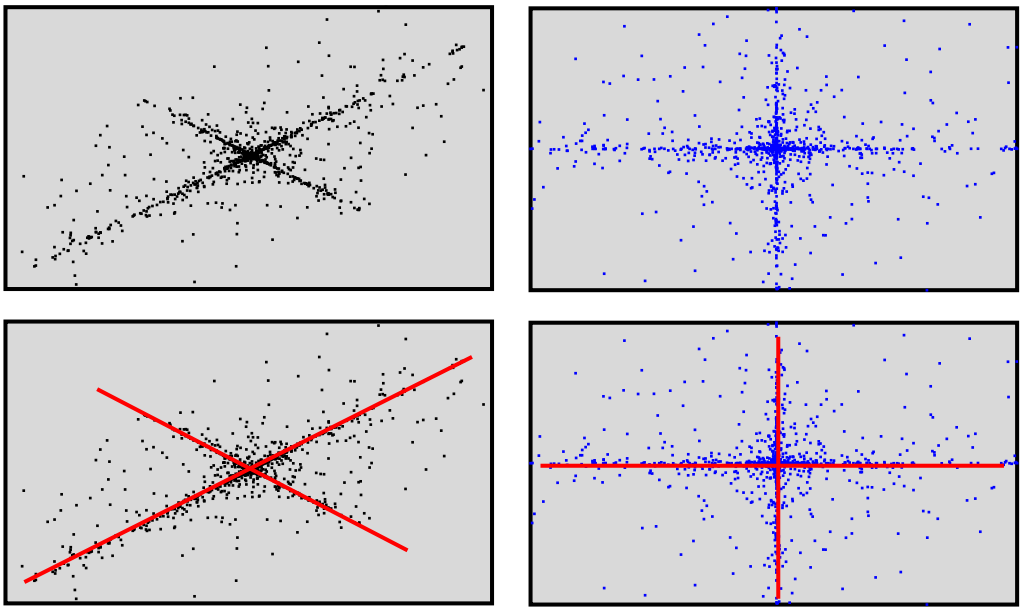


Figure 10.9: Independent component analysis on the data set of Fig. 10.6. Left subfigures show the original data and the right subfigures the transformed data by ICA. The bottom line is the top line but the ICA components are shown.

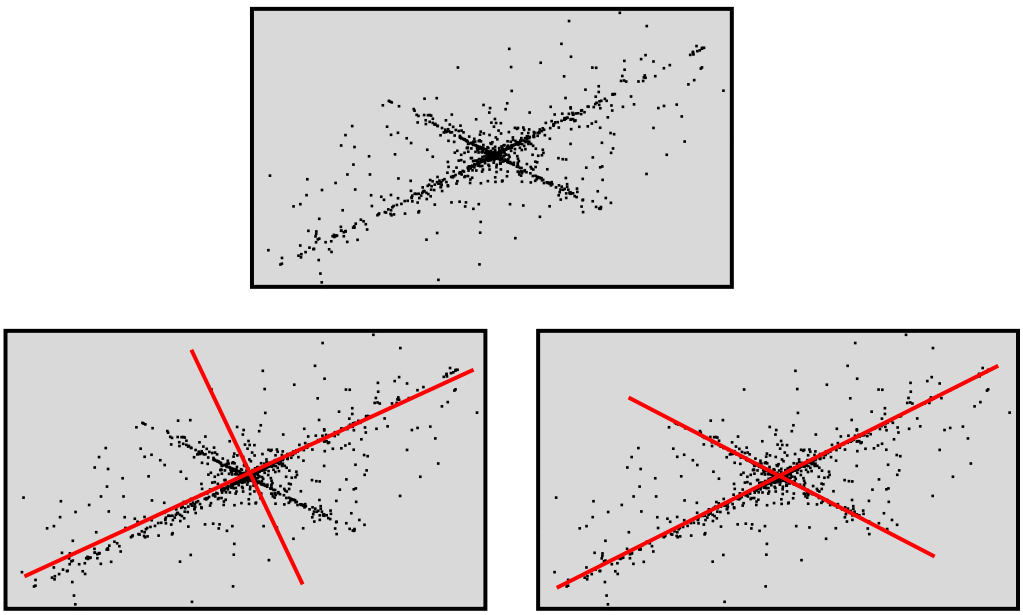


Figure 10.10: Comparison of PCA and ICA on the data set of Fig. 10.6. Top: original data set. Bottom left: PCA solution. Bottom right: ICA solution.

- the source components s_i are non-Gaussian (except for one component), because mixtures of Gaussians are Gaussians and no unique solution exists.
- the observation dimension must be at least as large as the number of sources
- the matrix $\boldsymbol{A}$ must have full rank.

We will combine the last two assumptions by assume that the observations $\boldsymbol{x}$ and the sources $\boldsymbol{s}$ live in a d-dimensional space and $\boldsymbol{A}^{-1} \in \mathbb{R}^{d \times d}$ exists.

If the densities $p(s_i)$ are known or approximated – e.g. by super-Gaussians –, then the maximum likelihood method can be applied. In this case the generative model is known and we can first estimate $\boldsymbol{A}$ and then compute $\boldsymbol{W} = \boldsymbol{A}^{-1}$.

10.3.1 Measuring Independence

The entropy of a factorial code is larger than the entropy of the joint distribution. The difference of the two expressions is the mutual information between the variables s_j:

$$I(s_1, \ldots, s_d) = \sum_{j=1}^{d} H(s_j) - H(\boldsymbol{s}) , \tag{10.14}$$

where H denotes the entropy

$$H(\boldsymbol{a}) = - \int p(\boldsymbol{a}) \ln p(\boldsymbol{a}) \, d\boldsymbol{a} . \tag{10.15}$$

If we set

$$\boldsymbol{s} \;=\; \boldsymbol{W}\,\boldsymbol{x} \tag{10.16}$$

then

$$I(s_1,\ldots,s_d) \;=\; \sum_{j=1}^{d} H(s_j) \;-\; H(\boldsymbol{x}) \;-\; \ln|\boldsymbol{W}| \;, \tag{10.17}$$

where $|\boldsymbol{W}|$ is the absolute value of the determinant of the matrix $\boldsymbol{W}$.

Above equation stems from the fact that

$$p(\boldsymbol{s}) \;=\; \frac{p(\boldsymbol{x})}{|\boldsymbol{W}|}\,. \tag{10.18}$$

The negentropy is defined as

$$J(\boldsymbol{y}) \;=\; H(\boldsymbol{y}_{\mathrm{gauss}}) \;-\; H(\boldsymbol{y})\,, \tag{10.19}$$

where $\boldsymbol{y}_{\mathrm{gauss}}$ is a Gaussian random vector with the same covariance matrix as $\boldsymbol{y}$. The negentropy is an affinely invariant version of the entropy.

The maximal negentropy is equivalent to representations where the mutual information between the components is minimized. Here the connection between ICA and projection pursuit is clear because both can be expressed in maximize the distance to Gaussian distributions.

Unfortunately, the negentropy cannot be used easily because its estimation is difficult.

The non-Gaussianity can be measured through other parameters for example through the cummulants. For example the fourth cummulant, *kurtosis* is a common measure for non-Gaussianity where positive kurtosis indicates super-Gaussians (the tails are smaller than for Gaussians) and negative kurtosis indicates sub-Gaussians (the tails are larger than for Gaussians).

For zero mean variables the cummulants are defined as

$$\kappa_1 \;=\; \mathrm{E}(x) \;=\; 0 \tag{10.20}$$
$$\kappa_2 \;=\; \mathrm{E}(x^2) \tag{10.21}$$
$$\kappa_3 \;=\; \mathrm{E}(x^3) \tag{10.22}$$
$$\kappa_4 \;=\; \mathrm{E}(x^4) \;-\; 3\left(\mathrm{E}(x^2)\right)^2\,. \tag{10.23}$$

κ_4 is called kurtosis. For Gaussians $\kappa_4 = 0$ and for x_1 and x_2 independent:

$$\kappa_4(x_1 \;+\; x_2) \;=\; \kappa_4(x_1) \;+\; \kappa_4(x_2) \tag{10.24}$$
$$\kappa_4(\alpha\,x) \;=\; \alpha^4\,\kappa_4(x)\,. \tag{10.25}$$

Many ICA algorithms use *contrast functions* which measure the independence of the variables and are used as objective functions.

Common contrast functions are

- $\kappa_4(y)$, the kurtosis
- $\frac{1}{12}\,\kappa_3^2(y) \;+\; \frac{1}{48}\,\kappa_4^2(y)$, where the variable y is normalized to zero mean and unit variance
- $|\mathrm{E}_y(G(y)) \;-\; E_\nu(G(\nu)))|^p$, where ν is a standardized Gaussian, $p = 1, 2$, and y is normalized to zero mean and unit variance. Here G can be the kurtosis for which $G(\nu) = 0$ would hold. Other choices for G are $G(x) \;=\; \log\cosh a\; x$ and $G(x) \;=\; \exp(-a\; x^2/2)$ with $a \geq 1$.

10.3.2 INFOMAX Algorithm

The INFOMAX algorithm is motivated by maximizing the entropy $H(\boldsymbol{g}(\boldsymbol{y}))$, where $\boldsymbol{g}$ is an activation function of a neural network with output

$$g(y_i) \tag{10.26}$$

and

$$\boldsymbol{y} \;=\; \boldsymbol{W}\;\boldsymbol{x}\;. \tag{10.27}$$

If the entropy is maximized then

$$I(g(y_1), \ldots, g(y_d)) \;=\; \sum_{j=1}^{d} H(g(y_j)) \;-\; H(\boldsymbol{g}(\boldsymbol{y})) \;=\; 0 \tag{10.28}$$

and the components $(g(y_1), \ldots, g(y_d))$ are statistically independent.

Very common choice for g_i is

$$g(y_i) \;=\; \tanh(y_i)\;. \tag{10.29}$$

We have

$$p(\boldsymbol{g}(\boldsymbol{y})) \;=\; p(\boldsymbol{x}) \left| \frac{\partial \boldsymbol{g}(\boldsymbol{y})}{\partial \boldsymbol{y}}\; \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right|^{-1} \;=\; p(\boldsymbol{x}) \left| \frac{\partial \boldsymbol{g}(\boldsymbol{y})}{\partial \boldsymbol{y}} \boldsymbol{W} \right|^{-1} \tag{10.30}$$

where

$$\left| \frac{\partial \boldsymbol{g}(\boldsymbol{y})}{\partial \boldsymbol{y}} \boldsymbol{W} \right| \;=\; \left| \prod_{j=1}^{d} g'(y_j) \right| \; |\boldsymbol{W}|\;. \tag{10.31}$$

The entropy is then

$$\begin{aligned} & H(\boldsymbol{g}(\boldsymbol{y})) \;=\; \mathrm{E}\,(-\;\ln \boldsymbol{g}(\boldsymbol{y})) \;= \\ & H(\boldsymbol{x}) \;+\; \mathrm{E}\left(\sum_{j=1}^{d} \left| \ln g'(y_j) \right| \right) \;+\; \ln|\boldsymbol{W}| \;\approx \\ & H(\boldsymbol{x}) \;+\; \frac{1}{l} \sum_{i=1}^{l} \sum_{j=1}^{d} \left| \ln g'(y_j^i) \right| \;+\; \ln|\boldsymbol{W}|\;, \end{aligned} \tag{10.32}$$

where

$$\boldsymbol{y}^i \;=\; \boldsymbol{W}\,\boldsymbol{x}^i\;. \tag{10.33}$$

Now we can maximize the entropy. Very common choice for g_i is

$$g(y_i) \;=\; \tanh(y_i) \tag{10.34}$$

which gives

$$\frac{\partial}{\partial \boldsymbol{w}_j} \ln g'(y_j) \;=\; \frac{g''(y_j)}{g'(y_j)}\,\boldsymbol{x}^T \;=\; -\,2\,g(y_j)\,\boldsymbol{x}^T\;. \tag{10.35}$$

For sigmoid activation function we have

$$\frac{\partial}{\partial \boldsymbol{w}_j} \ln g'(y_j) \;=\; (1\;-\;2\,g(y_j))\;\boldsymbol{x}^T\;. \tag{10.36}$$

Further we have

$$\frac{\partial}{\partial \boldsymbol{W}} \ln |\boldsymbol{W}| \;=\; \left(\boldsymbol{W}^T\right)^{-1}\;. \tag{10.37}$$

Because $H(\boldsymbol{x})$ does not depend on $\boldsymbol{W}$ we obtain

$$\frac{\partial}{\partial \boldsymbol{W}} H(\boldsymbol{g}(\boldsymbol{y})) \;=\; \left(\boldsymbol{W}^T\right)^{-1} -\;2\,\boldsymbol{g}(\boldsymbol{y})\,\boldsymbol{x}^T \tag{10.38}$$

for $\tanh$ and for the sigmoid activation function

$$\frac{\partial}{\partial \boldsymbol{W}} H(\boldsymbol{g}(\boldsymbol{y})) \;=\; \left(\boldsymbol{W}^T\right)^{-1} +\;(\mathbf{1}\;-\;2\,\boldsymbol{g}(\boldsymbol{y}))\;\boldsymbol{x}^T \tag{10.39}$$

for the derivatives.

The update rules are for the $\tanh$ activation function

$$\Delta\boldsymbol{W} \;\propto\; \left(\boldsymbol{W}^T\right)^{-1} -\;2\,\boldsymbol{g}(\boldsymbol{y})\,\boldsymbol{x}^T \tag{10.40}$$

and for the sigmoid activation function

$$\Delta\boldsymbol{W} \;\propto\; \left(\boldsymbol{W}^T\right)^{-1} +\;(\mathbf{1}\;-\;2\,\boldsymbol{g}(\boldsymbol{y}))\;\boldsymbol{x}^T\;. \tag{10.41}$$

For this update rule the natural gradient can be applied which takes the geometrical structure of the parameter space into account. In this case the update rule is multiplied with $\boldsymbol{W}^T\boldsymbol{W}$. The update rule are now for the $\tanh$ activation function

$$\Delta\boldsymbol{W} \;\propto\; \left(\boldsymbol{I}\;-\;2\,\boldsymbol{g}(\boldsymbol{y})\,\boldsymbol{x}^T\right)\;\boldsymbol{W} \tag{10.42}$$

and for the sigmoid activation function

$$\Delta\boldsymbol{W} \;\propto\; \left(\boldsymbol{I}\;+\;(\mathbf{1}\;-\;2\,\boldsymbol{g}(\boldsymbol{y}))\;\boldsymbol{x}^T\right)\;\boldsymbol{W}\;. \tag{10.43}$$

INFOMAX is equivalent to a maximum likelihood approach, when $g_i'(s_i) \;=\; p(s_i)$.

10.3.3 EASI Algorithm

The EASI algorithm uses the following update rule:

$$\Delta \boldsymbol{W} \;\propto\; \left(\boldsymbol{I} \;-\; \boldsymbol{y}\,\boldsymbol{y}^T \;-\; \boldsymbol{g}(\boldsymbol{y})\boldsymbol{y}^T \;+\; \boldsymbol{y}\,\boldsymbol{g}^T(\boldsymbol{y})\right)\; \boldsymbol{W}\,. \tag{10.44}$$

Here the nonlinear functions g are similar to the functions used by INFOMAX.

10.3.4 FastICA Algorithm

The FastICA algorithm is an fixed point algorithm which was originally based on the kurtosis and extracts a weight vector $\boldsymbol{w}$.

The iteration step of the FastICA algorithm is

$$\boldsymbol{w}^{\text{new}} \;=\; \mathrm{E}\left(\boldsymbol{x}\; g(\boldsymbol{w}^T\,\boldsymbol{x})\right) \;-\; \mathrm{E}\left(g'(\boldsymbol{w}^T\,\boldsymbol{x})\right)\; \boldsymbol{w}\,, \tag{10.45}$$

where g is the derivative of the contrast function, which has been defined above.

Instead of the expectation E the empirical mean over the training examples is used.

10.4 Factor Analysis

Here we focus on the maximum likelihood factor analysis. In contrast to PCA, factor analysis has the advantage that it is a generative approach and it does model the independent noise at the observations. Disadvantage of factor analysis is that it has to make assumptions on the distribution like that the factors are Gaussian and the noise is Gaussian.

We are given the data $\{\boldsymbol{x}\} \;=\; \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^l\}$ which is already normalized to mean zero (by subtracting the mean $\boldsymbol{\mu}$ from the data). The model is

$$\boldsymbol{x} \;\;=\;\; \boldsymbol{\Lambda}\boldsymbol{z} \;+\; \boldsymbol{\epsilon}\,, \tag{10.46}$$

where

$$\boldsymbol{z} \;\sim\; \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right) \quad \text{and} \quad \boldsymbol{\epsilon} \;\sim\; \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Psi}\right)\,. \tag{10.47}$$

The observations $\boldsymbol{x}, \boldsymbol{\epsilon} \in \mathbb{R}^d$, the *factors* $\boldsymbol{z} \in \mathbb{R}^p$, the *factor loading matrix* $\boldsymbol{\Lambda} \in \mathbb{R}^{d \times p}$, and the noise covariance matrix $\boldsymbol{\Psi}$ is a diagonal matrix from $\mathbb{R}^{d \times d}$.

In general the model has fewer factors than observations: $d \geq p$. The diagonal form of $\boldsymbol{\Psi}$ is reasonable if the measurements are taken independently and the noise at the components are mutually independent. Therefore, the observations are mutually independent if the factors are known (only the noise is the random component).

Correlations between observations can only be explained by factors.

We assume that $\boldsymbol{\epsilon}$ and $\boldsymbol{z}$ are independent which must not be true for all systems, e.g. if the noise changes with the signal strength.

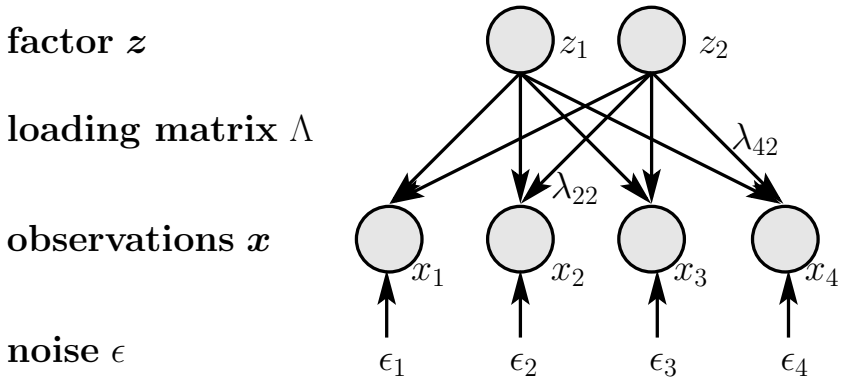


Figure 10.11: The factor analysis model.

The free parameters of the model are $\mathbf{\Lambda}$ and $\mathbf{\Psi}$, which can be estimated by maximum likelihood. Both parameter sets explain the variance in the observations $\boldsymbol{x}$, however $\mathbf{\Lambda}$ explains the dependent part whereas $\mathbf{\Psi}$ explains the independent part of the variance. The factors are comparable with the principal components of the PCA method and data components can be projected onto them.

Fig. 10.11 depicts the factor analysis model.

We have

$$\boldsymbol{x} \mid \boldsymbol{z} \;\sim\; \mathcal{N}\left(\mathbf{\Lambda}\boldsymbol{z}, \mathbf{\Psi}\right) \,, \tag{10.48}$$

because if $\boldsymbol{z}$ is given, then only the noise distribution is present.

We focus on the maximum likelihood approach to factor analysis is in most cases [Jöreskog, 1967] based on the Expectation-Maximization (EM) optimization technique [Dempster et al., 1977].

We will now consider the likelihood of the data. Let denote E the expectation of the data (i.e. the factor distribution and the noise distribution is combined), then we obtan for the first two moments:

$$\begin{aligned} &\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\mathbf{\Lambda}\boldsymbol{z} \;+\; \boldsymbol{\epsilon}) \;=\; \mathbf{\Lambda}\mathrm{E}(\boldsymbol{z}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0} \,, \\ &\mathrm{E}\left(\boldsymbol{x}\,\boldsymbol{x}^T\right) \;=\; \mathrm{E}\left((\mathbf{\Lambda}\boldsymbol{z} \;+\; \boldsymbol{\epsilon})(\mathbf{\Lambda}\boldsymbol{z} \;+\; \boldsymbol{\epsilon})^T\right) \;= \\ &\mathbf{\Lambda}\mathrm{E}\left(\boldsymbol{z}\,\boldsymbol{z}^T\right)\mathbf{\Lambda}^T \;+\; \mathbf{\Lambda}\mathrm{E}\left(\boldsymbol{z}\right)\mathrm{E}\left(\boldsymbol{\epsilon}^T\right) \;+\; \mathrm{E}\left(\boldsymbol{z}\right)\mathrm{E}\left(\boldsymbol{\epsilon}\right)\,\mathbf{\Lambda}^T \;+\; \mathrm{E}\left(\boldsymbol{\epsilon}\,\boldsymbol{\epsilon}^T\right) \;= \\ &\mathbf{\Lambda}\,\mathbf{\Lambda}^T \;+\; \mathbf{\Psi} \,. \end{aligned} \tag{10.49}$$

The variance can be computed as

$$\begin{aligned} &\mathrm{var}(\boldsymbol{x}) \;=\; \mathrm{E}\left(\boldsymbol{x}\,\boldsymbol{x}^T\right) \;-\; \left(\mathrm{E}(\boldsymbol{x})\right)^2 \;= \\ &\mathbf{\Lambda}\,\mathbf{\Lambda}^T \;+\; \mathbf{\Psi} \,. \end{aligned} \tag{10.50}$$

Therefore, the marginal distribution for $\boldsymbol{x}$ is

$$\boldsymbol{x} \;\sim\; \mathcal{N}\left(\mathbf{0}\,,\; \mathbf{\Lambda}\mathbf{\Lambda}^T \;+\; \mathbf{\Psi}\right) \,. \tag{10.51}$$

This means that the observations are Gaussian distributed. This is an assumption of the factor analysis model which can be checked to see whether the model is applicable to a certain problem.

The log-likelihood of the data $\{\boldsymbol{x}\}$ under the model $(\boldsymbol{\Lambda}, \boldsymbol{\Psi})$ is

$$\begin{aligned} &\log \prod_{i=1}^{l} (2\pi)^{-d/2} \left|\boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right|^{-1/2} \\ &\exp\left(-\frac{1}{2}\left((\boldsymbol{x}^i)^T \left(\boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right)^{-1} \boldsymbol{x}^i\right)\right) \, , \end{aligned} \tag{10.52}$$

where $|.|$ denotes the absolute value of the determinant of a matrix.

To maximize the likelihood is difficult because no closed form of directly maximizing the likelihood with respect to the parameters is known.

We again apply the EM-algorithm. We introduce a distribution which estimates the hidden states, here the factors.

Using

$$Q_i(\boldsymbol{z}^i) \, = \, p\left(\boldsymbol{z}^i \mid \boldsymbol{x}^i; \boldsymbol{\Lambda}, \boldsymbol{\Psi}\right) \tag{10.53}$$

then

$$\begin{aligned} \boldsymbol{z}^i \mid \boldsymbol{x}^i \; &\sim \; \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}, \boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\right) \\ \boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \; &= \; \left(\boldsymbol{x}^i\right)^T \left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right)^{-1} \boldsymbol{\Lambda} \\ \boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \; &= \; \boldsymbol{I} \, - \, \boldsymbol{\Lambda}^T \left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right)^{-1} \boldsymbol{\Lambda} \, + \\ &\qquad \left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right)^{-1} \boldsymbol{x}^i \, (\boldsymbol{x}^i)^T \left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T \, + \, \boldsymbol{\Psi}\right)^{-1} \, , \end{aligned} \tag{10.54}$$

where we used the fact that

$$\begin{aligned} &\boldsymbol{v} \, \sim \, \mathcal{N}\left(\boldsymbol{\mu}_v, \Sigma_{vv}\right) \, , \, \boldsymbol{u} \, \sim \, \mathcal{N}\left(\boldsymbol{\mu}_u, \Sigma_{uu}\right) \, , \\ &\Sigma_{uv} \, = \, \text{Covar}(\boldsymbol{u}, \boldsymbol{v}) \text{ and } \; \Sigma_{vu} \, = \, \text{Covar}(\boldsymbol{v}, \boldsymbol{u}) : \\ &\boldsymbol{v} \mid \boldsymbol{u} \, \sim \, \mathcal{N}\left(\boldsymbol{\mu}_v \, + \, \Sigma_{vu}\Sigma_{uu}^{-1}\left(\boldsymbol{u} \, - \, \boldsymbol{\mu}_u\right) \, , \, \Sigma_{vv} \, + \, \Sigma_{vu}\Sigma_{uu}^{-1}\Sigma_{uv}\right) \end{aligned} \tag{10.55}$$

and

$$\mathrm{E}(\boldsymbol{z}\boldsymbol{x}) \, = \, \boldsymbol{\Lambda} \, \mathrm{E}(\boldsymbol{z} \, \boldsymbol{z}^T) \, = \, \boldsymbol{\Lambda} \, . \tag{10.56}$$

We obtain

$$\begin{aligned} &Q_i(\boldsymbol{z}^i) \, = \, (2\pi)^{-d/2} \left|\boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\right|^{-1/2} \\ &\exp\left(-\frac{1}{2} \, \left(\boldsymbol{z}^i \, - \, \boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\right)^T \, \boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i}^{-1} \left(\boldsymbol{z}^i \, - \, \boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\right)\right) \, . \end{aligned} \tag{10.57}$$

The EM algorithm for maximum likelihood maximizes in the M-step a lower bound for the likelihood:

$$\begin{aligned} &\log\left(p(\boldsymbol{x}^i \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})\right) \; = \\ &\log\left(\int_{\mathbb{R}^p} \frac{Q_i(\boldsymbol{z}^i)\; p(\boldsymbol{x}^i, \boldsymbol{z}^i \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})}{Q_i(\boldsymbol{z}^i)} d\boldsymbol{z}^i\right) \; \geq \\ &\int_{\mathbb{R}^p} Q_i(\boldsymbol{z}^i)\; \log\left(\frac{p(\boldsymbol{x}^i, \boldsymbol{z}^i \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})}{Q_i(\boldsymbol{z}^i)}\right) d\boldsymbol{z}^i \; . \end{aligned} \tag{10.58}$$

Using the expectation

$$\mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(f(\boldsymbol{z}^i)\right) \; = \; \int_{\mathbb{R}^p} Q_i(\boldsymbol{z}^i)\; f(\boldsymbol{z}^i)\; d\boldsymbol{z}^i \tag{10.59}$$

and neglecting all terms which are independent of $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$, the M-step requires to maximize

$$\begin{aligned} &\log \mathcal{L} \; = \; \frac{d\, l}{2} \log\left(2\pi\right) \; - \frac{l}{2} \log\left|\boldsymbol{\Psi}\right| \; - \\ &\frac{1}{2} \sum_{i=1}^{l} \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}\boldsymbol{z}^i\right)^T \boldsymbol{\Psi}^{-1} \left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}\boldsymbol{z}^i\right)\right) \; . \end{aligned} \tag{10.60}$$

The optimality criteria are

$$\begin{aligned} &\frac{1}{l} \nabla_{\boldsymbol{\Lambda}} \log \mathcal{L} \; = \; \frac{1}{l} \sum_{i=1}^{l} \boldsymbol{\Psi}^{-1}\; \boldsymbol{\Lambda}\; \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\boldsymbol{z}^i\; (\boldsymbol{z}^i)^T\right) \; - \\ &\frac{1}{l} \sum_{i=1}^{l} \boldsymbol{\Psi}^{-1}\; \boldsymbol{x}^i\; \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\boldsymbol{z}^i\right) \; = \; \mathbf{0} \end{aligned} \tag{10.61}$$

and

$$\begin{aligned} &\nabla_{\boldsymbol{\Psi}} \log \mathcal{L} \; = \; -\frac{l}{2} \boldsymbol{\Psi}^{-1} \; + \\ &\frac{1}{2} \sum_{i=1}^{l} \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\boldsymbol{\Psi}^{-1} \left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}\boldsymbol{z}^i\right)\; \left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}\boldsymbol{z}^i\right)^T \boldsymbol{\Psi}^{-1}\right) \; = \; \mathbf{0} \; . \end{aligned} \tag{10.62}$$

Solving above equations gives:

$$\boldsymbol{\Lambda}^{\mathrm{new}} \; = \; \left(\frac{1}{l} \sum_{i=1}^{l} \boldsymbol{x}^i\; \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\boldsymbol{z}^i\right)\right) \; \left(\frac{1}{l} \sum_{i=1}^{l} \mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i} \left(\boldsymbol{z}^i\; (\boldsymbol{z}^i)^T\right)\right)^{-1} \tag{10.63}$$

and

$$
\begin{aligned}
&\boldsymbol{\Psi}^{\text{new}} \; = \qquad\qquad (10.64)\\
&\text{diag}\left(\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}^{\text{new}}\boldsymbol{z}^i\right)\left(\boldsymbol{x}^i \; - \; \boldsymbol{\Lambda}^{\text{new}}\boldsymbol{z}^i\right)^T\right)\right) \; = \\
&\text{diag}\left(\frac{1}{l}\sum_{i=1}^{l} \boldsymbol{x}^i\,(\boldsymbol{x}^i)^T \; - \; \frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\boldsymbol{x}^i\,(\boldsymbol{\Lambda}^{\text{new}})^T \; - \right.\\
&\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\boldsymbol{\Lambda}^{\text{new}}(\boldsymbol{x}^i)^T \; + \\
&\left.\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\,(\boldsymbol{z}^i)^T\right)\boldsymbol{\Lambda}^{\text{new}}\,(\boldsymbol{\Lambda}^{\text{new}})^T\right) \; ,
\end{aligned}
$$

where "diag" makes a diagonal matrix from a matrix by setting all non-diagonal elements to zero.

From eq. (10.63) we obtain

$$
\boldsymbol{\Lambda}^{\text{new}}\left(\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\,(\boldsymbol{z}^i)^T\right)\right) \; = \; \left(\frac{1}{l}\sum_{i=1}^{l} \boldsymbol{x}^i\,\text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\right) \; . \qquad (10.65)
$$

and can replace the left hand side of above equation in the last term of eq. (10.64) which leads to the fact that one term $\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\boldsymbol{\Lambda}^{\text{new}}(\boldsymbol{x}^i)^T$ cancels in eq. (10.64). We obtain

$$
\boldsymbol{\Psi}^{\text{new}} \; = \; \frac{1}{l}\text{diag}\left(\sum_{i=1}^{l} \boldsymbol{x}^i\,(\boldsymbol{x}^i)^T \; - \; \sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\boldsymbol{x}^i\,(\boldsymbol{\Lambda}^{\text{new}})^T\right) \; . \qquad (10.66)
$$

This leads to following EM updates:

E-step: (10.67)

$$
\begin{aligned}
&\text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right) \; = \; \boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\\
&\text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\,(\boldsymbol{z}^i)^T\right) \; = \; \boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\,\boldsymbol{\mu}^T_{\boldsymbol{z}^i|\boldsymbol{x}^i} \; + \; \boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i}
\end{aligned}
$$

M-step: (10.68)

$$
\begin{aligned}
&\boldsymbol{\Lambda}^{\text{new}} \; =\\
&\left(\frac{1}{l}\sum_{i=1}^{l} \boldsymbol{x}^i\,\text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\right)\left(\frac{1}{l}\sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\,(\boldsymbol{z}^i)^T\right)\right)^{-1}
\end{aligned}
$$

$$
\begin{aligned}
&\boldsymbol{\Psi}^{\text{new}} \; = \qquad\qquad (10.69)\\
&\frac{1}{l}\text{diag}\left(\sum_{i=1}^{l} \boldsymbol{x}^i\,(\boldsymbol{x}^i)^T \; - \; \sum_{i=1}^{l} \text{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right)\boldsymbol{x}^i\,(\boldsymbol{\Lambda}^{\text{new}})^T\right) \; .
\end{aligned}
$$

Speed Ups. To speed up the algorithm for especially for $d > p$ the matrix inversion lemma can be used:

$$\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1} = \boldsymbol{\Psi}^{-1} - \boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\left(\boldsymbol{I} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right)^{-1}\boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\,, \tag{10.70}$$

where $\boldsymbol{\Psi}^{-1}$ can be evaluated very fast because it is a diagonal matrix.

Another speed up is obtained

$$\begin{aligned}
&\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{x}^i\,\mathrm{E}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\left(\boldsymbol{z}^i\right) = \\
&\left(\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T\right)\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Lambda} = \\
&\boldsymbol{C}\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Lambda} = \\
&\boldsymbol{C}\left(\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda} - \boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\left(\boldsymbol{I} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right)^{-1}\boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right) = \\
&\boldsymbol{C}\left(\boldsymbol{A} - \boldsymbol{A}\left(\boldsymbol{I} + \boldsymbol{B}\right)^{-1}\boldsymbol{B}\right)\,,
\end{aligned} \tag{10.71}$$

where $\boldsymbol{A} = \boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}$, $\boldsymbol{B} = \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T\boldsymbol{A}$, and $\boldsymbol{C}$ is the empirical covariance matrix, which has to be computed only once.

We can also compute

$$\begin{aligned}
&\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{\Sigma}_{\boldsymbol{z}^i|\boldsymbol{x}^i} = \\
&\boldsymbol{I} - \boldsymbol{\Lambda}^T\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Lambda} + \\
&\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1}\left(\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T\right)\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T + \boldsymbol{\Psi}\right)^{-1} = \\
&\boldsymbol{I} - \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\left(\boldsymbol{I} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right)^{-1}\boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda} + \\
&\left(\boldsymbol{\Psi}^{-1} - \boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\left(\boldsymbol{I} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right)^{-1}\boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\right)\boldsymbol{C} \\
&\left(\boldsymbol{\Psi}^{-1} - \boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\left(\boldsymbol{I} + \boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}\right)^{-1}\boldsymbol{\Lambda}^T\boldsymbol{\Psi}^{-1}\right) = \\
&\boldsymbol{I} - \boldsymbol{B} + \boldsymbol{B}\left(\boldsymbol{I} + \boldsymbol{B}\right)^{-1}\boldsymbol{B} + \\
&\left(\boldsymbol{\Psi}^{-1} - \boldsymbol{A}\left(\boldsymbol{I} + \boldsymbol{B}\right)^{-1}\boldsymbol{A}^T\right)\boldsymbol{C}\left(\boldsymbol{\Psi}^{-1} - \boldsymbol{A}\left(\boldsymbol{I} + \boldsymbol{B}\right)^{-1}\boldsymbol{A}^T\right)
\end{aligned} \tag{10.72}$$

and

$$\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}\,\boldsymbol{\mu}_{\boldsymbol{z}^i|\boldsymbol{x}^i}^T \;= \tag{10.73}$$
$$\boldsymbol{\Lambda}^T\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T\,+\,\boldsymbol{\Psi}\right)^{-1}\left(\frac{1}{l}\sum_{i=1}^{l}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T\right)\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T\,+\,\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Lambda}\;=$$
$$\boldsymbol{\Lambda}^T\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T\,+\,\boldsymbol{\Psi}\right)^{-1}\boldsymbol{C}\,\left(\boldsymbol{\Lambda}\,\boldsymbol{\Lambda}^T\,+\,\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Lambda}\;=$$
$$\left(\boldsymbol{A}\,-\,\boldsymbol{A}\,(\boldsymbol{I}\,+\,\boldsymbol{B})^{-1}\,\boldsymbol{B}\right)^T\boldsymbol{C}\left(\boldsymbol{A}\,-\,\boldsymbol{A}\,(\boldsymbol{I}\,+\,\boldsymbol{B})^{-1}\,\boldsymbol{B}\right)\,.$$

Using these equations the E-step and the M-step can be unified and all sums $\sum_{i=1}^{l}$ are removed and the matrix $\boldsymbol{C}$ can be computed once at the beginning of the iterative procedure.

MAP factor analysis. This algorithm can be generalized to a maximum a posteriori method with posterior $p(\boldsymbol{\Lambda}, \boldsymbol{\Psi} \mid \{\boldsymbol{x}\})$ which is proportional to the product between the likelihood $p(\{\boldsymbol{x}\} \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})$ and the prior $p(\boldsymbol{\Lambda})$:

$$p(\boldsymbol{\Lambda}, \boldsymbol{\Psi} \mid \{\boldsymbol{x}\}) \;\propto\; p(\{\boldsymbol{x}\} \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})\, p(\boldsymbol{\Lambda})\,, \tag{10.74}$$

therefore up to a constant independent of the parameters the log-posterior is

$$\log\left(p(\boldsymbol{\Lambda}, \boldsymbol{\Psi} \mid \{\boldsymbol{x}\})\right) \;=\; \log\left(p(\{\boldsymbol{x}\} \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi})\right) \;+\; \log\left(p(\boldsymbol{\Lambda})\right)\,. \tag{10.75}$$

An example for the prior on λ_j is a rectified Gaussian $\mathcal{N}_{\text{rect}}\left(\mu_\Lambda, \sigma_\Lambda\right)$ in order to allow only positive factor loading values which assume that the factors are only additive:

$$y_j \;\sim\; \mathcal{N}\left(\mu_\Lambda, \sigma_\Lambda\right) \tag{10.76}$$
$$\lambda_j \;=\; \max\{y_j, 0\}\,. \tag{10.77}$$

MAP factors. The E-step gives also the most probable values for the factors $\boldsymbol{z}$. This can be important for analyzing data and extracting hidden causes.

Interpretation. We already mentioned that the data variance is explained through signal variance and through noise. The *communality* c_j of an observations variable x_j is

$$c_j \;=\; \frac{\text{var}(x_j) \;-\; \text{var}(\epsilon_j)}{\text{var}(x_j)} \;=\; \frac{\Psi_{jj}}{\Psi_{jj} \;+\; \sum_{k=1}^{p}\lambda_{jk}^2} \tag{10.78}$$

which is the proportion in x_i explained by the factors. Here each factor z_l contributes

$$\frac{\lambda_{jl}^2}{\Psi_{jj} \;+\; \sum_{k=1}^{p}\lambda_{jk}^2}\,. \tag{10.79}$$

10.5 Projection Pursuit and Multidimensional Scaling

10.5.1 Projection Pursuit

Projection pursuit attempts to find "interesting" projections of the data in order to visualize, cluster the data or for dimensionality reduction.

"Interesting" is defined as the least Gaussian distribution. The important question is how to define non-Gaussianity.

If the covariance of a zero mean variable $\boldsymbol{y}$ is fixed, then a Gaussian distribution maximizes the entropy $H(\boldsymbol{y})$.

Then for $y = \boldsymbol{w}^T\boldsymbol{x}$ the vector $\boldsymbol{w}$ must be found which maximized $H(y)$ if y is normalized to zero mean and unit variance.

However the density of $y = \boldsymbol{w}^T\boldsymbol{x}$ is difficult to estimate.

Other more practical measures of non-Gaussianity have been given in as independent component analysis was introduced in Subsection 10.3.1.

10.5.2 Multidimensional Scaling

Multidimensional Scaling (MDS) aims at representing data points $\boldsymbol{x}$ by $\boldsymbol{y}$ in a lower dimensional space so that he distances between the $\boldsymbol{y}$'s correspond to the distances (dissimilarities) between the $\boldsymbol{x}$'es.

We define

$$\boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{w}) \tag{10.80}$$

$$\delta_{ij} = \left\|\boldsymbol{x}^i - \boldsymbol{x}^j\right\| \tag{10.81}$$

$$d_{ij} = \left\|\boldsymbol{y}^i - \boldsymbol{y}^j\right\| . \tag{10.82}$$

The goal is to define a measure which measures the difference between δ and d which is mostly done by a weighted sum of the differences between δ_{ij} and d_{ij}.

Possible measures are

$$R_1(d,\delta) = \frac{\sum_{i<j}(d_{ij} - \delta_{ij})^2}{\sum_{i<j}\delta_{ij}^2} \propto \sum_{i<j}(d_{ij} - \delta_{ij})^2 \tag{10.83}$$

$$R_2(d,\delta) = \sum_{i<j}\left(\frac{d_{ij} - \delta_{ij}}{\delta_{ij}}\right)^2 \tag{10.84}$$

$$R_3(d,\delta) = \frac{1}{\sum_{i<j}\delta_{ij}}\sum_{i<j}\frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}} \propto \sum_{i<j}\frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}} , \tag{10.85}$$

where "$\propto$" removed factors which are constant in the parameters $\boldsymbol{w}$.

The measure R_1 is basically the mean squared error and penalizes large errors even if the δ_{ij} are large. The measure R_2 measures the fractional errors (relative errors) but small δ_{ij} may increase the relative error. R_3 is called "Sammon mapping" and is an compromise of R_1 and R_2.

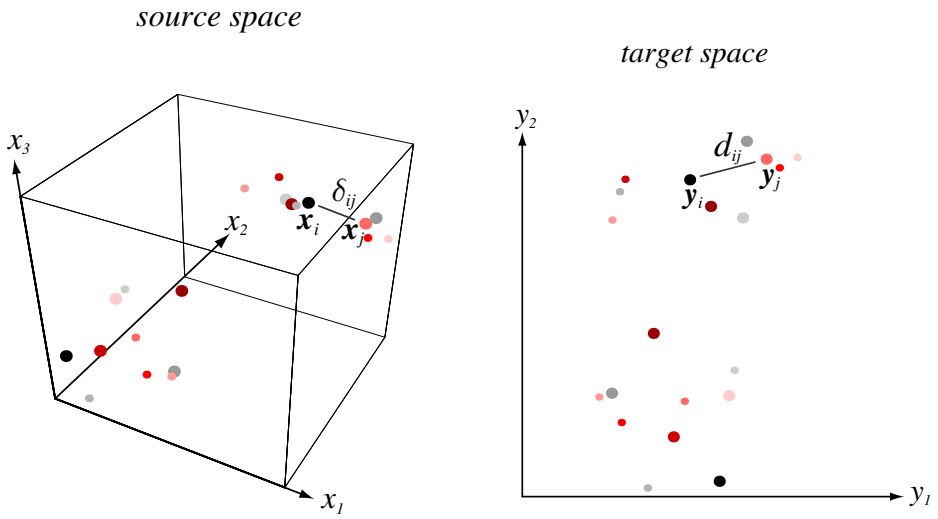


Figure 10.12: Example for multidimensional scaling. Points $\boldsymbol{x}$ from a 3-dimensional space (left) are mapped by multidimensional scaling to a 2-dimensional space (right). Copyright © 2001 John Wiley & Sons, Inc.

The derivatives which can be used in gradient based methods are

$$\frac{\partial}{\partial \boldsymbol{y}_k} R_1(d,\delta) = \frac{2}{\sum_{i<j} \delta_{ij}^2} \sum_{j \neq k} (d_{kj} - \delta_{kj}) \frac{\boldsymbol{y}_k - \boldsymbol{y}_j}{d_{kj}} \tag{10.86}$$

$$\frac{\partial}{\partial \boldsymbol{y}_k} R_2(d,\delta) = 2 \sum_{j \neq k} \frac{d_{kj} - \delta_{kj}}{\delta_{kj}^2} \frac{\boldsymbol{y}_k - \boldsymbol{y}_j}{d_{kj}} \tag{10.87}$$

$$\frac{\partial}{\partial \boldsymbol{y}_k} R_3(d,\delta) = \frac{2}{\sum_{i<j} \delta_{ij}} \sum_{j \neq k} \frac{d_{kj} - \delta_{kj}}{\delta_{kj}} \frac{\boldsymbol{y}_k - \boldsymbol{y}_j}{d_{kj}} \,. \tag{10.88}$$

If the measures are viewed as potential functions, then the derivatives can be considered as forces on certain points $\boldsymbol{y}_k$.

Fig. 10.12 shows and example for multidimensional scaling.

10.6 Clustering

One of the most popular unsupervised learning techniques is clustering. Here "clusters" in the data, that is regions of high data density, are identified. Often these clusters represent typically data points which stem from one "prototype data point" by noise perturbations or represent a certain situation in the real world where the situation is expressed as dependencies in the components.

Clustering extracts structure in the data. Often clustering identifies *new data classes* which were unknown so far.

An important application of clustering is also *data visualization* where is some cases both down-projection and clustering are combined (e.g. for self-organizing maps).

If data points are represented by their prototypes then clustering is a data *compression method* called "vector quantization".

10.6.1 Mixture Models

Because clusters are regions of high data density, density estimators which locally assign a component can be used for clustering. A local component j out of c components has a location $\boldsymbol{\mu}_j$, a width or shape $\boldsymbol{\Sigma}_j$ and a weight w_j giving the local probability mass.

If we consider this as a generative model, then w_j is the probability $p(j)$ of choosing component j. The values $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ describe the local component j which has density $p(\boldsymbol{x} \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$.

If we summarize all parameters $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$, and $\boldsymbol{w}$ in the parameter vector $\boldsymbol{\theta}$, then we obtain the model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) \;=\; \sum_{j=1}^{c} p(j)\; p(\boldsymbol{x} \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\;. \tag{10.89}$$

The log-likelihood is

$$\ln \mathcal{L} \;=\; \sum_{i=1}^{l} \ln p(\boldsymbol{x}^i \mid \boldsymbol{\theta})\;. \tag{10.90}$$

If we summarize $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ in $\boldsymbol{\theta}_j$ then the derivative of the log-likelihood is

$$\frac{\partial}{\partial \boldsymbol{\theta}_j} \ln \mathcal{L} \;=\; \sum_{i=1}^{l} \frac{1}{p(\boldsymbol{x}^i \mid \boldsymbol{\theta})} \sum_{j=1}^{c} p(j) \frac{\partial}{\partial \boldsymbol{\theta}_j} p(\boldsymbol{x}^i \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;= \tag{10.91}$$

$$\sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \frac{\partial}{\partial \boldsymbol{\theta}_j} \ln p(\boldsymbol{x}^i \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\;,$$

where we used Bayes' formula

$$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;=\; \frac{p(\boldsymbol{x}^i \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; p(j)}{p(\boldsymbol{x}^i \mid \boldsymbol{\theta})}\;. \tag{10.92}$$

The derivative of the log-likelihood of the model with respect to the parameters of the j component is posterior expectation of component j of the derivative of the log-likelihood of the component j.

These consideration are valid of each mixture model.

We will now consider mixture of Gaussian (MoG), where

$$p(\boldsymbol{x}^i \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;\sim\; \mathcal{N}\left(\boldsymbol{\mu}_j\;,\; \boldsymbol{\Sigma}_j\right)\;. \tag{10.93}$$

The model is

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) \;=\; \sum_{j=1}^{c} w_j \, \mathcal{N}\left(\boldsymbol{\mu}_j \,,\; \boldsymbol{\Sigma}_j\right) \tag{10.94}$$

$$\sum_{j=1}^{c} w_j \;=\; 1 \tag{10.95}$$

$$w_j \;\geq\; 0 \tag{10.96}$$

$$\mathcal{N}\left(\boldsymbol{\mu}_j \,,\; \boldsymbol{\Sigma}_j\right)(\boldsymbol{x}) \;= \tag{10.97}$$
$$(2\pi)^{-d/2} \left|\boldsymbol{\Sigma}_j\right|^{-1/2} \exp\left(-\frac{1}{2}\; (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)^T \;\boldsymbol{\Sigma}_j^{-1} (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)\right) .$$

Gaussian are convenient because in eq. (10.91) the logarithm inverts the exponential function:

$$\ln p(\boldsymbol{x} \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;= \tag{10.98}$$
$$-\frac{d}{2}\; \ln(2\pi) \;-\; \frac{1}{2}\; \ln\left|\boldsymbol{\Sigma}_j\right| \;-\; \frac{1}{2}\; (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)^T \;\boldsymbol{\Sigma}_j^{-1} (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)$$

which gives for the derivatives

$$\frac{\partial}{\partial \boldsymbol{\mu}_j} \ln p(\boldsymbol{x} \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;=\; \boldsymbol{\Sigma}_j^{-1} (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j) \tag{10.99}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_j} \ln p(\boldsymbol{x} \mid j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;= \tag{10.100}$$
$$\frac{1}{2}\; \left(\boldsymbol{\Sigma}_j^T\right)^{-1} \;+\; \frac{1}{2}\; \boldsymbol{\Sigma}_j^{-T} (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)\; (\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-T} .$$

Here also the EM-algorithm can be used where the hidden parameters are $p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ must be estimated to evaluate eq. (10.91).

The EM-algorithm is

E-step: (10.101)

$$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;=\; \frac{w_j \, \mathcal{N}\left(\boldsymbol{\mu}_j \,,\; \boldsymbol{\Sigma}_j\right)(\boldsymbol{x}^i)}{\sum_{l=1}^{c} w_l \, \mathcal{N}\left(\boldsymbol{\mu}_l \,,\; \boldsymbol{\Sigma}_l\right)(\boldsymbol{x}^i)}$$

M-step: (10.102)

$$\boldsymbol{w}_j^{\text{new}} \;=\; \frac{1}{l} \sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

$$\boldsymbol{\mu}_j^{\text{new}} \;=\; \frac{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; \boldsymbol{x}^i}{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{10.103}$$

$$\boldsymbol{\Sigma}_j^{\text{new}} \;=\; \frac{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; \left(\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right)\; \left(\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right)^T}{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \,. \tag{10.104}$$

In order to avoid too small variances and near zero eigenvalues of $\mathbf{\Sigma}_j$ the mixture of Gaussian can be optimized by a maximum a posterior approach.

A proper prior for the covariance $\mathbf{\Sigma}$ is the Wishart density $\mathcal{W}(\mathbf{\Sigma}^{-1} \mid \alpha, \mathbf{\Psi})$, a proper prior for the weighting factors w_j is a Dirichlet density $\mathcal{D}(\boldsymbol{w} \mid \gamma)$, and a proper prior for the mean values $\boldsymbol{\mu}$ is a Gaussian $\mathcal{N}\left(\boldsymbol{\mu} \mid \boldsymbol{\nu}, \eta^{-1}\mathbf{\Sigma}\right)$:

$$\mathcal{W}(\mathbf{\Sigma}^{-1} \mid \alpha, \mathbf{\Psi}) \;=\; c(\alpha, \mathbf{\Psi}) \;\left|\mathbf{\Sigma}^{-1}\right|^{\alpha-(d+1)/2} \;\exp\left(-\mathrm{tr}\left(\mathbf{\Psi}\;\mathbf{\Sigma}^{-1}\right)\right) \tag{10.105}$$

$$\mathcal{D}(\boldsymbol{w} \mid \gamma) \;=\; c(\gamma) \;\prod_{j=1}^{c} w_j^{\gamma-1} \tag{10.106}$$

$$\mathcal{N}\left(\boldsymbol{\mu} \mid \boldsymbol{\nu}, \eta^{-1}\mathbf{\Sigma}\right) \;= \tag{10.107}$$
$$(2\pi)^{-d/2} \left|\eta^{-1}\;\mathbf{\Sigma}_j\right|^{-1/2} \exp\left(-\frac{\eta}{2}\;\left(\boldsymbol{\mu} \;-\; \boldsymbol{\nu}\right)^T\;\mathbf{\Sigma}_j^{-1}\left(\boldsymbol{\mu} \;-\; \boldsymbol{\nu}\right)\right)\;,$$

where $\alpha > (d-1)/2$ and $c(\gamma)$ as well as $c(\alpha, \mathbf{\Psi})$ are normalizing constants. The operator "tr" is the trace operator.

The expectation-maximization algorithm is now

E-step: (10.108)

$$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j) \;=\; \frac{w_j\,\mathcal{N}\left(\boldsymbol{\mu}_j\;,\;\mathbf{\Sigma}_j\right)(\boldsymbol{x}^i)}{\sum_{l=1}^{c} w_l\,\mathcal{N}\left(\boldsymbol{\mu}_l\;,\;\mathbf{\Sigma}_l\right)(\boldsymbol{x}^i)}$$

M-step: (10.109)

$$w_j^{\mathrm{new}} \;=\; \frac{\sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j) \;+\; \gamma \;-1}{l \;+\; c\;(\gamma \;-\; 1)}$$

$$\boldsymbol{\mu}_j^{\mathrm{new}} \;=\; \frac{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j)\;\boldsymbol{x}^i \;+\; \eta\,\boldsymbol{\nu}_j}{\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j) \;+\; \eta} \tag{10.110}$$

$$\mathbf{\Sigma}_j^{\mathrm{new}} \;=\; \left(\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j)\;\left(\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right)\;\left(\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right)^T \;+ \right. \tag{10.111}$$
$$\left.\eta\;\left(\boldsymbol{\nu}_j \;-\; \boldsymbol{\mu}_j\right)\;\left(\boldsymbol{\nu}_j \;-\; \boldsymbol{\mu}_j\right)^T \;+\; 2\;\mathbf{\Psi}\right)$$
$$\left(\sum_{i=1}^{l}\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j) \;+\; 2\;\alpha \;-d\right)^{-1} \quad .$$

Above formulae are obtained as follows: Because

$$\sum_{j=1}^{c} w_j^{\mathrm{new}} \;=\; 1 \tag{10.112}$$

we obtain as Lagrangian for the constraint optimization problem for the w_j

$$L \;=\; \sum_{i=1}^{l}\sum_{j=1}^{c} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; \ln w_j^{\text{new}} \;+\; \ln \mathcal{D}(\boldsymbol{w} \mid \gamma) \;- \tag{10.113}$$
$$\lambda \left(\sum_{j=1}^{c} w_j^{\text{new}} \;-\; 1 \right) .$$

Setting the derivative to zero:

$$\frac{\partial L}{\partial w_j} \;=\; \sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; \left(w_j^{\text{new}}\right)^{-1} \;+ \tag{10.114}$$
$$(\gamma \;-\; 1)\; \left(w_j^{\text{new}}\right)^{-1} \;-\; \lambda \;=\; 0$$
$$\sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;+\; (\gamma \;-\; 1) \;=\; \lambda\, w_j^{\text{new}}$$

Summing over j gives

$$\sum_{i=1}^{l}\sum_{j=1}^{c} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;+\; c\,(\gamma \;-\; 1) \;=\; \lambda \sum_{j=1}^{c} w_j^{\text{new}} \tag{10.115}$$
$$l \;+\; c\,(\gamma \;-\; 1) \;=\; \lambda \tag{10.116}$$

We obtain

$$w_j^{\text{new}} \;=\; \frac{\sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;+\; (\gamma \;-\; 1)}{l \;+\; c\,(\gamma \;-\; 1)} \tag{10.117}$$

For the other parameters we do not have constraints. The gradient of the log-posterior LP with respect to $\boldsymbol{\mu}_j$ contains the log-likelihood and the log-prior:

$$\frac{\partial \text{LP}}{\partial \boldsymbol{\mu}_j} \;=\; \sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\; \boldsymbol{\Sigma}_j^{-1}\; \left(\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right) \;+ \tag{10.118}$$
$$\eta\, \boldsymbol{\Sigma}_j^{-1}\; (\boldsymbol{\nu}_j \;-\; \boldsymbol{\mu}_j) \;=\; 0\,.$$

For the gradient with respect to $\boldsymbol{\Sigma}_j$ a trick is applied: the gradient is taken with respect to $\boldsymbol{\Sigma}_j^{-1}$ with also must be zero at the minimum, because

$$\frac{\partial \text{LP}}{\partial \boldsymbol{\Sigma}_j} \;=\; -\,\boldsymbol{\Sigma}_j^{-2}\, \frac{\partial \text{LP}}{\partial \boldsymbol{\Sigma}_j^{-1}} \tag{10.119}$$

and the variables $\boldsymbol{\Sigma}_j^{-1}$ fully represent the variables $\boldsymbol{\Sigma}_j$.

We obtain

$$\frac{\partial \mathrm{LP}}{\partial \boldsymbol{\Sigma}_j^{-1}} = \tag{10.120}$$
$$\frac{1}{2}\sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \left(\boldsymbol{\Sigma}_j - \left(\boldsymbol{x}^i - \boldsymbol{\mu}_j\right)\left(\boldsymbol{x}^i - \boldsymbol{\mu}_j\right)^T\right) +$$
$$\frac{1}{2}\left(\boldsymbol{\Sigma}_j - \eta\left(\boldsymbol{\nu}_j - \boldsymbol{\mu}_j\right)\left(\boldsymbol{\nu}_j - \boldsymbol{\mu}_j\right)^T\right) +$$
$$\boldsymbol{\Sigma}_j\left(\alpha - (d+1)/2\right) - \boldsymbol{\Psi} = 0\,,$$

where we used

$$\frac{\partial \ln|\boldsymbol{A}|}{\partial \boldsymbol{A}} = \boldsymbol{A}^{-1} \tag{10.121}$$

for $\boldsymbol{A} = \boldsymbol{\Sigma}_j^{-1}$.

Note that each component can have prior so that we would obtain $\boldsymbol{\nu}_j$, η_j, α_j, $\boldsymbol{\Psi}_j$, and γ_j. The update formulae would we similar as above.

Default values for the hyperparameters are

$$\alpha = \frac{d}{2} \tag{10.122}$$
$$\boldsymbol{\Psi} = \frac{1}{2}\boldsymbol{I} \text{ OR } \boldsymbol{\Psi} = \frac{1}{2}\mathrm{covar}(\boldsymbol{x}) \tag{10.123}$$
$$\gamma = 1 \tag{10.124}$$
$$\eta = 0 \tag{10.125}$$
$$\boldsymbol{\nu}_j = \mathrm{mean}(\boldsymbol{x}) \tag{10.126}$$

A prior on the mean values $\boldsymbol{\mu}$ is in most cases not useful except a preferred region is known.

The posterior $p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ can be used for clustering: $\boldsymbol{x}^i$ belongs to the cluster j for which the posterior is largest.

But also soft clustering is possible: $p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ gives the graded or fuzzy membership of $\boldsymbol{x}^i$ to the cluster j.

10.6.2 k-Means Clustering

$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ in eq. (10.101) is determined by the weight w_j of the jth component and the distance $\left(\boldsymbol{x}^i - \boldsymbol{\mu}_j\right)^T \boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{x}^i - \boldsymbol{\mu}_j\right)$ of $\boldsymbol{x}^i$ to the mean $\boldsymbol{\mu}_j$.

If we set $w_j = \frac{1}{c}$ and $\boldsymbol{\Sigma}_j^{-1} = \boldsymbol{I}$, then x^i belongs to the cluster j which center $\boldsymbol{\mu}_j$ has the smallest Euclidean distance $\left\|\boldsymbol{x}^i - \boldsymbol{\mu}_j\right\|$ to $\boldsymbol{x}^i$.

If we discretize

$$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j) = \begin{cases} 1 & \text{if } j = c_{\boldsymbol{x}^i} = \arg\min_l \left\|\boldsymbol{x}^i - \boldsymbol{\mu}_l\right\| \\ 0 & \text{otherwise} \end{cases} \tag{10.127}$$

then the M-step in eq. (10.102) is

$$\boldsymbol{\mu}_j^{\text{new}} \;=\; \frac{1}{l_j} \sum_{i=1,\; j=c_{\boldsymbol{x}^i}}^{l} \boldsymbol{x}^i \tag{10.128}$$

$$l_j \;=\; \sum_{i=1}^{l} p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \;=\; \sum_{i=1,\; j=c_{\boldsymbol{x}^i}}^{l} 1 \;, \tag{10.129}$$

where l_j is the number of data points assigned to cluster j.

Therefore $\boldsymbol{\mu}_j^{\text{new}}$ is the mean of the data points assigned to cluster j. This method is called *k-means clustering* and its algorithm is given in Alg. 10.1.

Algorithm 10.1 k-means

Given: data $\{\boldsymbol{x}\} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^l)$, number of clusters c

BEGIN initialization
initialize the cluster centers $\boldsymbol{\mu}_j$, $1 \leq j \leq c$
END initialization

BEGIN Iteration

Stop=false
while Stop=false **do**
for $(i = 1 \;;\; i \;\geq\; l \;;\; i++)$ **do**
assign $\boldsymbol{x}^i$ to the nearest $\boldsymbol{\mu}_j$
end for
for $(j = 1 \;;\; j \;\geq\; c \;;\; j++)$ **do**

$$\boldsymbol{\mu}_j^{\text{new}} \;=\; \frac{1}{l_j} \sum_{i=1,\; j=c_{\boldsymbol{x}^i}}^{l} \boldsymbol{x}^i$$

end for
if stop criterion fulfilled **then**
Stop=true
end if
end while
END Iteration

The k-means clustering is fast but it is prone to initial initialization. For example consider an initialization which places one center near an outlier data point which is separated from the rest of the data points which all have other cluster centers closer to them. Then this outlier cluster will contain in each iteration only one data point.

This behavior can be serious in high dimensions.

Let us again assume $w_j = \frac{1}{c}$ and $\boldsymbol{\Sigma}_j^{-1} = \boldsymbol{I}$. But now we use a continuous estimate of $p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j)$, where the distances do not have an exponential decay as in eq. (10.101).

We set

$$p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j) \;=\; \frac{\left\|\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right\|^{-2/(b-1)}}{\sum_{l=1}^{c} \left\|\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_l\right\|^{-2/(b-1)}} \;. \tag{10.130}$$

and obtain

$$\boldsymbol{\mu}_j^{\text{new}} \;=\; \frac{\sum_{i=1}^{l} p^b(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j)\; \boldsymbol{x}^i}{\sum_{i=1}^{l} p^b(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j)} \;. \tag{10.131}$$

The error function which is minimized is

$$\sum_{l=1}^{c} \sum_{i=1}^{l} p^b(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j)\; \boldsymbol{x}^i \;\left\|\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right\|^2 \;. \tag{10.132}$$

This algorithm is called *fuzzy k-means clustering* and described in Alg. 10.2.

Algorithm 10.2 Fuzzy k-means

Given: data $\{\boldsymbol{x}\} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^l)$, number of clusters c

BEGIN initialization
initialize the cluster centers $\boldsymbol{\mu}_j$, $1 \leq j \leq c$, and $w_j(\boldsymbol{x}^i) \;=\; p(j \mid \boldsymbol{x}^i, \boldsymbol{\mu}_j)$ so that $\sum_{j=1}^{c} w_j(\boldsymbol{x}^i) = 1$, $w_j(\boldsymbol{x}^i) \geq 0$.
END initialization

BEGIN Iteration

Stop=false
while Stop=false **do**

$$\boldsymbol{\mu}_j^{\text{new}} \;=\; \frac{\sum_{i=1}^{l} w_j^b(\boldsymbol{x}^i)\; \boldsymbol{x}^i}{\sum_{i=1}^{l} w_j^b(\boldsymbol{x}^i)} \;.$$

$$w_j(\boldsymbol{x}^i) \;=\; \frac{\left\|\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_j\right\|^{-2/(b-1)}}{\sum_{l=1}^{c} \left\|\boldsymbol{x}^i \;-\; \boldsymbol{\mu}_l\right\|^{-2/(b-1)}}$$

if stop criterion fulfilled **then**
Stop=true
end if
end while
END Iteration

10.6.3 Hierarchical Clustering

Until now we did not consider distances and structures between the clusters. Structures between clusters can be obtained through *hierarchical clustering* where in a dendrogram also the neighborhood between clusters is shown.

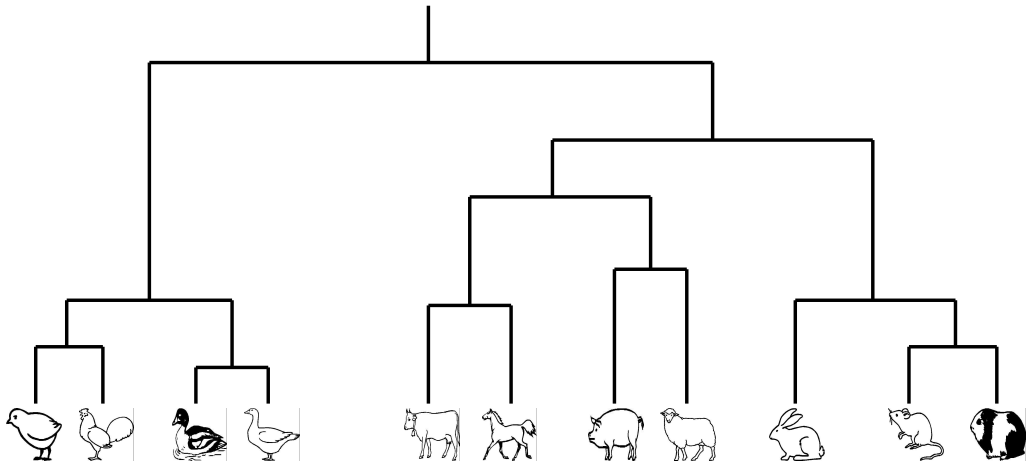


Figure 10.13: Example for hierarchical clustering given as a dendrogram (corresponding tree) of animal species.

Fig. 10.13 shows and example for hierarchical clustering where a dendrogram is depicted.

Hierarchical clustering is already known from "Bioinformatics 1" where we constructed a phylogenetic tree. The method "Unweighted Pair Group Method using arithmetic Averages" (UPGMA) was an example of hierarchical clustering in order to construct a phylogenetic tree.

In machine learning the UPGMA method is called *agglomerative hierarchical clustering*, where the closest clusters are merged do give a new cluster. Starting point is where each cluster consists of a single element.

This method can be varied by using different distance measures between clusters A and B

$$d_{\min}(A,B) = \min_{\boldsymbol{a}\in A,\boldsymbol{b}\in B} \|\boldsymbol{a} - \boldsymbol{b}\| \tag{10.133}$$

$$d_{\max}(A,B) = \max_{\boldsymbol{a}\in A,\boldsymbol{b}\in B} \|\boldsymbol{a} - \boldsymbol{b}\| \tag{10.134}$$

$$d_{\text{avg}}(A,B) = \frac{1}{l_A\, l_B} \sum_{\boldsymbol{a}\in A} \sum_{\boldsymbol{b}\in B} \|\boldsymbol{a} - \boldsymbol{b}\| \tag{10.135}$$

$$d_{\text{mean}}(A,B) = \|\bar{\boldsymbol{a}} - \bar{\boldsymbol{b}}\|\,, \tag{10.136}$$

where l_A (l_B) is the number of elements in A (B) and $\bar{\boldsymbol{a}}$ ($\bar{\boldsymbol{b}}$) is the mean of cluster A (B).

For single elements these distance measures are equivalent, however for clusters there is a difference. The use of $d_{\max}$ avoids that clusters are elongated in some direction but the smallest distance between their points remains small.

UPGMA is based on $d_{\min}$, therefore, it is known in machine learning as "nearest neighbor algorithm".

Also the "Neighbor Joining" algorithm from "Bioinformatics 1" to construct a phylogenetic tree falls into the class of "nearest neighbor algorithm".

The use of $d_{\max}$ measures how broad are two clusters together and avoids that extreme points of two clusters are widely separated if the clusters have small distance.

Other hierarchical clustering methods can be based on graph theoretic considerations. On first builds the minimal spanning tree. Then the largest edge is removed which gives two clusters. Now the second largest edge can be removed and so on. However a better procedure may be to compute the average edge length of the clusters and find an edge which is considerably large than other edges in the cluster. This procedure can be done at node level, where for each node the edge is determined which is considerably larger than other edges of this node. The inconsistent (considerably larger) edges can be removed stepwise and new clusters are produced.

10.6.4 Self-Organizing Maps

A method which is similar to multidimensional scaling is the *Self-Organizing Map* (SOM) also called *Kohonen map*.

SOMs comprise two objectives: clustering and down-projecting. Data points $\boldsymbol{x} \in \mathbb{R}^d$ are clustered and down-projected to points $\boldsymbol{y} \in \mathbb{R}^l$ with $d > l$. Because of the clustering only finite many $\boldsymbol{y}$ exist, namely $\boldsymbol{y}^k$.

In almost all applications the $\boldsymbol{y}^k$ equidistantly fill a hypercube in $\mathbb{R}^l$. For each $\boldsymbol{y}^k$ there exist an associated $\boldsymbol{w}^k \in \mathbb{R}^d$ representing the cluster center in the data space.

Goal now is to find cluster centers $\boldsymbol{w}^k$ that for data points $\boldsymbol{x}$ which are neighbors in $\mathbb{R}^d$ their projections are also neighbors in $\mathbb{R}^l$. The goal is to down-project but preserving the neighborhood relation which gave these methods also the name "topologically ordered maps".

The learning can be done on-line, that is each new data point $\boldsymbol{x}$ leads to an update of the $\boldsymbol{w}^k$. The update rule is

$$k \;=\; \arg\max_m \; \boldsymbol{x}^T \boldsymbol{w}^m \tag{10.137}$$

$$(\boldsymbol{w}^n)^{\text{new}} \;=\; \boldsymbol{w}^n \;+\; \eta\, \delta \left(\left\| \boldsymbol{y}^n \;-\; \boldsymbol{y}^k | \right\| \right) \; (\boldsymbol{x} \;-\; \boldsymbol{w}^n) \;, \tag{10.138}$$

where η is the learning rate which also depends on the iteration number and is annealed and δ is the "window function" which is largest for $\boldsymbol{y}^n \;=\; \boldsymbol{y}^k$ and is decreasing with the distance to $\boldsymbol{y}^k$.

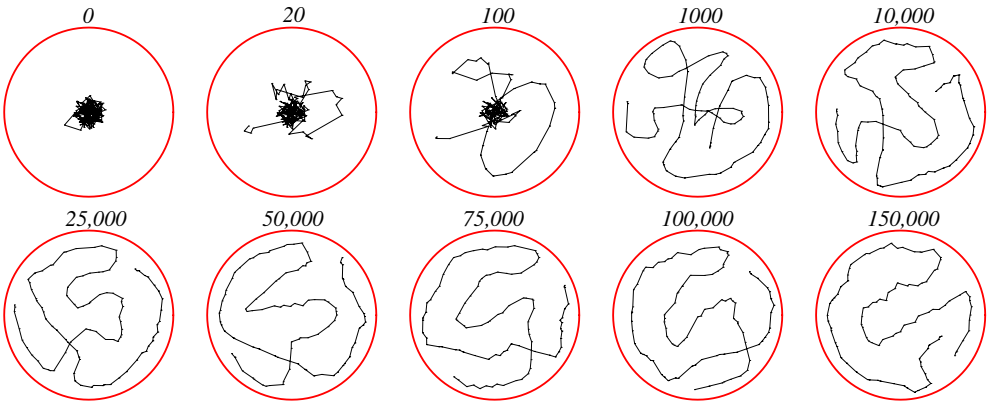


Figure 10.14: Self-Organizing Map. Example of a one-dimensional representation of a two-dimensional space. Copyright © 2001 John Wiley & Sons, Inc.

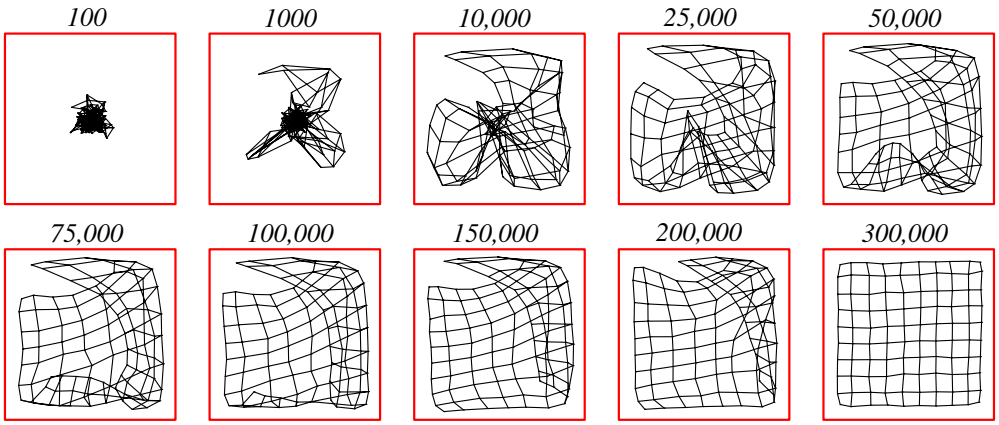


Figure 10.15: Self-Organizing Map. Mapping from a square data space to a square (grid) representation space. Copyright © 2001 John Wiley & Sons, Inc.

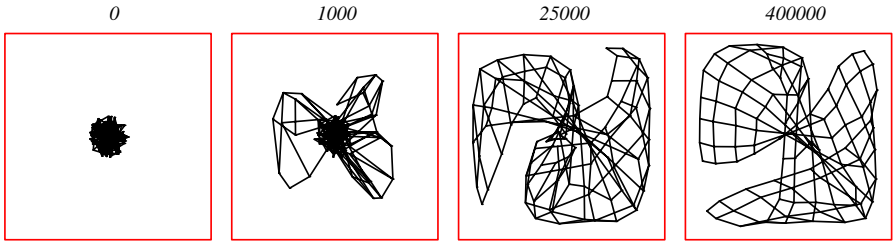


Figure 10.16: Self-Organizing Map. The problem from Fig. 10.14 but with different initialization. Kinks in the map do not vanish even if more patterns are presented – that is a local minimum. Copyright © 2001 John Wiley & Sons, Inc.

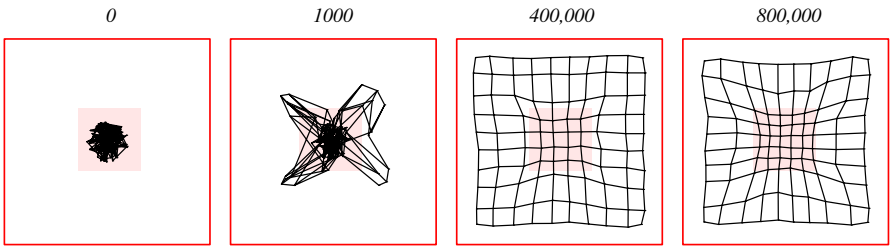


Figure 10.17: Self-Organizing Map. The problem from Fig. 10.14 but with a non-uniformly sampling: the density at the center was higher than at the border.

Bibliography

B. Adam, Y. Qu, J. W. Davis, M. D. Ward, M. A. Clements, L. H. Cazares, O. J. Semmes, P. F. Schellhammer, Y. Yasui, Z. Feng, and Jr. G. L. Wright. Serum protein fingerprinting coupled with a pattern-matching algorithm distinguishes prostate cancer from benign prostate hyperplasia and healthy men. *Cancer Research*, 62(13):3609–3614, 2002.

H. Alashwal, S. Deris, and R. M. Othman. One-class support vector machines for protein-protein interactions prediction. *International Journal of Biomedical Sciences*, 1(2):120–127, 2006.

L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 609–618, 1987.

H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 547–552, Anaheim, California, 1991. AAAI Press.

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6562–6566, 2002.

D. C. Anderson, W. Li, D. G. Payan, and W. S. Noble. A new algorithm for the evaluation of shotgun peptide sequencing in proteomics: support vector machine classification of peptide MS/MS spectra and SEQUEST scores. *Journal of Proteome Research*, 2(2):137–146, 2003.

H. Attias. Independent Factor Analysis. *Neural Computation*, 11(4):803–851, 1999.

A. D. Back and A. C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modelling. *Neural Computation*, 3:375–385, 1991.

W. Bains and G. Smith. A novel method for nucleic acid sequence determination. *Journal of Theoretical Biology*, 135:303–307, 1988.

J. Bala, K. A. D. Jong, J. Haung, H. Vafaie, and H. Wechsler. Hybrid learning usnig genetic algorithms and decision trees for pattern classification. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 719–724. Morgan Kaufmann Publishers, Inc, 1995.

C. Barrett, R. Hughey, and K. Karplus. Scoring hidden marov models. *CABIOS*, 13(2):191–19, 1997.

A. Bateman, E. Birney, R. Durbin, S. R. Eddy, K. L. Howe, and E. L. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Res.*, 28:263–266, 2000.

A. Bateman, L. Coin, R. Durbin, R. D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E. L. L. Sonnhammer, D. J. Studholme, C. Yeats, and S. R. Eddy. The pfam protein families database. *Nucleic Acids Research*, 32:D138–141, 2004.

A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

A. Ben-Hur and D. Brutlag. Remote homology detection: A motif based approach. In *Eleventh International Conference on Intelligent Systems for Molecular Biology*, pages i26–i33. volume 19 suppl 1 of Bioinformatics, 2003.

A. Ben-Hur and D. Brutlag. Sequence motifs: Highly predictive features of protein function. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, pages 625–645. MIT Press, 2004.

J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003. Special Issue on Variable and Feature Selection.

C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

C. M. Bishop. Curvature-driven smoothing: A learning algorithm for feed-forward networks. *IEEE Transactions on Neural Networks*, 4(5):882–884, 1993.

A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271, 1997.

J. R. Bock and D. A. Gough. Predicting protein-protein interactions from primary structure. *Bioinformatics*, 17:455–460, 2001.

U. Bodenhausen. Learning internal representations of pattern sequences in a neural network with adaptive time-delays. In *Proceedings of the International Joint Conference on Neural Networks*. Hillsdale, NJ. Erlbaum, 1990.

U. Bodenhausen and A. Waibel. The tempo 2 algorithm: Adjusting time-delays by supervised learning. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 155–161. San Mateo, CA: Morgan Kaufmann, 1991.

B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences*, 97(1):262–267, 2000.

C. Burge and S. Karlin. Prediction of complete gene structures in human genomic dna. *J. Mol. Biol.*, 268:78–94, 1997.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

C. Cardie. Using decision trees to improve case-based learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 25–32. Morgan Kaufmann Publishers, Inc., 1993.

J.-F. Cardoso. Infomax and maximum likelihood for source separation. *IEEE Signal Processing Letters*, 4:112–114, 1997.

J.-F. Cardoso and B. Laheld. Equivariant adaptive source separation. *IEEE Transactions on Signal Processing*, 44:3017–3030, 1996.

J.-F. Cardoso and A. Souloumiac. Blind beamforming for non Gaussian signals. *IEE Proceedings-F*, 140(6):362–370, 1993.

M. J. Carter, F. J. Rudolph, and A. J. Nucci. Operational fault tolerance of CMAC networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 340–347. San Mateo, CA: Morgan Kaufmann, 1990.

R. J. Carter, I. Dubchak, and S. R. Holbrook. A computational approach to identify genes for functional RNAs in genomic sequences. *Nucleic Acids Research*, 29(19):3928–3938, 2001.

R. Caruana and D. Freitag. Greedy attribute selection. In *International Conference on Machine Learning*, pages 28–36, 1994.

A. Cichocki, R. Unbehauen, L. Moszczynski, and E. Rummert. A new on-line adaptive algorithm for blind separation of source signals. In *Proc. Int. Symposium on Artificial Neural Networks, ISANN-94*, pages 406–411, 1994.

P. Comon. Independent component analysis – a new concept? *Signal Processing*, 36(3):287–314, 1994.

T. P. Conrads, M. Zhou, E. F. Petricoin III, L. Liotta, and T. D. Veenstra. Cancer diagnosis using proteomic patterns. *Expert Reviews in Molecular Diagnostics*, 3(4):411–420, 2003.

C. Cortes and V. N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

M. Cottrell, J. C. Fort, and G. Pages. Comments about "Analysis of the convergence properties of topology preserving neural networks". *IEEE Transactions on Neural Networks*, 6(3):797–799, 1995.

T. M. Cover and J. M. V. Campenhout. On the possible orderings in the measurement selection problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(9):657–661, 1977.

S. Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the 18th International Conference on Machine Learning*, pages 74–81. Morgan Kaufmann, San Francisco, CA, 2001.

B. de Vries and J. C. Principe. A theory for neural networks with time delays. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 162–168. San Mateo, CA: Morgan Kaufmann, 1991.

G. Deco and W. Brauer. Higher order statistical decorrelation without information loss. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 247–254. The MIT Press, 1995.

S. Degroeve, B. De Baets, Y. Van de Peer, and P. Rouz. Feature subset selection for splice site prediction. *Bioinformatics*, 18:S75–S83, 2002.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society B*, 39(1):1–22, 1977.

E. Diamandis. Proteomic patterns in biological fluinds: Do they represent the future of cancer diagnostics. *Clinical Chemistry (Point/CounterPoint)*, 48(8):1272–1278, 2003.

R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14:755–763, 1998.

S. R. Eddy. What is a hidden markov model? *Nature Biotechnology*, 22:1315–1316, 2004.

S.R. Eddy and R. Durbin. Maximum discrimination hidden Markov models of sequence consensus. *J. Comp. Biol.*, 2:9–23, 1995.

M. Eisen, P. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95:14863–14868, 1998.

J. L. Elman. Finding structure in time. Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.

E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.

B. S. Everitt. *An introduction to latent variable models*. Chapman and Hall, London, 1984.

L. Falquet, M. Pagni, P. Bucher, N. Hulo, C. J. Sigrist, K. Hofmann, and A. Bairoch. The PROSITE database, its status in 2002. *Nucleic Acids Research*, 30:235–238, 2002.

G. W. Flake and B. A. Pearlmutter. Differentiating functions of the Jacobian with respect to the weights. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 435–441. The MIT Press, Cambridge, MA, London, England, 2000.

B. Flower and M. Jabri. Summed weight neuron perturbation: An $O(N)$ improvement over weight perturbation. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 212–219. Morgan Kaufmann, San Mateo, CA, 1993.

P. Frasconi, M. Gori, and G. Soda. Local feedback multilayered networks. *Neural Computation*, 4:120–130, 1992.

J. H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987.

J. H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, 1981.

J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–890, 1974.

T.-T. Frieß and R. F. Harrison. Linear programming support vector machines for pattern classification and regression estimation and the set reduction algorithm. TR RR-706, University of Sheffield, Sheffield, UK, 1998.

K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

T. S. Furey, N. Duffy, N. Cristianini, D. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

D. Gerhold, T. Rushmore, and C. T. Caskey. DNA chips: promising toys have become powerful tools. *Trends in Biochemical Science*, 24(5):168–173, 1999.

Z. Ghahramani and M. I. Jordan. Factorial hidden markov models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 472–478. The MIT Press, 1996.

Z. Ghahramani and M. I. Jordan. Factorial hidden markov models. *Machine Learning*, 29:245, 1997.

M. Gherrity. A learning algorithm for analog fully recurrent neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 1, pages 643–644, 1989.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

M. Gori, Y. Bengio, and R. DeMori. BPS: a learning algorithm for capturing the dynamic nature of speech. In *Proceedings of the International Joint Conference on Neural Networks*, pages 417–423, 1989.

M. Gribskov, R. Lüthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology*, 183:146–159, 1990.

M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, pages 4355–4358, 1987.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. Special Issue on Variable and Feature Selection.

I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, editors. *Feature Extraction – Foundations and Applications*. Number 207 in Studies in Fuzziness and Soft Computing. Springer, Berlin, Heidelberg, 2006.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. San Mateo, CA: Morgan Kaufmann, 1989.

A. Hartemink, D. Gifford, T. Jaakkola, and R. Young. Maximum likelihood estimation of optimal scaling factors for expression array normalization. In *Microarrays: Optical Technologies and Informatics, Proceedings of SPIE*, volume 4266, 2001.

V. Hasselblad. Estimation of parameters for a mixture of normal distributions. *Technometrics*, 8: 431–444, 1966.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. San Mateo, CA: Morgan Kaufmann, 1993.

T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84: 502–516, 1989.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, New York, 2001.

G. Hinton and T. J. Sejnowski. Introduction. In G. Hinton and T. J. Sejnowski, editors, *Unsupervised Learning: Foundations of Neural Computation*, pages VII–XVI. MIT Press, Cambridge, MA, London, England, 1999.

G. E. Hinton and T. E. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1, pages 282–317. MIT Press, 1986.

S. Hochreiter and M. C. Mozer. A discrete probabilistic memory model for discovering dependencies in time. In G. Dorffner, H. Bischof, and K. Hornik, editors, *International Conference on Artificial Neural Networks*, pages 661–668. Springer, 2001.

S. Hochreiter and K. Obermayer. Gene selection for microarray data. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, pages 319–355. MIT Press, 2004.

S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997a.

S. Hochreiter and J. Schmidhuber. Unsupervised coding with Lococode. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings of the International Conference on Artificial Neural Networks, Lausanne, Switzerland*, pages 655–660. Springer, 1997b.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Y. Hou, W. Hsu, M. L. Lee, and C. Bystroff. Remote homolog detection using local sequence-structure correlations. *Proteins: Structure, Function and Bioinformatics*, 57:518–530, 2004.

S. Hua and Z. Sun. A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach. *Journal of Molecular Biology*, 308:397–407, 2001a.

S. Hua and Z. Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17(8):721–728, 2001b.

J.Y. Huang and D.L. Brutlag. The eMOTIF database. *Nucleic Acids Research*, 29(1):202–204, 2001.

T.M. Huang and V. Kecman. Gene extraction for cancer diagnosis by support vector machines - an improvement. *Artificial Intelligence in Medicine*, 2005.

P. J. Huber. *Robust Statistics*. John Wiley and Sons, New York, 1981.

P. J. Huber. Projection pursuit. *Annals of Statistics*, 13(2):435–475, 1985.

A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999.

N. Intrator. Combining exploratory projection pursuit and projection pursuit regression with application to neural networks. *Neural Computation*, 5(3):443–457, 1993.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

J. E. Jackson. *A User's Guide to Principal Components*. Wiley, New York, 1991.

J. Jäger, R. Sengupta, and W. L. Ruzzo. Improved gene selection for classification of microarrays. In *Biocomputing - Proceedings of the 2003 Pacific Symposium*, pages 53–64, 2003.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *International Conference on Machine Learning*, pages 121–129, 1994.

I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.

L. K. Jones. On a conjecture of Huber concerning the convergence of projection pursuit regression. *Annals of Statistics*, 15:880–882, 1987.

M. C. Jones and R. Sibson. What is projection pursuit? *Journal of the Royal Statistical Society, series A*, 150:1–36, 1987.

M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of Ninth Annual Conference of the Cognitive Science Society, Amherst*, pages 531–546, 1986.

K. G. Jöreskog. Some contributions to maximum likelihood factor analysis. *Psychometrika*, 32: 443–482, 1967.

K. G. Jöreskog. Some contributions to maximum likelihood factor analysis. *Psychometrika*, 32: 443–482, 1967.

C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991.

K. Karplus, C. Barrett, M. Cline, M. Diekhans, L. Grate, and R. Hughey. Predicting protein structure using only sequence information. *Proteins: Structure, Function, and Genetics*, 37 (S3):121–125, 1999.

K. Karplus, C. Barrett, and R. Hughey. Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

W. J. Kent. BLAT - the BLAST like alignment tool. *Genome Biology*, 12(4), 2002.

P. Kerlirzin and F. Vallet. Robustness in multilayer perceptrons. *Neural Computation*, 5(1):473–482, 1993.

M. K. Kerr, M. Martin, and G. A. Churchill. Analysis of variance for gene expression microarray data. *Journal of Computational Biology*, 7:819–837, 2000.

K. Kira and L. A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 129–134. MIT Press, 1992.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2): 273–324, 1997.

T. Kohonen. Analysis of a simple self-organizing process. *Biological Cybernetics*, 43:135–140, 1982.

T. Kohonen. *Self-Organization and Associative Memory*. Springer, second ed., 1988.

T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.

T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

D. Koller and M. Sahami. Toward optimal feature selection. In *International Conference on Machine Learning*, pages 284–292, 1996.

D. Komura, H. Nakamura, S. Tsutsumi, H. Aburatani, and S. Ihara. Multidimensional support vector machines for visualization of gene expression data. *Bioinformatics*, 21:439–444, 2005.

I. Kononenko. Estimating attributes: Analysis and extensions of Relief. In F. Bergadano and L. D. Raedt, editors, *Proceedings of the European Conference on Machine Learning*, 1994.

A. Krogh. Two methods for improving performance of a hmm and their application for gene finding. In T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia, editors, *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 179–186. AAAI Press, Menlo Park, CA, 1997.

A. Krogh, M. Brown, I. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994a.

A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. San Mateo, CA: Morgan Kaufmann, 1992.

A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in E. coli DNA. *Nucl. Acids Res.*, 22:4768–4778, 1994b.

R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of Bioinformatics and Computational Biology*, 3(3):527–550, 2005.

D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden markov model for the recognition of human genes in dna. In D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. F. Smith, editors, *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 134–142. AAAI Press, Menlo Park, CA, 1996.

E.S. Lander, L.M. Linton, and B. Birren. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

P. Langley. Selection of relevant features in machine learning. In *AAAI Fall Symposium on Relevance*, pages 140–144, 1994.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. San Mateo, CA: Morgan Kaufmann, 1990.

C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004a.

C. Leslie, R. Kuang, and E. Eskin. Inexact matching string kernels for protein classification. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, pages 95–111. MIT Press, 2004b.

A. U. Levin, T. K. Leen, and J. E. Moody. Fast pruning using principal components. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 35–42. Morgan Kaufmann, San Mateo, CA, 1994.

L. Liao and W. S. Noble. Combining pairwise squence similarity support vector machines for remote protein homology detection. In *Proceedings of the Sixth International Conference on Computational Molecular Biology*, pages 225–232, 2002.

R.H. Lilien, H. Farid, and B. R. Donald. Probabilistic disease classification of expression-dependent proteomic data from mass spectrometry of human serum. *Computational Biology*, 10(6), 2003.

T. Lin, B. G. Horne, P. Tino, and C. L. Giles. Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

T. Lingner and P. Meinicke. Remote homology detection based on oligomer distances. *Bioinformatics*, 22(18):2224–2236, 2006.

P. Lio, J. L. Thorne, N. Goldman, and D. T. Jones. Passml: Combining evolutionary inference and protein secondary structure prediction. *Bioinformatics*, 14:726–73, 1999.

H. Liu and H. Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Boston, 1998.

H. Liu and R. Setiono. A probabilistic approach to feature selection - a filter solution. In *Proceedings of the 13th International Conference on Machine Learning*, pages 319–327. Morgan Kaufmann, 1996.

B. Logan, P. Moreno, B. Suzek, Z. Weng, and S. Kasif. A study of remote homology detection. Technical Report CRL 2001/05, Cambridge Research Laboratory, 2001. `http://www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-5.html`.

Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Doklady Academy Nauk USSR*, 303:1508–1511, 1988.

D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

D. J. C. MacKay. Bayesian non-linear modelling for the 1993 energy prediction competition. In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Barbara 1993*. Kluwer, Dordrecht, 1993.

T. Marill and D. M. Green. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9:11–17, 1963.

K. Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

M. B. Matthews. Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm. In *Proceedings of the Interantional Neural Networks Conference*, pages 115–119, 1990.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

A. A. Minai and R. D. Williams. Perturbation response in feedforward networks. *Neural Networks*, 7(5):783–796, 1994.

F. Model, P. Adorján, A. Olek, and C. Piepenbrock. Feature selection for DNA methylation based cancer classification. *Bioinformatics*, 17(Suppl 1):S157–S164, 2001.

E. J. Moler, M. L. Chow, and I. S. Mian. Analysis of molecular profile data using generative and discriminative methods. *Physiol Genomics*, 4:109–126, 2000.

M. F. Mller. Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Technical Report PB-432, Computer Science Department, Aarhus University, Denmark, 1993.

J. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 847–854. San Mateo, CA: Morgan Kaufmann, 1992.

E. Moulines, J.-F. Cardoso, and E. Gassiat. Maximum-likelihood for blind separation and deconvolution of noisy signals using mixture models. In *Proceedings of the IEEE International Conference non Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 3617–3620, 1997.

M. C. Mozer. A focused back-propagation algorithm for temporal sequence recognition. *Complex Systems*, 3:349–381, 1989.

M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. San Mateo, CA: Morgan Kaufmann, 1989.

S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical Report AI Memo 1677, Massachusetts Institute of Technology, 1999.

S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. P. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2000.

F. Mulier and V. Cherkassky. Self-organization as an iterative kernel smoothing process. *Neural Computation*, 7(6):1165–1177, 1995.

A. F. Murray and P. J. Edwards. Synaptic weight noise during MLP learning enhances fault-tolerance, generalisation and learning trajectory. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 491–498. San Mateo, CA: Morgan Kaufmann, 1993.

E. Myasnikova, A. Samsonova, M. Samsonova, , and J. Reinitz. Support vector regression applied to the determination of the developmental age of a *Drosophila* embryo from its segmentation gene expression patterns. *Bioinformatics*, 18:S87–S95, 2002.

K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. In *IEEE Transactions on Neural Networks*, volume 1, pages 4–27, 1990.

R. Neal. *Bayesian Learning for Neural Networks*. Springer Verlag, New York, 1996.

R. M. Neal and P. Dayan. Factor analysis using delta-rule wake-sleep learning. *Neural Computation*, 9(8):1781–1803, 1997.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

C. Neti, M. H. Schneider, and E. D. Young. Maximally fault tolerant neural networks. In *IEEE Transactions on Neural Networks*, volume 3, pages 14–23, 1992.

C.G. Nevill-Manning, T.D. Wu, and D.L. Brutlag. Highly specific protein sequence motifs for genome analysis. *Proc. Natl. Acad. Sci. USA*, 95(11), 5865-5871.

S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4:173–193, 1992.

K. Obermayer, G. G. Blasdel, and K. Schulten. A statistical mechanical analysis of self-organization and pattern formation during the development of visual maps. *Physical Review A*, 45:7568–7589, 1992.

E. Oja. A simplified neuron model as principal component analyser. *Journal of Mathematical Biology*, 15:267–273, 1982.

E. Oja. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1):61–68, 1989.

E. E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. Technical Report AIM No. 1602, CBCL no. 144, MIT, 1997.

J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia. Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4):1201–1210, 1998.

P. Pavlidis, T. S. Furey, M. Liberto, and W. N. Grundy. Promoter region-based classification of genes. *Proceedings of the Pacific Symposium on Biocomputing*, pages 151–163, 2001.

B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

B. A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

B. A. Pearlmutter and L. C. Parra. Maximum likelihood blind source separation: A context-sensitive generalization of ICA. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 613–619. MIT Press, Cambridge MA, 1997.

K. Pearson. Contribution to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society A*, 185:71–110, 1894.

W. R. Pearson. Rapid and sensitive sequence comparisions with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003. Special Issue on Variable and Feature Selection.

E. F. Petricoin, A. M. Ardekani, B. A. Hitt, P. J. Levine, V. A. Fusaro, S. M. Steinberg, G. B. Mills, C. Simone, D. A. Fishman, E. C. Kohn, and L. A. Liotta. Use of proteomic patterns in serum to identify ovarian cancer. *The Lancet*, 359(9306):572–577, 2002a.

E. F. Petricoin, D.K. Ornstein, C. P. Paweletz, A. Ardekani, P.S. Hackett, B. A. Hitt, A. Velassco, C.Trucco, L. Wiegand, K. Wood, C. Simone, P. J. Levine, W. M. Linehan, M. R. Emmert-Buck, S. M. Steinberg, E. C. Kohn, and L. A. Liotta. Serum preteomic patterns for detection of prostate cancer. *Journal of the National Cancer Institute*, 94(20):1576–1578, 2002b.

F. J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 19(59):2229–2232, 1987.

S. L. Pomeroy, P. Tamayo, M. Gaasenbeek, L. M. Sturla, M. Angelo, M. E. McLaughlin, J. Y. H. Kim, L. C. Goumnerova, P. M. Black, C. Lau, J. C. Allen, D. Zagzag, J. M. Olson, T. Curran, C. Wetmore, J. A. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. N. Louis, J. P. Mesirov, E. S. Lander, and T. R. Golub. Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415(6870):436–442, 2002.

G. V. Puskorius and L. A. Feldkamp. Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, 1994.

N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural networks. *J. Mol. Biol.*, 202:865–884, 1988.

Y. Qu, B. Adam, Y. Yasui, M. D. Ward, L. H. Cazares, P. F. Schellhammer, Z. Feng, O. J. Semmes, and Jr. G. L. Wright. Boosted decision tree analysis of surfaceenhanced laser desorption/ionization mass spectral serum profiles discriminates prostate cancer from noncancer patients. *Clinical Chemistry*, 48(10):1835–1843, 2002.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

A. Rakotomamonjy. Variable selection using SVM-based criteria. *Journal of Machine Learning Research*, 3:1357–1370, 2003. Special Issue on Variable and Feature Selection.

H. Rangwala and G. Karypis. Profile based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21(23):4239–4247, 2005.

A. N. Refenes, G. Francis, and A. D. Zapranis. Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*, 7(2):375–388, 1994.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps. An Introduction*. Addison-Wesley, Reading, MA, 1992.

H. Ritter, K. Obermayer, K. Schulten, and J. Rubner. Self-organizing maps and adaptive filters. In E. Domani, J. L. van Hemmen, and K. Schulten, editors, *Physics of Neural Networks*, pages 281–306. Springer-Verlag, New York, 1991.

A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

M. Robnik-Sikonja and I. Kononenko. An adaptation of Relief for attribute estimation in regression. In *Proceedings of the 14th International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann, 1997.

B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *J. Mol. Biol.*, 232:584–599, 1993.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986a.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986b.

J. Schmidhuber. A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992a.

J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4 (6):863–879, 1992b.

B. Schölkopf, S. Mika, A. J. Smola, G. Rätsch, and K.-R. Müller. Kernel PCA pattern reconstruction via approximate pre-images. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, Perspectives in Neural Computing, pages 147–152, Berlin, 1998. Springer Verlag.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

B. Schölkopf and A. J. Smola. *Learning with kernels – Support Vector Machines, Reglarization, Optimization, and Beyond*. MIT Press, Cambridge, 2002.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

J. Schuchhardt and D. Beule. Normalization strategies for cDNA microarrays. *Nucleic Acids Research*, 28(10):e47, 2000.

J. Schultz, R. R. Copley, T. Doerks, C. P. Ponting, and P. Bork. A web-based tool for the study of genetically mobile domains. *Nucleic Acids Res.*, 28:231–23, 2000.

N. H. Segal, P. Pavlidis, C. R. Antonescu, R. G. Maki, W. S. Noble, J. M. Woodruff, J. J. Lewis, M. F. Brennan, A. N. Houghton, and C. Cordon-Cardo. Classification and subtype prediction of soft tissue sarcoma by functional genomics and support vector machine analysis. *American Journal of Pathology*, 2003. To appear.

L. Shen and E. C. Tan. Reducing multiclass cancer classification to binary by output coding and SVM. *Computational Biology and Chemistry*, 30(1):63–71, 2006.

M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, R. C. T. Aguiar J. L. Kutok, M. Gaasenbeek, M. Angelo, M. Reich, T. S. Ray G. S. Pinkus, M. A. Koval, K. W. Last, A. Norton, J. Mesirov T. A. Lister, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, 8(1):68–74, 2002.

W. Siedlecki and J. Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197–220, 1988.

H.T. Siegelmann. Computation beyond the turing limit. *Science*, 268:545–548, 1995.

H.T. Siegelmann and E.D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler. Dirichlet mixtures: a method for improved detection of weak but significant protein sequence homology. *CABIOS*, 12(4):327–345, 1996.

D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *International Conference on Machine Learning*, pages 293–301, 1994.

T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

A. J. Smola, T. Frieß, and B. Schölkopf. Semiparametric support vector and linear programming machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 585–591, Cambridge, MA, 1999. MIT Press.

A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 2002. Forthcoming. Also: NeuroCOLT Technical Report NC-TR-98-030.

E. L. L. Sonnhammer, S.R. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins*, 28:405–420, 1997.

E.L. Sonnhammer, S.R. Eddy, and E. Birney. PFAM: Multiple sequence alignments and HMM-profiles of protein domains. *Nucleic Acids Research*, 26(1):320–322, 1998.

E. Southern. United Kingdom patent application GB8810400, 1988.

E. J. Spinosa and A. C. P. L. F. de Carvalho. Support vector machines for novel class detection in bioinformatics. *Genetics and Molecular Research*, 4(3):608–615, 2005.

M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks*, pages I–607–I–611, Washington D.C., 1989. IEEE TAB Neural Network Committee.

C. J. Stone. Optimal rates of convergence for nonparametric estimators. *Annals of Statistics*, 8(6): 1348–1360, 1980.

Y. Su, T. M. Mural, V. Pavlovic, M. Schaffer, and S. Kasif. RankGene: Identification of diagnostic genes based on expression data. *Bioinformatics*, 2003. To appear.

G.Z. Sun, H.H. Chen, Y.C. Lee, and C.L. Giles. Turing equivalence of neural networks with second order connection weights. In *IEEE INNS International Joint Conference on Neural Networks*, volume II, pages 357–362, Piscataway, NJ, 1991. IEEE Press.

E. K. Tang, P.N. Suganthan, and X. Yao. Gene selection algorithms for microarray data based on least squares support vector machine. *BMC Bioinformatics*, 7:95, 2006.

R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu. Class prediction by nearest shrunken centroids, with applications to DNA microarrays. *Statistical Science*, 18(1):104–117, 2003.

R. J. Tibshirani and B. Efron. Pre-validation and inference in microarrays. *Statistical Applications in Genetics and Molecular Biology*, 1(1):1–18, 2002.

G. Tseng, M. Oh, L. Rohlin, J. Liao, and W. Wong. Issues in cDNA microarray analysis: quality filtering, channel normalization, models of variations and assessment of gene effects. *Nucleic Acids Research*, 29:2549–2557, 2001.

P. D. Turney. Robust classification with context-sensitive features. In *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 268–276, 1993a. ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-35074.ps.Z.

P. D. Turney. Robust classification with context-sensitive features. In *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 268–276, 1993b. ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-35074.ps.Z.

V. G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Science*, 98:5116–5121, 2001.

H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learing. In *Proceedings of the Fourth Conference on Tools for Artificial Intelligence*, pages 200–203. IEEE Computer Society Press, 1992.

H. Vafaie and K. De Jong. Robust feature selection algorithms. In *Proceedings of the Fifth Conference on Tools for Artificial Intelligence*, pages 356–363. IEEE Computer Society Press, 1993.

L. J. van't Veer, H. Dai, M. J. van de Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, 2002.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995. ISBN 0-387-94559-8.

J.C. Venter, M.D. Adams, E.W. Myers, and P.W. Li. The sequence of the human genome. *Science*, 2901(16):1304–1351, 2001.

J.-P. Vert. Support vector machine prediction of signal peptide cleavage site using a new class of kernels for strings. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Kevin Lauerdale, and Teri E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 649–660. World Scientific, 2002a.

J.-P. Vert. A tree kernel to analyze phylogenetic profiles. In *Proceedings of ISMB'02*, 2002b.

J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 1425–1432. The MIT Press, 2003.

J.-P. Vert, H. Saigo, and T. Akutsu. Local alignment kernels for biological sequences. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, pages 131–154. MIT Press, 2004.

Jean-Phlippe Vert. A tree kernel to analyze phylogenetic profiles. *Bioinformatics*, 18:S276–S284, 2002c.

R. Vollgraf, M. Scholz, I. Meinertzhagen, and K. Obermayer. Nonlinear filtering of electron micrographs by means of support vector regression. In *Advances in Neural Information Processing Systems 16*, pages 717–724. MIT Press, Cambridge, Massachusetts, 2004.

C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Jrnl.*, 11 (2):185–194, 1968.

E. A. Wan. Temporal backpropagation for FIR neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 575–580, 1990.

D. G. Wang, J.-B. Fan, and C.-J. Siao. Large-scale identification, mapping, and genotyping of single-nucleotide polymorphisms in the human genome. *Science*, 280:1077–1082, 1998.

X. Wang, A. Li, Z. Jiang, and H. Feng. Missing value estimation for DNA microarray gene expression data by support vector regression imputation and orthogonal coding scheme. *BMC Bioinformatics*, 7:32, 2006.

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. San Mateo, CA: Morgan Kaufmann, 1991.

P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kozin, editors, *System Modeling and Optimization: Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770. Springer-Verlag, 1981. number 38 in Lecture Notes in Control and Information Sciences.

P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.

P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461, 2003. Special Issue on Variable and Feature Selection.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, Cambridge, MA, 2000.

H. White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1 (4):425–464, 1989.

H. White. Connectionist nonparametric regression: Multilayer perceptrons can learn arbitrary mappings. *Neural Networks*, 3(535-549), 1990.

P. M. Williams. Bayesian regularisation and pruning using a Laplace prior. Technical report, School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton, 1994.

R. J. Williams. Training recurrent networks using the extended Kalman filter. In *Proceedings of the International Joint Conference on Neural Networks IJCNN-92*, pages 241–250. Lawrence Erlbaum, Hillsdale, N.J., 1992.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum, 1992.

B. Wu, T. Abbott, D. Fishman, W. McMurray, G. Mor, K. Stone, D. Ward, K. Williams, and H. Zhao. Comparison of statistical methods for classifcation of ovarian cancer using mass spectrometry data. *Bioinformatics*, 19(13), 2003.

J. D. Wulfkuhle, L. A. Liotta, and E. F. Petricoin. Proteomic applications for the early detection of cancer. *Nature Reviews*, 3:267–275, 2003.

L. Xu, P. Zan, and T. Chang. Best first strategy for feature selection. In *Ninth International Conference on Pattern Recognition*, pages 706–708. IEEE Computer Society Press, 1989.

G. Yang and S. Amari. Adaptive online learning algorithms for blind source separation: Maximum entropy and minimum mutual information. *Neural Computation*, 9(7):1457–1482, 1997.

H. H. Yang, S. Amari, and A. Cichocki. Information back-propagation for blind separation of sources from non-linear mixtures. In *Proceedings of the International Conference on Neural Networks*, pages 2141–2146. Houston, USA, 1996.

N. Zavaljevski and J. Reifman. Support vector machines with selective kernel scaling for protein classification and identification of key amino acid positions. *Bioinformatics*, 18(5):698–696, 2002.

H. H. Zhang, J. Ahn, X. Lin, and C. Park. Gene selection using support vector machines with non-convex penalty. *Bioinformatics*, 22:88–95, 2006.

Y. Zhao and C. G. Atkeson. Implementing projection pursuit learning. *IEEE Transactions on Neural Networks*, 7(2):362–373, 1996.

A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.