Lecture Notes
Institute of Bioinformatics, Johannes Kepler University Linz

Machine Learning: Supervised Techniques

Winter Semester 2014

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
- C. M. Bishop; Pattern Recognition, Oxford University Press, 2008
- Schölkopf, Smola; Learning with kernels, MIT Press, 2002
- 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

1.1 Machine Learning Introduction

This course is part of the curriculum of the master in computer science (in particular the majors "Computational Engineering" and "Intelligent Information Systems") and part of the master in bioinformatics at the Johannes Kepler University Linz.

Machine learning is currently a major research topic at companies like Google, Microsoft, Amazon, Facebook, AltaVista, Zalando, and many more. Applications are found in computer vision (image recognition), speech recognition, recommender systems, analysis of Big Data, information retrieval. Companies that try to mine the world wide web are offering search engines, social networks, videos, music, information, or connecting people use machine learning techniques. Machine learning methods are used to classify and label web pages, images, videos, and sound recordings in web data. They can find specific objects in images and detect a particular music style if only given the raw data. Therefore Google, Microsoft, Facebook are highly interested in machine learning methods. Machine learning methods attracted the interest of companies offering products via the web. These methods are able to identify groups of similar users, to predict future behavior of customers, and can give recommendation of products in which customers will be interested based previous costumer data.

Machine learning has major applications in biology and medicine. 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 and sequencing techniques. The measurement data are first preprocessed, then genes of interest are identified, and finally predictions made. Further machine learning methods are used to detect alternative splicing, nucleosome positions, gene regulation, etc. Alongside neural networks the most prominent machine learning techniques relate to support vector machines, kernel approaches, projection method and probabilistic models like latent variable models. These methods provide noise reduction, feature selection, structure extraction, classification / regression, and assist modeling. In the biomedical context, machine learning algorithms categorize the disease subtype or predict treatment outcomes based on DNA characteristics, gene expression profiles. Machine learning approaches classify novel protein sequences into structural or functional classes. For analyzing data of association studies, machine learning methods extract new dependencies between DNA markers (SNP - single nucleotide polymorphisms, SNV - single nucleotide variants, CNV - copy number variations) and diseases (Alzheimer, Parkinson, cancer, multiples sclerosis, schizophrenia or alcohol dependence).

The machine learning course series comprises:

- "Basic Methods of Data Analysis": this course gives a smooth introduction to machine learning with examples in R ; it covers summary statistics (mean, variance), data summary plots (boxplot, violin plot, scatter plot), principal component analysis, independent component analysis, multidimensional scaling (Kruskal's or Sammon's map), locally linear embedding, Isomap, hierarchical clustering, mixture models, k-means, similarity based clustering (affinity propagation), biclustering

- "Machine Learning: Supervised Methods": classification and regression techniques, time series prediction, kernel methods, support vector machines, neural networks, deep learning, deep neural and belief networks, ARMA and ARIMA models, recurrent neural networks, LSTM

- "Machine Learning: Unsupervised Methods": maximum likelihood estimation, maximum a posterior estimation, maximum entropy, expectation maximization, principal component analysis, statistical independence, independent component analysis, factor analysis, mixture models, sparse codes, population codes, kernel PCA, hidden Markov models (factorial HMMs and input-output HMMs), Markov networks and random fields, clustering, biclustering, restricted Boltzmann machines, auto-associators, unsupervised deep neural networks

- "Theoretical Concepts of Machine Learning": estimation theory (unbiased and efficient estimator, Cramer-Rao lower bound, Fisher information matrix), consistent estimator, complexity of model classes (VC-dimension, growth, annealed entropy), bounds on the generalization error, Vapnik and worst case bounds on the generalization error, optimization (gradient based methods and convex optimization), Bayes theory (posterior estimation, error bounds, hyperparameter optimization, evidence framework), theory on linear functions (statistical tests, intervals, ANOVA, generalized linear functions, mixed models)

In this course the most prominent machine learning techniques are introduced and their mathematical basis and derivatives are explained. If the student understands these techniques, then the student can select the methods which best fit to the problem at hand, the student is able to optimize the parameter settings for the methods, the student can adapt and improve the machine learning methods, and the student can develop new machine learning methods.

Most importantly, students should learn how to chose appropriate methods from a given pool of approaches for solving a specific problem. To this end, 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 particular applications with certain prior knowledge or problem-specific constraints.

1.2 Course Specific Introduction

This course introduces supervised machine learning methods. The first kind of methods are *classification techniques*. An object or state is represented by measurements or features and must be classified into given categories like diseased vs. healthy or relevant vs. irrelevant for binary classifications or content types of documents, category of web page users, type of object in an image (traffic signs), letters and digits in handwriting, words in speech recognition, or preferences of the

user in recommender systems. The second kind of methods are *regression techniques*. Again an object or state is represented by measurements or features and another characteristic of the object or state of the system must be estimated. In contrast to classification, regression methods predict real values. The third kind of methods are *time series prediction techniques* which are closely related to regression methods. In time series prediction current and previous measurements or system characteristics are used to predict future characteristics or system states.

Methods are kernel-based methods like support vector machines (SVMs) and neural networks. The latter have a renaissance by deep learning in which companies like Facebook or Google are highly interested as they won several object recognition and speech recognition challenges. For time series prediction recurrent neural networks (cf. LSTM) as well as auto-regressive moving-average (ARMA) and ARIMA models are considered.

For supervised machine learning methods the data is in form of a feature vector which describes an object or relation of an object to other objects and a label or target. Based on the feature vector or relations the label or target must be predicted. To learn these predictions, a model is selected on a training set of such kind of data. The model is expected to make correct predictions for future or new data. For supervised methods an error, that is how good was the prediction, can be assigned to every training object. Therefore the training error can be broken down to single objects which is not possible in most unsupervised methods. "Supervised" indicates that for each training object a label or target is available: a teacher immediately tells what would be correct. An immediate feedback of the quality of the model prediction is available.

Supervised methods are used for performing prediction of future data. In contrast, unsupervised methods allow to explore the data, find structure in the data, visualize the data, compress the data. Hence, supervised methods should perform well on new or unseen data while unsupervised methods help to understand and explore the data and generate new knowledge.

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 to 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. Predictions support the current action through the knowledge of the future. A prominent example is stock market prediction but also prediction of 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) detection
- Peptide and protein array analysis
- 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 "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).

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.

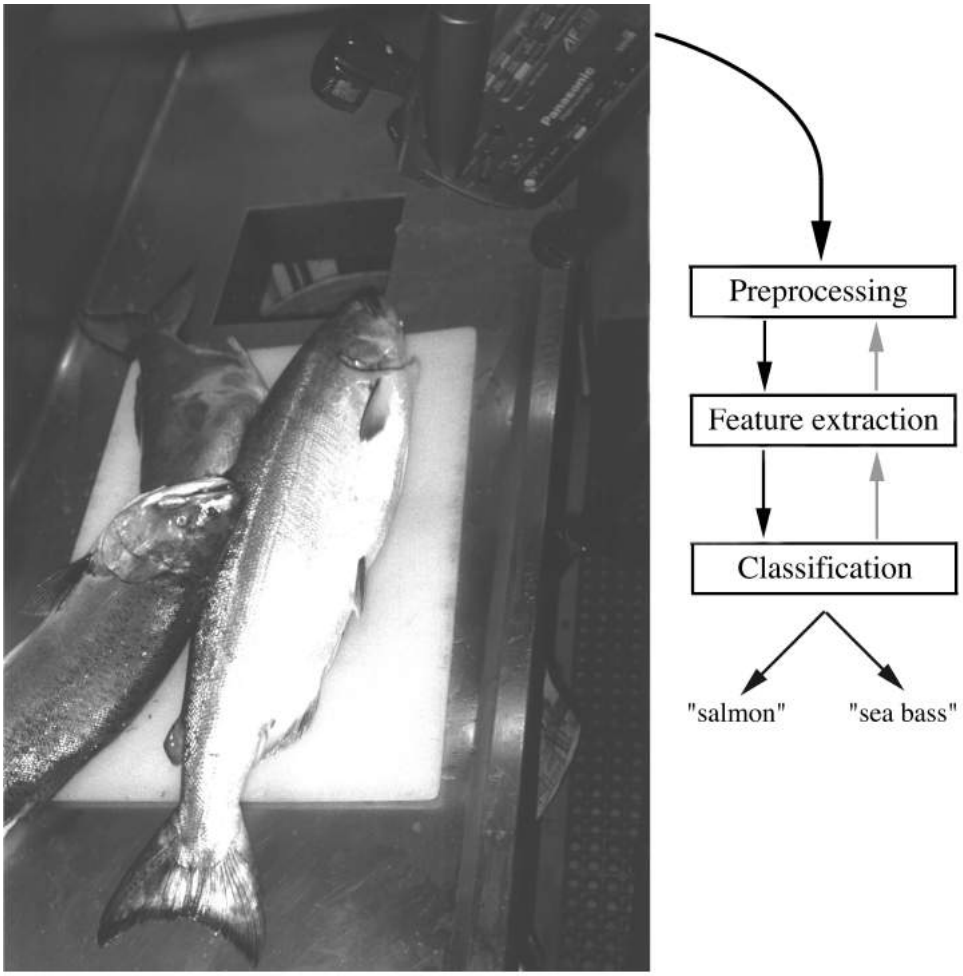


Figure 2.1: Salmons must be distinguished from sea bass. A camera takes pictures of the fishes and these pictures have to be classified as showing either a salmon or a sea bass. The pictures must be preprocessed and features extracted whereafter classification can be performed. Copyright © 2001 John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

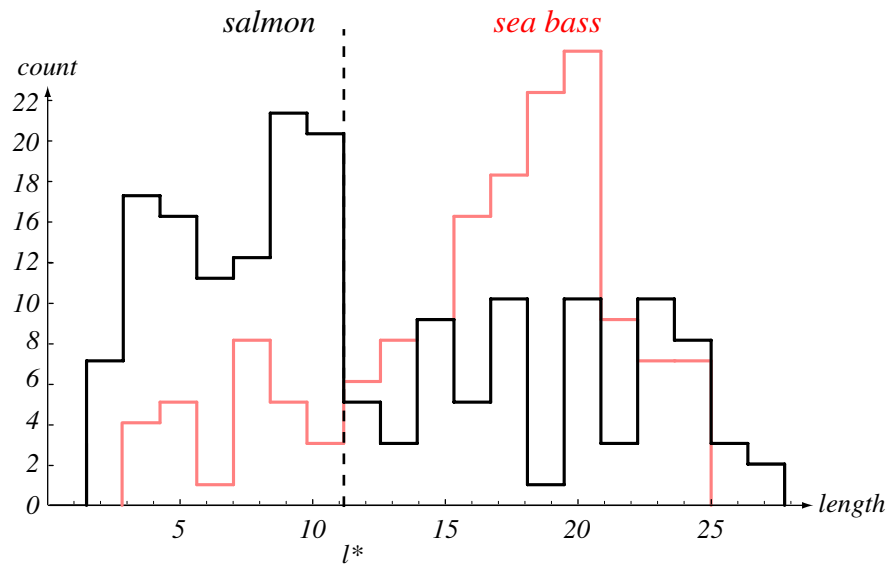


Figure 2.2: Salmon and sea bass are separated by their length. Each vertical line gives a decision boundary l, where fish with length smaller than l are assumed to be salmon and others as sea bass. l^* gives the vertical line which will lead to the minimal number of misclassifications. Copyright © 2001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

Before the classification can be done the pictures must be preprocessed and features extracted. Classification is performed with 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 a 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.

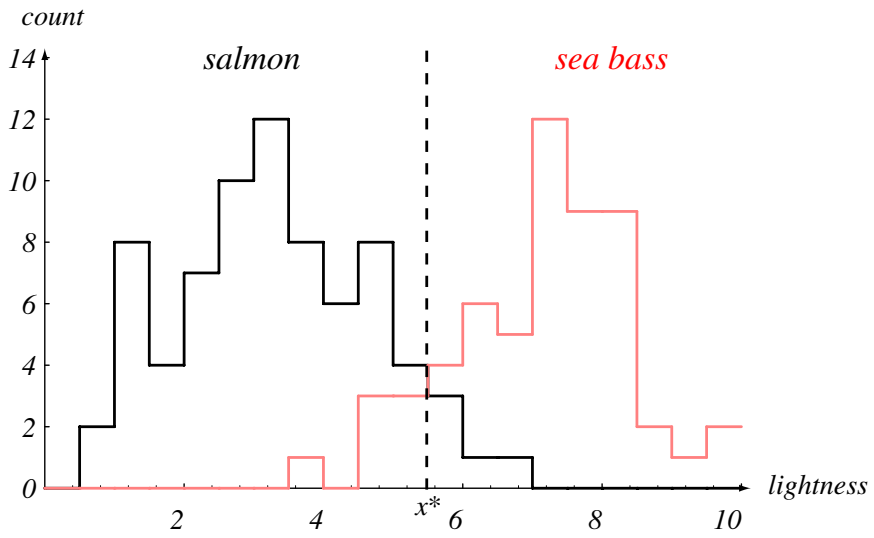


Figure 2.3: Salmon and sea bass are separated by their lightness. x^* gives the vertical line which will lead to the minimal number of misclassifications. Copyright © 2001 John Wiley & Sons, Inc.

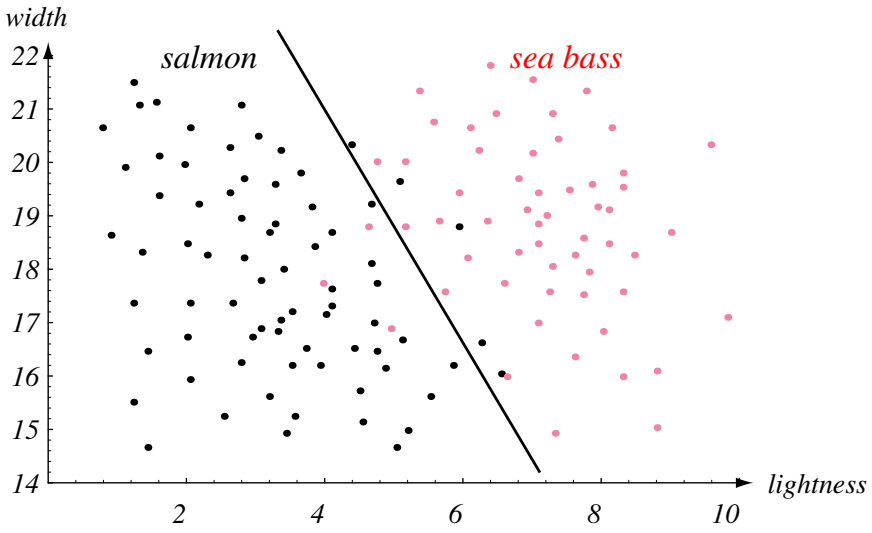


Figure 2.4: Salmon and sea bass are separated by their lightness and their width. For each width there is an optimal separating lightness value given by the line. Here the optimal lightness is a linear function of the width. Copyright © 2001 John Wiley & Sons, Inc.

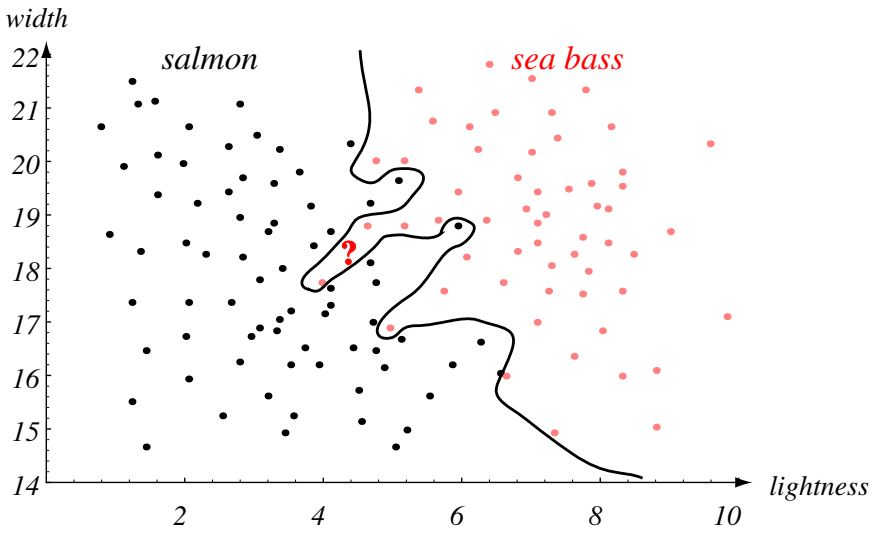


Figure 2.5: Salmon and sea bass are separated by a nonlinear curve in the two-dimensional space spanned by the lightness and the width of the fishes. The training set is separated perfectly. A new fish with lightness and width given at the position of the question mark "?" would be assumed to be sea 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

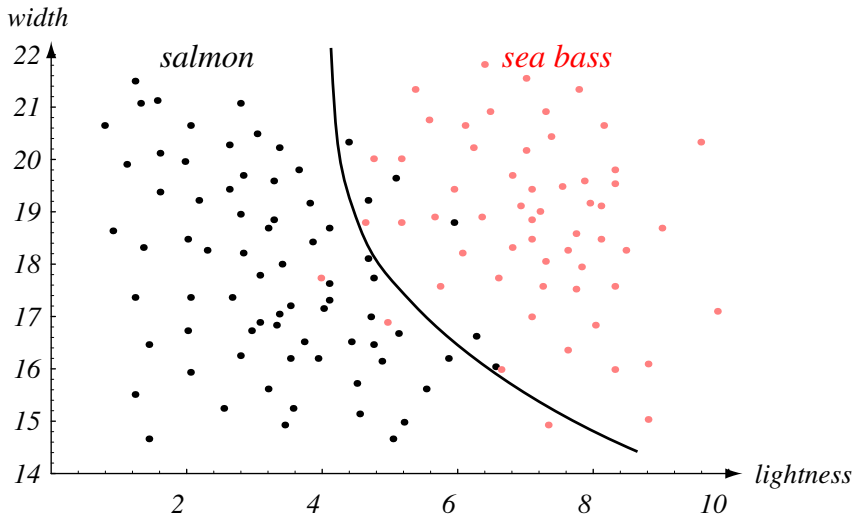


Figure 2.6: Salmon and sea bass are separated by a nonlinear curve in the two-dimensional space spanned by the lightness and the width of the fishes. The curve may represent the decision boundary leading to the best generalization. Copyright © 2001 John Wiley & Sons, Inc.

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 generating new features and thereafter selecting 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 the 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 a supervised task one can immediately measure how good the model performs on the training data, because the optimal outputs, the tar-

gets, are given. Further the measurement is done for each single object. This means that 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.

2.4 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* Hochreiter and Schmidhuber [1997a,d, 1999c,a,b], Hochreiter and Mozer [2000, 2001a,d].

In our example features were extracted from images. Another example is given in Fig. 2.7 and Fig. 2.8 where brain patterns have to be extracted from fMRI brain images. In these figures also temporal patterns are given as EEG measurements from which 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 tasks for which a huge number of features is available. In the bioinformatics context 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 predicting the outcome of a therapy) and all other genes are not. An example is given in Fig. 2.9, where one variable is related to the classification task and the other is not.

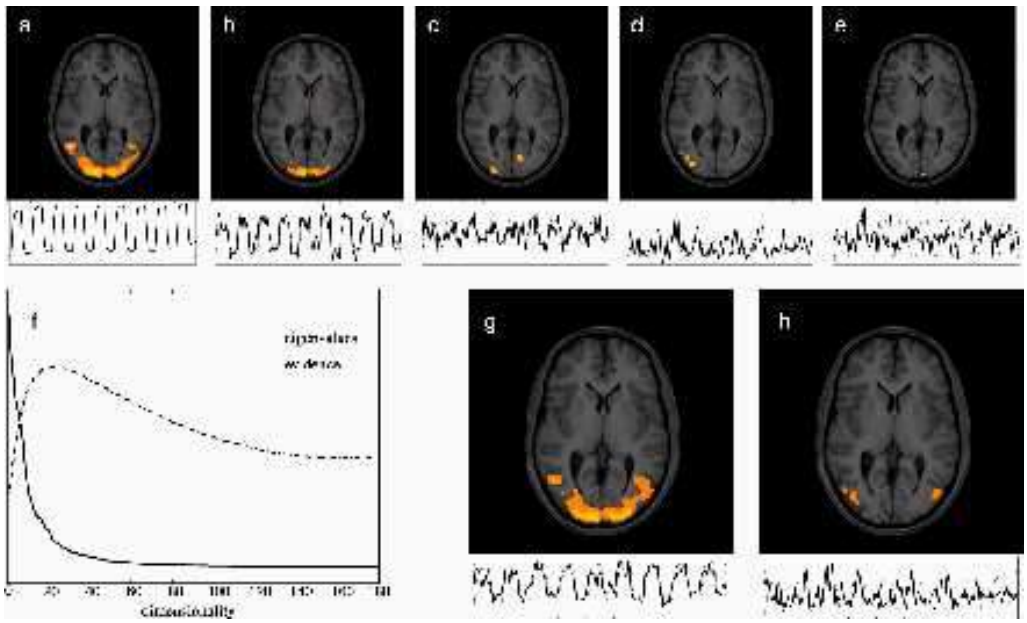


Figure 2.7: Images of fMRI brain data together with EEG data. Certain active brain regions are marked.

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. 2.10 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. 2.11 shows an example where single features cannot improve the classification performance but both features simultaneously help to classify correctly. Fig. 2.12 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 selected because it is correlated to all other features.

In some tasks it is helpful to combine some features to a new feature, 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

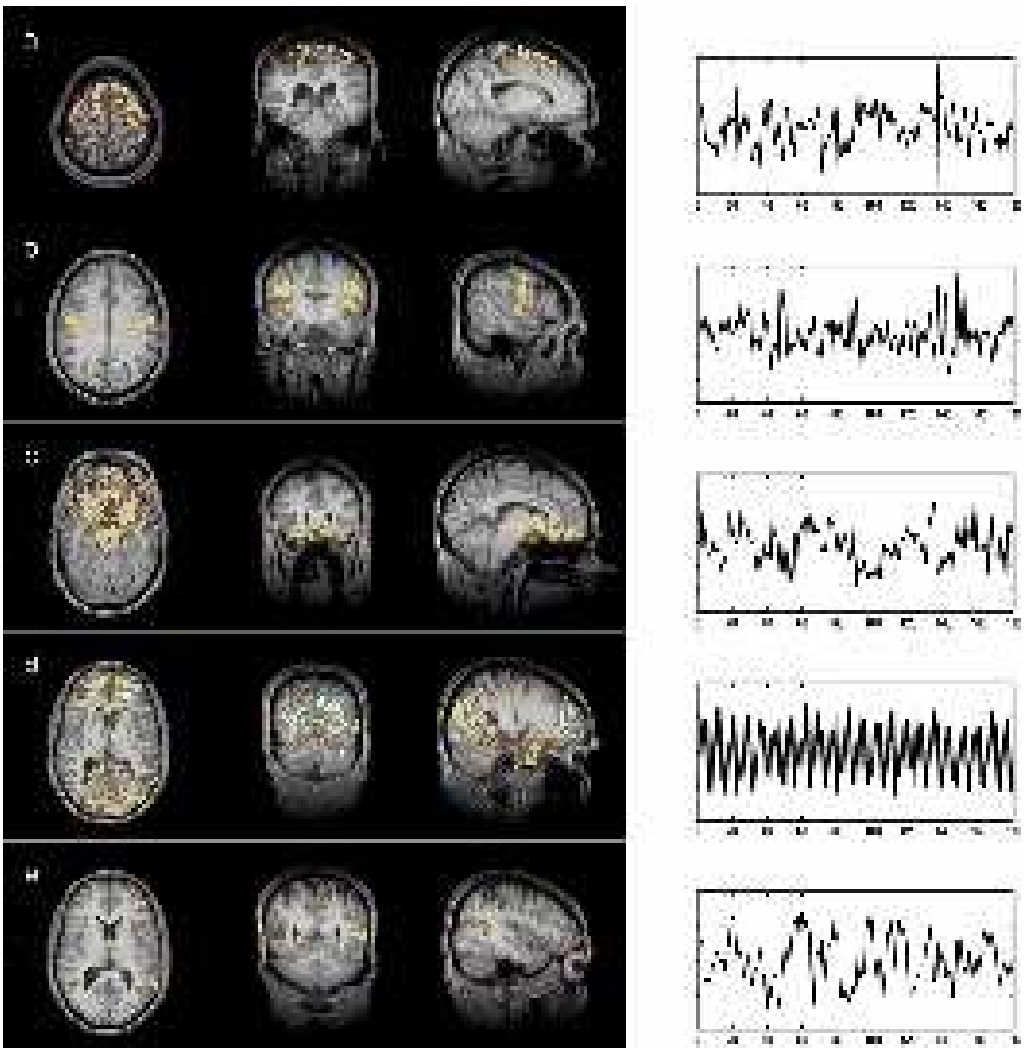


Figure 2.8: Another image of fMRI brain data together with EEG data. Again, active brain regions are marked.

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.

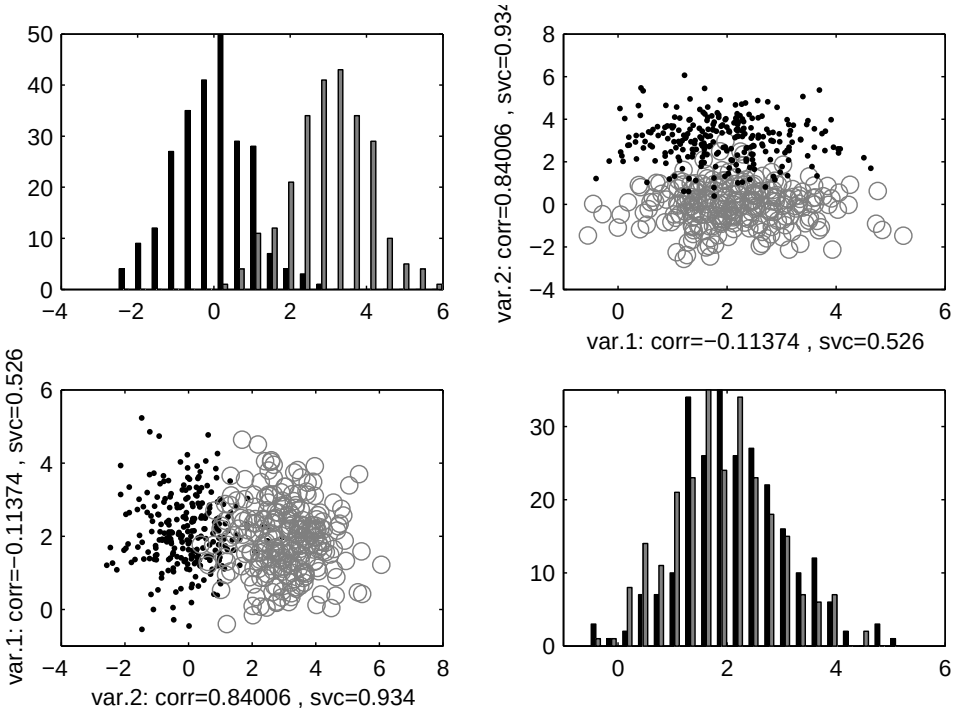


Figure 2.9: Simple two feature classification problem, where feature 1 (var. 1) is noise and feature 2 (var. 2) is correlated to the classes. In the upper right figure and lower left figure only the axis are exchanged. The upper left figure gives the class histogram along feature 2 whereas the lower 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.

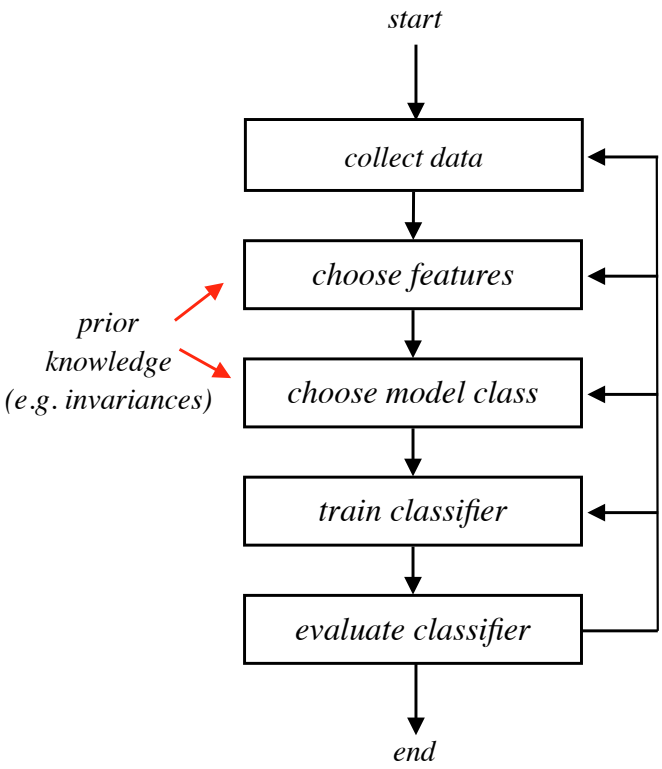


Figure 2.10: The design cycle for machine learning in order to solve a certain task. Copyright © 2001 John Wiley & Sons, Inc.

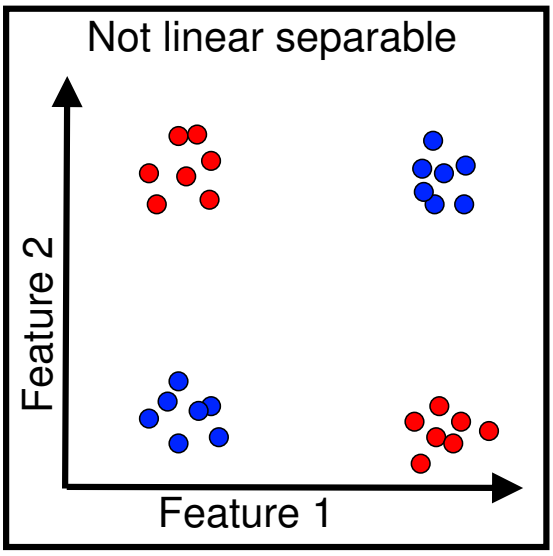


Figure 2.11: An XOR problem of two features, where each single feature is neither correlated to the problem nor helpful for classification. Only both features together help.

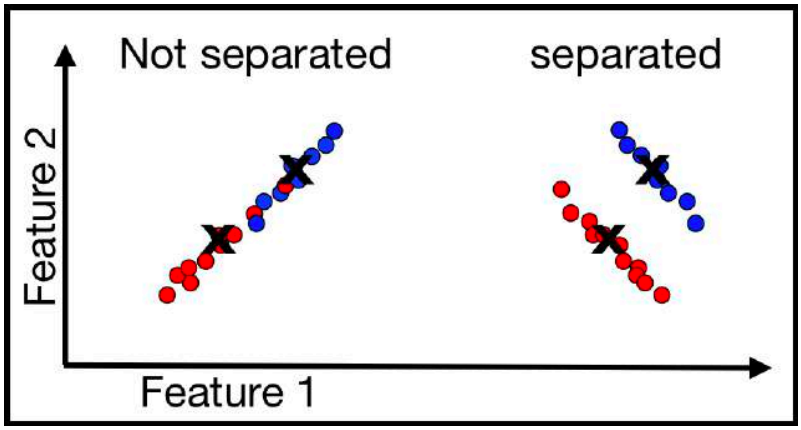


Figure 2.12: The left and right subfigure shows each two classes where the features mean value and variance for each class is equal. However, the direction of the variance differs in the subfigures leading to different performance in classification.

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.

2.5 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. 2.10. 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 has 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 are 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 partitioning the space.

2.6 Prior and Domain Knowledge

In the 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.

2.7 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 be 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 represent this structure. This case is called *"underfitting"*. Thus, the optimal generalization is a trade-off between underfitting and overfitting. See Fig. 2.13 for the trade-off between over- and underfitting error.

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" 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.

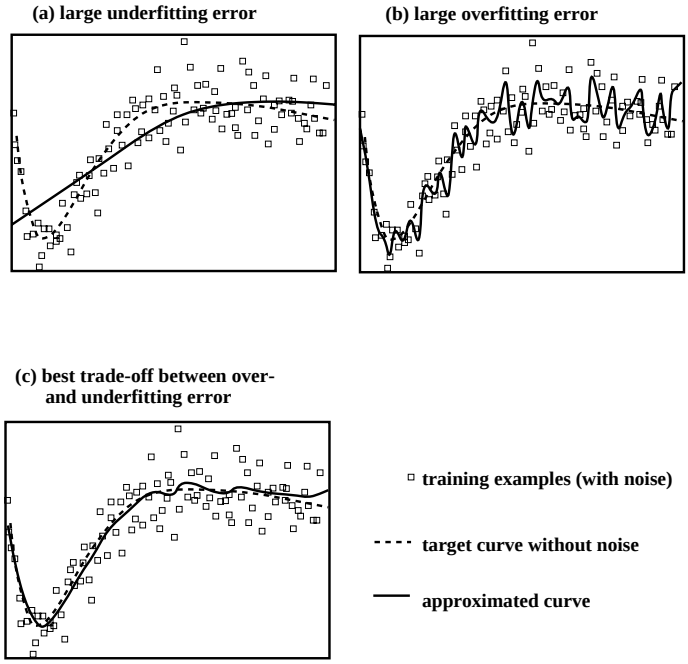


Figure 2.13: The trade-off between underfitting and overfitting is shown. The left upper subfigure shown underfitting, the right upper subfigure overfitting error, and the right lower subfigure shows the best compromise between both leading to the highest generalization (best performance on future data).

2.8 Model Evaluation, Hyperparameter Selection, and Final Model

In the 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* (see more and theoretical detailed treatment in Section 2.9.2.2). 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 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 is 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 known 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.

For comparing different methods it is important to test whether the observed differences in the performances are significant or not. The tests for assessing 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.

2.9 Generalization Error / Risk

2.9.1 Definition of the Generalization Error

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 which 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

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = \begin{cases} 0 & \text{for } y = g(\boldsymbol{x}; \boldsymbol{w}) \\ 1 & \text{for } y \neq g(\boldsymbol{x}; \boldsymbol{w}) \end{cases} . \tag{2.1}$$

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):

$$R(g(.; \boldsymbol{w})) = \mathrm{E}_{\boldsymbol{z}} \left(L(y, g(\boldsymbol{x}; \boldsymbol{w}))\right) . \tag{2.2}$$

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

$$R(g(.; \boldsymbol{w})) = \int_Z L(y, g(\boldsymbol{x}; \boldsymbol{w}))\, p(\boldsymbol{z})\, d\boldsymbol{z} . \tag{2.3}$$

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

$$y = f(\boldsymbol{x}) + \epsilon , \tag{2.4}$$

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

$$p(y \mid \boldsymbol{x}) = p_n(y - f(\boldsymbol{x})) . \tag{2.5}$$

Here the probabilities can be rewritten as

$$p\left(\boldsymbol{z}\right) \;=\; p\left(\boldsymbol{x}\right)\; p(y \mid \boldsymbol{x}) \;=\; p\left(\boldsymbol{x}\right)\; p_n(y \;-\; f(\boldsymbol{x}))\;. \tag{2.6}$$

Now the risk can be computed as

$$\begin{aligned} R(g(.;\boldsymbol{w})) \;=\; \int_Z L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p\left(\boldsymbol{x}\right)\; p_n(y \;-\; f(\boldsymbol{x}))\, d\boldsymbol{z} \;= \\ \int_X p\left(\boldsymbol{x}\right) \int_{\mathbb{R}} L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p_n(y \;-\; f(\boldsymbol{x}))\, dy\, d\boldsymbol{x}\;, \end{aligned} \tag{2.7}$$

where

$$\begin{aligned} R(g(\boldsymbol{x};\boldsymbol{w})) \;=\; \mathrm{E}_{y|\boldsymbol{x}}\left(L(y, g(\boldsymbol{x};\boldsymbol{w}))\right) \;= \\ \int_{\mathbb{R}} L(y, g(\boldsymbol{x};\boldsymbol{w}))\, p_n(y \;-\; f(\boldsymbol{x}))\, dy\;. \end{aligned} \tag{2.8}$$

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

$$\int_{\mathbb{R}} h(\boldsymbol{x})\delta(\boldsymbol{x})d\boldsymbol{x} \;=\; h(\mathbf{0}) \tag{2.9}$$

therefore

$$R(g(\boldsymbol{x};\boldsymbol{w})) \;=\; L(f(\boldsymbol{x}), g(\boldsymbol{x};\boldsymbol{w})) \;=\; L(y, g(\boldsymbol{x};\boldsymbol{w})) \tag{2.10}$$

and eq. (2.3) simplifies to

$$R(g(.;\boldsymbol{w})) \;=\; \int_X p\left(\boldsymbol{x}\right) L(f(\boldsymbol{x}), g(\boldsymbol{x};\boldsymbol{w}))d\boldsymbol{x}\;. \tag{2.11}$$

Because we do not know $p\left(\boldsymbol{z}\right)$ 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 $\boldsymbol{w} \;=\; \boldsymbol{w}\left(\boldsymbol{Z}\right)$, i.e. the parameters depend on the training set.

2.9.2 Empirical Estimation of the Generalization Error

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

2.9.2.1 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 $E_{\boldsymbol{z}}\left(|L(y, g(\boldsymbol{x};\boldsymbol{w}))|\right) \;<\; \infty$.

The risk is an expectation of the loss function:

$$R(g(.;\boldsymbol{w})) \;=\; \mathrm{E}_{\boldsymbol{z}}\left(L(y, g(\boldsymbol{x};\boldsymbol{w}))\right)\;, \tag{2.12}$$

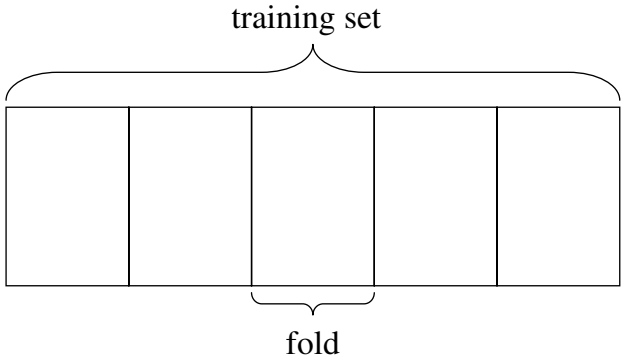


Figure 2.14: Cross-validation: The data set is divided into 5 parts for 5-fold cross-validation — each part is called fold.

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

$$R(g(.;\boldsymbol{w})) \;\approx\; \frac{1}{m}\sum_{i=l+1}^{l+m} L\left(y^i, g(\boldsymbol{x}^i;\boldsymbol{w})\right) \;, \tag{2.13}$$

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.

2.9.2.2 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 times 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. 2.14). 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. 2.15). 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 over training sets with l elements.

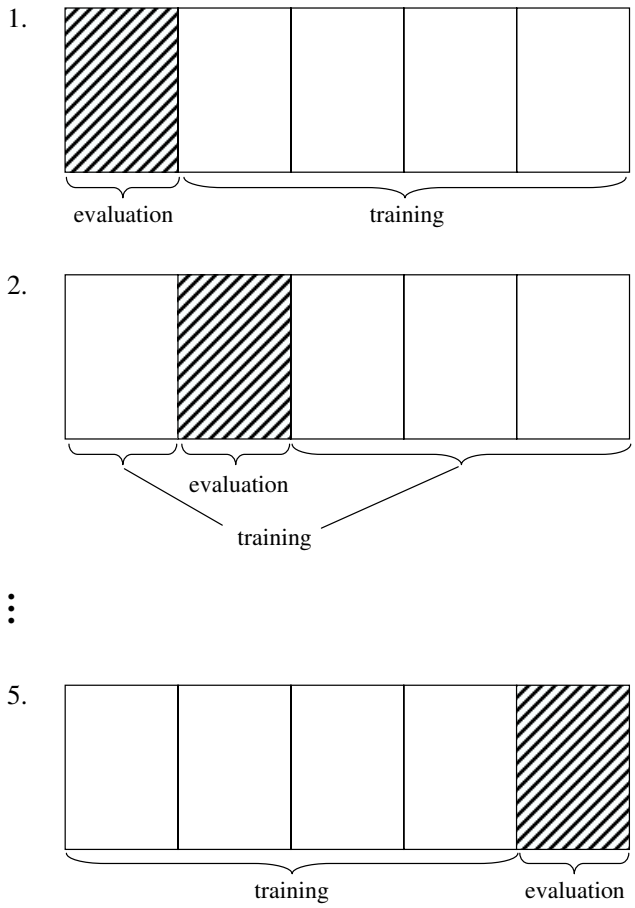


Figure 2.15: Cross-validation: For 5-fold cross-validation there are 5 iterations and in each iteration a different fold is omitted and the remaining folds form the training set. After training the model is tested on the omitted fold. The cumulative error on all folds is the cross-validation error.

We will write $\boldsymbol{Z}_l := \boldsymbol{Z}$ as a variable for training sets with l elements. The j fold of an 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 n-fold cross-validation risk is

$$R_{n-\mathrm{cv}}(\boldsymbol{Z}_l) \;=\; \frac{1}{n}\sum_{j=1}^{n}\frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}^j_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}_j\left(\boldsymbol{Z}_l \setminus \boldsymbol{Z}^j_{l/n}\right)\right)\right)\right)\;, \tag{2.14}$$

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

$$R_{n-\mathrm{cv},j}(\boldsymbol{Z}_l) \;=\; \frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}^j_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}_j\left(\boldsymbol{Z}_l \setminus \boldsymbol{Z}^j_{l/n}\right)\right)\right)\right) \tag{2.15}$$

is the risk for the jth fold.

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

$$\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\left(R\left(g\left(.; \boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;=\; \mathrm{E}_{\boldsymbol{Z}_l}\left(R_{n-\mathrm{cv}}\left(\boldsymbol{Z}_l\right)\right)\;. \tag{2.16}$$

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. (2.16).

The left hand side of eq. (2.16) can be rewritten as

$$\begin{aligned}
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\left(R\left(g\left(.; \boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}\cup\boldsymbol{z}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\mathrm{E}_{\boldsymbol{Z}_{l/n}}\left(\frac{n}{l}\sum_{\boldsymbol{z}\in\boldsymbol{Z}_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right)\right)\;.
\end{aligned} \tag{2.17}$$

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. (2.16) can be rewritten as

$$\begin{aligned}
&\mathrm{E}_{\boldsymbol{Z}_l}\left(R_{n-\mathrm{cv}}\left(\boldsymbol{Z}_l\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_l}\left(\frac{1}{n}\sum_{j=1}^{n}\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}^j_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}_j\left(\boldsymbol{Z}_l \setminus \boldsymbol{Z}^j_{l/n}\right)\right)\right)\right)\right) \;= \\
&\frac{1}{n}\sum_{j=1}^{n}\mathrm{E}_{\boldsymbol{Z}_l}\left(\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}^j_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}_j\left(\boldsymbol{Z}_l \setminus \boldsymbol{Z}^j_{l/n}\right)\right)\right)\right)\right) \;= \\
&\mathrm{E}_{\boldsymbol{Z}_{l(1-1/n)}}\mathrm{E}_{\boldsymbol{Z}_{l/n}}\left(\frac{n}{l}\sum_{(\boldsymbol{x},y)\in\boldsymbol{Z}_{l/n}}\left(L\left(y, g\left(\boldsymbol{x}; \boldsymbol{w}\left(\boldsymbol{Z}_{l(1-1/n)}\right)\right)\right)\right)\right)\;.
\end{aligned} \tag{2.18}$$

The first equality comes from the fact that sum and integral are 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. (2.16) 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.

2.10 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

$$p(\boldsymbol{x} \mid y = 1) \propto \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \tag{2.19}$$

and Class $y = -1$ according to

$$p(\boldsymbol{x} \mid y = -1) \propto \mathcal{N}(\boldsymbol{\mu}_{-1}, \boldsymbol{\Sigma}_{-1}) \tag{2.20}$$

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

$$p(\boldsymbol{x}) = \frac{1}{(2\,\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu})^T\,\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) . \tag{2.21}$$

$\boldsymbol{x}$ 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. 2.16, 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.

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 $y = 1$

$$p(\boldsymbol{x}, y = 1) = p(\boldsymbol{x} \mid y = 1)\, p(y = 1) . \tag{2.22}$$

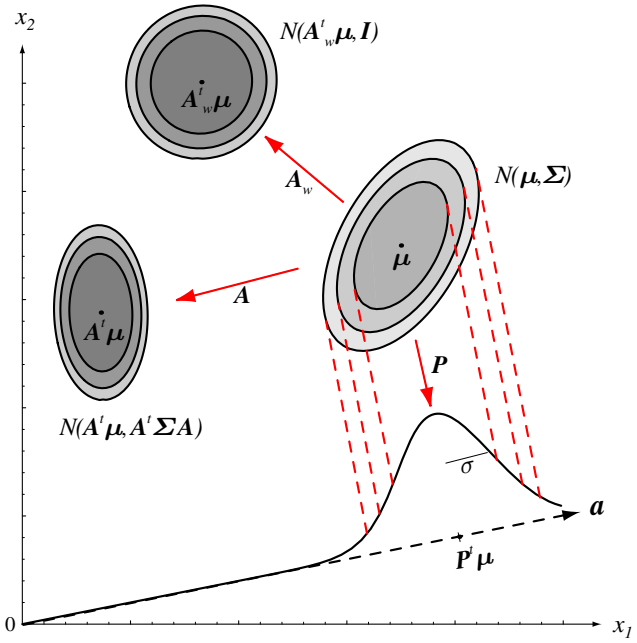


Figure 2.16: Linear transformations of the Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The linear transformation $\boldsymbol{A}$ leads to the Gaussian $\mathcal{N}\left(\boldsymbol{A}^T\boldsymbol{\mu}, \boldsymbol{A}^T\boldsymbol{\Sigma}\,\boldsymbol{A}\right)$. All projections $\boldsymbol{P}$ of a Gaussian are Gaussian. A certain transformation A_w ("whitening") leads to a Gaussian with the identity matrix as covariance matrix.

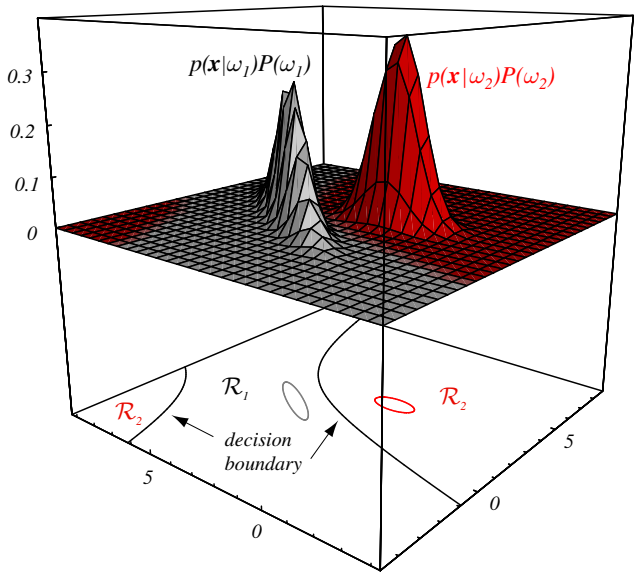


Figure 2.17: A two-dimensional classification task where the data for each class are drawn from a Gaussian (black: class 1, red: class -1). The optimal decision boundaries are two hyperbolas. Here $\omega_1 \equiv y = 1$ and $\omega_2 \equiv y = -1$. In the gray regions $p(y = 1 \mid \boldsymbol{x}) > p(y = -1 \mid \boldsymbol{x})$ holds and in the red regions the opposite holds. Copyright © 2001 John Wiley & Sons, Inc.

Fig. 2.17 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

$$p(\boldsymbol{x}) \;=\; p(\boldsymbol{x}, y = 1) \;+\; p(\boldsymbol{x}, y = -1)\;. \tag{2.23}$$

Here the variable y is "integrated out".

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

$$p(y = 1 \mid \boldsymbol{x}) \;=\; \frac{p(\boldsymbol{x} \mid y = 1)\, p(y = 1)}{p(\boldsymbol{x})}\;. \tag{2.24}$$

This formula is obtained by applying the Bayes rule.

We define the regions of class 1 as

$$X_1 \;=\; \{\boldsymbol{x} \mid g(\boldsymbol{x}) > 0\} \tag{2.25}$$

and regions of class -1 as

$$X_{-1} \;=\; \{\boldsymbol{x} \mid g(\boldsymbol{x}) < 0\}\;. \tag{2.26}$$

and the loss function as

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) \;=\; \begin{cases} 0 & \text{for} \quad y\, g(\boldsymbol{x}; \boldsymbol{w}) > 0 \\ 1 & \text{for} \quad y\, g(\boldsymbol{x}; \boldsymbol{w}) < 0 \end{cases}\;. \tag{2.27}$$

The risk of eq. (2.3) is for the zero-one loss

$$R(g(.;\boldsymbol{w})) \;=\; \int_{X_1} p\,(\boldsymbol{x}, y=-1)\; d\boldsymbol{x} \;+\; \int_{X_{-1}} p\,(\boldsymbol{x}, y=1)\; d\boldsymbol{x} \;= \tag{2.28}$$
$$\int_{X_1} p\,(y=-1 \mid \boldsymbol{x})\; p(\boldsymbol{x})\; d\boldsymbol{x} \;+\; \int_{X_{-1}} p\,(y=1 \mid \boldsymbol{x})\; p(\boldsymbol{x})\; d\boldsymbol{x} \;=$$
$$\int_X \left\{ \begin{array}{lll} p\,(y=-1 \mid \boldsymbol{x}) & \text{for} & g(\boldsymbol{x}) > 0 \\ p\,(y=1 \mid \boldsymbol{x}) & \text{for} & g(\boldsymbol{x}) < 0 \end{array} \right\} p(\boldsymbol{x})\; d\boldsymbol{x}\;.$$

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

$$g(\boldsymbol{x};\boldsymbol{w}) \left\{ \begin{array}{lll} > 0 & \text{for} & p\,(y=1 \mid \boldsymbol{x}) \;>\; p\,(y=-1 \mid \boldsymbol{x}) \\ < 0 & \text{for} & p\,(y=-1 \mid \boldsymbol{x}) \;>\; p\,(y=1 \mid \boldsymbol{x}) \end{array} \right. . \tag{2.29}$$

The minimal risk is

$$R_{\min} \;=\; \int_X \min\{p\,(\boldsymbol{x}, y=-1)\,, p\,(\boldsymbol{x}, y=1)\}\; d\boldsymbol{x} \;= \tag{2.30}$$
$$\int_X \min\{p\,(y=-1 \mid \boldsymbol{x})\,, p\,(y=1 \mid \boldsymbol{x})\}\; p(\boldsymbol{x})\; d\boldsymbol{x}\;.$$

Because at each point either class $y=1$ or class $y=-1$ will be misclassified we classify the point as belonging to the class with higher probability. This is demonstrated in Fig. 2.18 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. (2.28). 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. 2.19 shows such an example.

We call function g a *discriminant function* if it has a positive value at $\boldsymbol{x}$ and the corresponding data point is 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

$$\hat{y}(\boldsymbol{x}) \;=\; \text{sign}\; g(\boldsymbol{x})\;. \tag{2.31}$$

A discriminant function which minimizes the future risk is

$$g(\boldsymbol{x}) \;=\; p(y=1 \mid \boldsymbol{x}) \;-\; p(y=-1 \mid \boldsymbol{x}) \;= \tag{2.32}$$
$$\frac{1}{p(\boldsymbol{x})}\,(\; p(\boldsymbol{x} \mid y=1)\, p(y=1) \;-\; p(\boldsymbol{x} \mid y=-1)\, p(y=-1)\;)\;,$$

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 $p(y=$

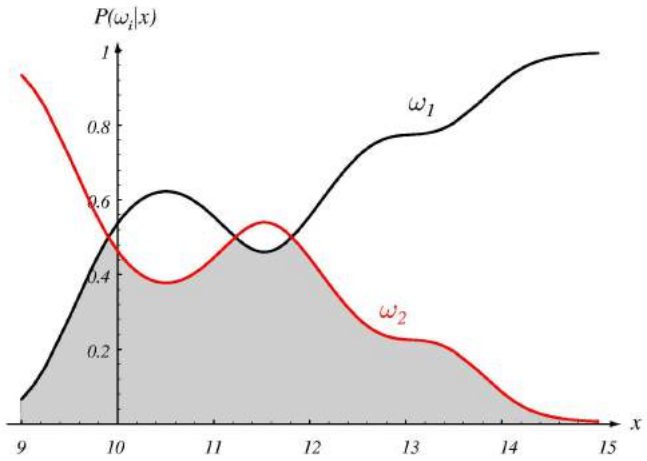


Figure 2.18: Posterior densities $p(y = 1 \mid \boldsymbol{x})$ and $p(y = -1 \mid \boldsymbol{x})$ as a function of $\boldsymbol{x}$. If using the optimal discriminant function the gray region is the integral eq. (2.28) and gives the probability of misclassifying a data point. Modified figure with copyright © 2001 John Wiley & Sons, Inc.

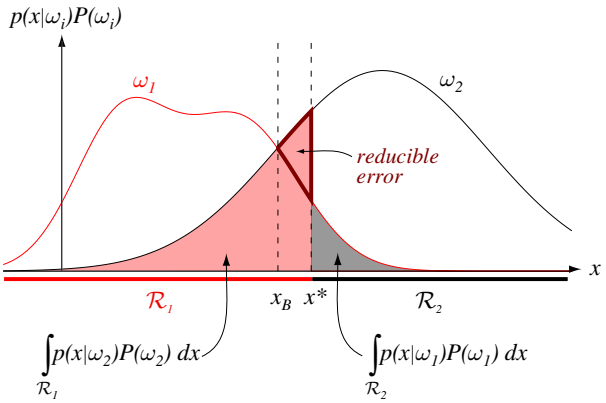


Figure 2.19: x^* is a non-optimal decision point because for some regions the posterior $y = 1$ is above the posterior $y = -1$ but data is classified as $y = -1$. The misclassification rate is given by the filled region. However the misclassification mass in the red triangle can be saved if using as decision point x_B. Copyright © 2001 John Wiley & Sons, Inc.

$1 \mid \boldsymbol{x})$ 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 minimizes the future risk:

$$g(\boldsymbol{x}) \;=\; \ln p(y=1 \mid \boldsymbol{x}) \;-\; \ln p(y=-1 \mid \boldsymbol{x}) \;= \qquad (2.33)$$
$$\ln \frac{p(\boldsymbol{x} \mid y=1)}{p(\boldsymbol{x} \mid y=-1)} \;+\; \ln \frac{p(y=1)}{p(y=-1)}\,.$$

For our Gaussian case we obtain

$$\begin{aligned} g(\boldsymbol{x}) \;=\; & -\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\,\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) \;-\; \frac{d}{2}\ln 2\pi \;- \qquad (2.34)\\ & \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \ln p(y=1) \;+\\ & \frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_{-1})^T\,\boldsymbol{\Sigma}_{-1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{-1}) \;+\; \frac{d}{2}\ln 2\pi \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;-\; \ln p(y=-1) \;=\\ & -\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\,\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) \;-\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \ln p(y=1) \;+\\ & \frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_{-1})^T\,\boldsymbol{\Sigma}_{-1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{-1}) \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;-\; \ln p(y=-1) \;=\\ & -\frac{1}{2}\boldsymbol{x}^T\left(\boldsymbol{\Sigma}_1^{-1} \;-\; \boldsymbol{\Sigma}_{-1}^{-1}\right)\,\boldsymbol{x} \;+\; \boldsymbol{x}^T\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 \;-\; \boldsymbol{\Sigma}_{-1}^{-1}\boldsymbol{\mu}_{-1}\right) \;-\\ & \frac{1}{2}\,\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\,\boldsymbol{\mu}_{-1}^T\boldsymbol{\Sigma}_{-1}^{-1}\boldsymbol{\mu}_{-1} \;-\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_1| \;+\; \frac{1}{2}\,\ln|\boldsymbol{\Sigma}_{-1}| \;+\\ & \ln p(y=1) \;-\; \ln p(y=-1) \;=\\ & -\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \;+\; \boldsymbol{w}^T\boldsymbol{x} \;+\; b\,. \end{aligned}$$

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

$$\begin{aligned} g(\boldsymbol{x}) \;=\; & \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\,(\boldsymbol{\mu}_1 \;-\; \boldsymbol{\mu}_{-1}) \;+ \qquad (2.35)\\ & \boldsymbol{\mu}_{-1}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_{-1} \;-\; \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 \;+\; \ln p(y=1) \;-\; \ln p(y=-1) \;=\\ & \boldsymbol{w}^T\boldsymbol{x} \;+\; b\,. \end{aligned}$$

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. 2.20.

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. 2.21.

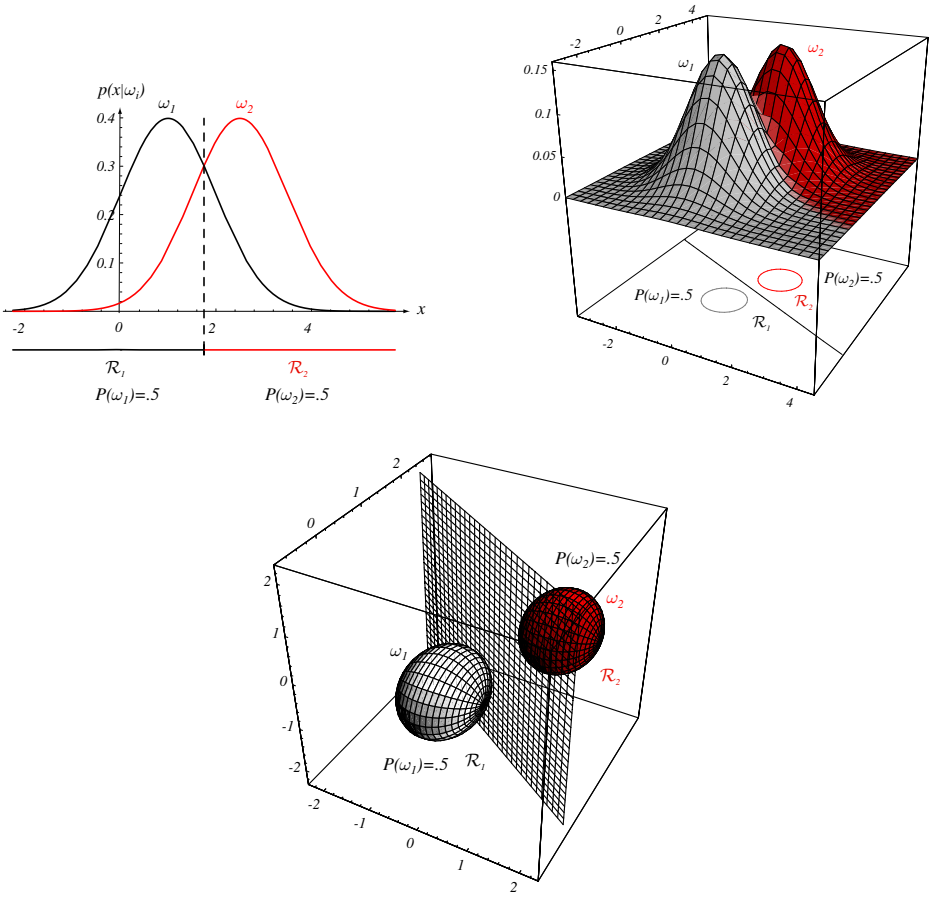


Figure 2.20: Two classes with covariance matrix $\mathbf{\Sigma} = \sigma^2 \boldsymbol{I}$ each in one (top left), two (top right), and three (bottom) dimensions. The optimal boundary is a hyperplane. Copyright © 2001 John Wiley & Sons, Inc.

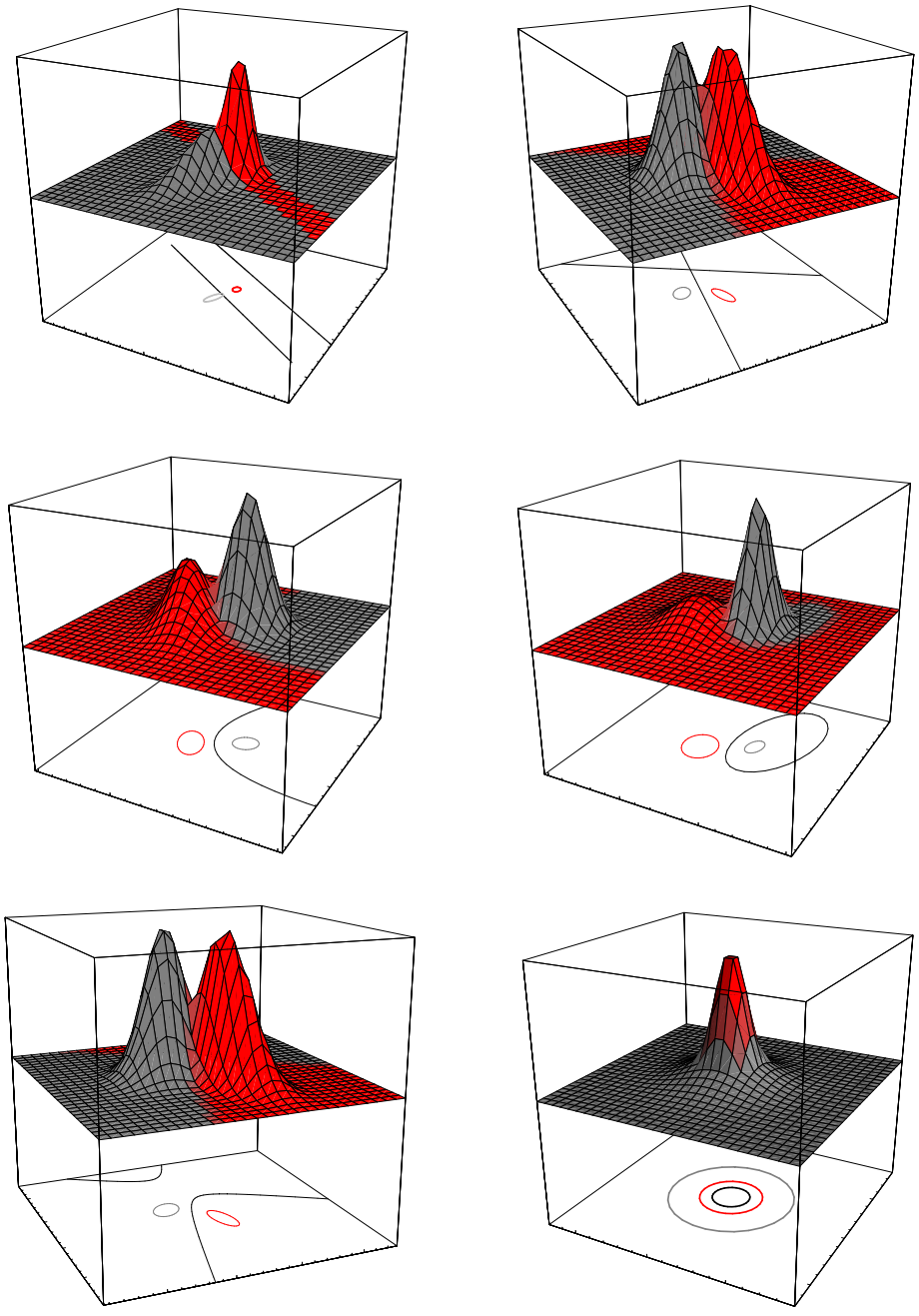


Figure 2.21: Two classes with arbitrary Gaussian covariance lead to boundary functions which are hyperplanes, hyper-ellipsoids, hyperparaboloids etc.

Chapter 3

Support Vector Machines

In this chapter we focus the "support vector machine" (SVM), one of the most prominent techniques in supervised machine learning. The SVM minimizes the structural risk, which is the training error (empirical risk) together with a bound on the complexity, namely the margin. See course "Theoretical Concepts of Machine Learning" for the concept of *structural risk minimization*.

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.

Since SVMs allow for bounds on the future error (see course "Theoretical Concepts of Machine Learning") and have been proven very successful in different applications, they were preferred to neural networks or other supervised learning methods. In contrast to neural networks the SVMs have a unique solution and can be solved by a convex quadratic optimization problem.

3.1 Support Vector Machines in Bioinformatics

Support vector machines 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 ways. 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 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, 2004a] 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 from 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 SVMs 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`.

3.2 Linearly Separable Problems

First we consider classification tasks which are linearly separable.

As in the 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 $\boldsymbol{y} = \left(y^1, \ldots, y^l\right)^T$.

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

$$y^i = \text{sign}\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right),\ 1 \leq i \leq l\,. \tag{3.1}$$

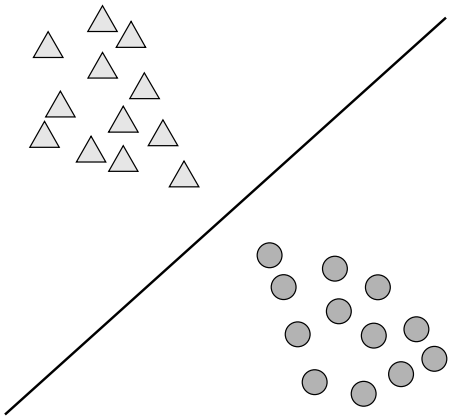


Figure 3.1: A linearly separable problem, where data points of class 1 (the circles) are on one side of the hyperplane (a line in two dimensions) and data points of class 2 (the triangles) are on the other side of the hyperplane.

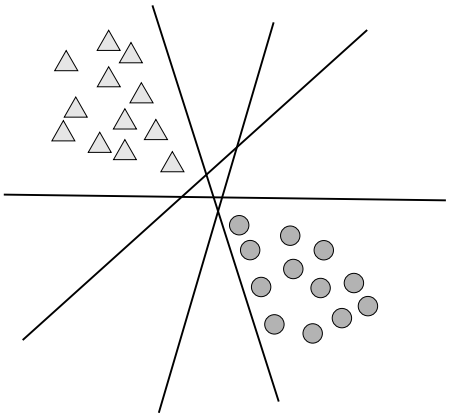


Figure 3.2: Different solutions for linearly separating the classes.

That means all positive examples are on one side of the boundary function $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ and all negative examples are on the other side. Fig. 3.1 shows a two-dimensional example for a linear separable problem.

However the data can be separated by different hyperplanes as shown in Fig. 3.2 for the two-dimensional example. Which is the best separating hyperplane?

Our theoretical considerations in the last chapter suggest to use the classification function with the lowest complexity. For the complexity measure we will use the margin. Fig. 3.3 gives an intuition why a larger margin is desirable as data points can be noisy without jumping over the boundary function.

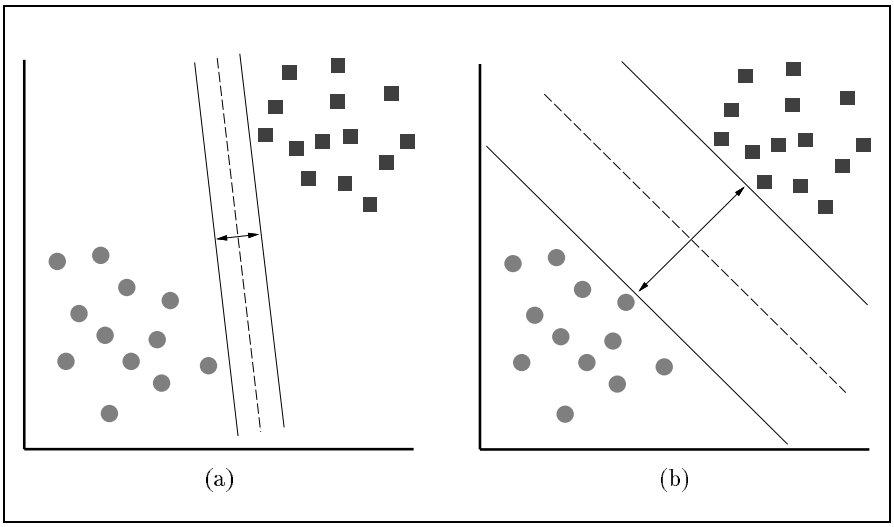


Figure 3.3: Intuitively, better generalization is expected from separation on the right hand side than from the left hand side because larger distance to the data points allows for noisy data. Copyright © 1997 [Osuna et al., 1997].

3.3 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* is defined as the minimal distance of the boundary function $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ to all data points. 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

$$\min_{i=1,\ldots,l} \left|\boldsymbol{w}^T\boldsymbol{x}^i + b\right| = 1 \,. \tag{3.2}$$

optimizing b in order to obtain the smallest $\boldsymbol{w}$ of vectors having the same directions as $\boldsymbol{w}$ means that 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 margin is given by

$$\gamma = \frac{1}{\|\boldsymbol{w}\|} \,. \tag{3.3}$$

The situation is depicted in Fig. 3.4.

To optimize b together with the length of $\boldsymbol{w}$ is straightforward as it only moves the separating hyperplane. However it is not clear how to optimize the direction $\boldsymbol{w}$. We will do this in the following.

If we assume correct classification and the canonical form then

$$\boldsymbol{w}^T\boldsymbol{x}^i + b \geq 1 \text{ for } y^i = 1 \text{ and} \tag{3.4}$$

$$\boldsymbol{w}^T\boldsymbol{x}^i + b \leq -1 \text{ for } y^i = -1 \,. \tag{3.5}$$

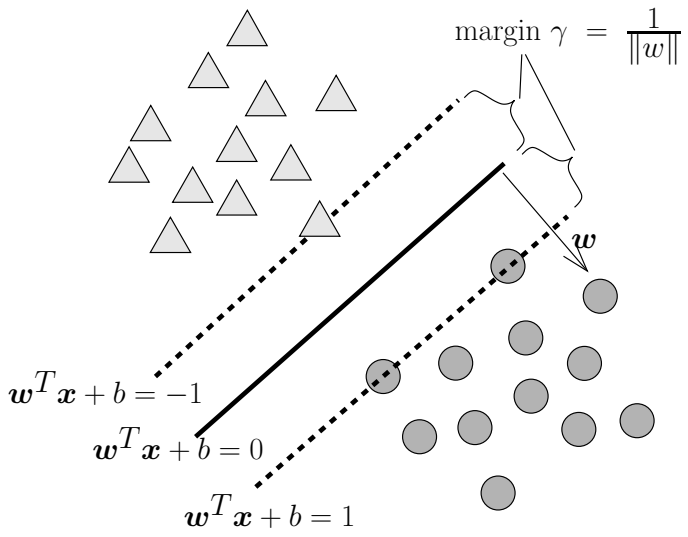


Figure 3.4: For the hyperplane described by the canonical discriminant function and for the optimal offset b (same distance to class 1 and class 2), the margin is $\gamma = \frac{1}{\|\boldsymbol{w}\|}$. At least one point exists for which $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and at least one point 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:

$$\begin{aligned} &\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t. } \boldsymbol{w}^T\boldsymbol{x}^i + b \geq 1 \text{ for } y^i = 1 \\ &\qquad \boldsymbol{w}^T\boldsymbol{x}^i + b \leq -1 \text{ for } y^i = -1\,. \end{aligned} \tag{3.6}$$

This can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t. } y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) \geq 1\,. \end{aligned} \tag{3.7}$$

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

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{l}\alpha_i\left(y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) - 1\right), \tag{3.8}$$

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:

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i = \mathbf{0} \text{ and} \tag{3.9}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{l} \alpha_i \, y^i = 0 \, . \tag{3.10}$$

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

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \, . \tag{3.11}$$

And the second equation gives an equality constraint

$$\sum_{i=1}^{l} \alpha_i \, y^i = 0 \, . \tag{3.12}$$

The expression for $\boldsymbol{w}$ in eq. (3.11) can be substituted into the Lagrangian:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}) &= \frac{1}{2} \left(\sum_{j=1}^{l} \alpha_j \, y^j \, \left(\boldsymbol{x}^j\right)^T \right) \left(\sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \right) - \\ &\sum_{i=1}^{l} \alpha_i \, y^i \left(\sum_{j=1}^{l} \alpha_j \, y^j \, \left(\boldsymbol{x}^j\right)^T \right) \boldsymbol{x}^i - b \sum_{i=1}^{l} \alpha_i \, y^i + \sum_{i=1}^{l} \alpha_i = \\ &\frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i - \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i + \sum_{i=1}^{l} \alpha_i = \\ &- \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i + \sum_{i=1}^{l} \alpha_i \, . \end{aligned} \tag{3.13}$$

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

$$\frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i - \sum_{i=1}^{l} \alpha_i \, . \tag{3.14}$$

The dual formulation of the optimization problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} & \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \, \left(\boldsymbol{x}^j\right)^T \boldsymbol{x}^i \, - \, \sum_{i=1}^{l} \alpha_i \\ \text{s.t. } & \alpha_i \, \geq \, 0 \\ & \sum_{i=1}^{l} \alpha_i \, y^i \, = \, 0 \, . \end{aligned} \tag{3.15}$$

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

$$\begin{aligned} \min_{\boldsymbol{\alpha}} & \frac{1}{2} \, \boldsymbol{\alpha}^T \boldsymbol{Y}^T \boldsymbol{X}^T \boldsymbol{X} \, \boldsymbol{Y} \, \boldsymbol{\alpha} \, - \, \mathbf{1}^T \boldsymbol{\alpha} \\ \text{s.t. } & \boldsymbol{\alpha} \, \geq \, \mathbf{0} \\ & \boldsymbol{\alpha}^T \boldsymbol{y} \, = \, 0 \, , \end{aligned} \tag{3.16}$$

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

$$\boldsymbol{w} \, = \, \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \, . \tag{3.17}$$

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 $\boldsymbol{x}^i$:

$$\boldsymbol{w}^T \boldsymbol{x} \, + b \, = \, \sum_{i=1}^{l} \alpha_i \, y^i \, \left(\boldsymbol{x}^i\right)^T \boldsymbol{x} \, + \, b \, . \tag{3.18}$$

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

$$\alpha_i \, \left(y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \, + \, b\right) \, - \, 1\right) \, = \, 0 \tag{3.19}$$

It follows that either

$$\alpha_i \, = \, 0 \text{ or} \tag{3.20}$$

$$y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \, + \, b\right) \, = \, 1 \, . \tag{3.21}$$

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 vector $\boldsymbol{x}^i$ because

$$y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \, + \, b\right) \, = \, 1 \tag{3.22}$$

therefore

$$b \;=\; y^i \;-\; \boldsymbol{w}^T \boldsymbol{x}^i \;. \tag{3.23}$$

If the solutions α_i are not exact then b can be computed by averaging the computed b values over all support vectors.

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

$$b \sum_{i=1}^{l} \alpha_i \, y^i \;=\; 0 \tag{3.24}$$

holds. We have for the optimal solution

$$\begin{aligned} \boldsymbol{w}^T \boldsymbol{w} \;&=\; \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{w}^T \boldsymbol{x}^i \;= \\ &\sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{w}^T \boldsymbol{x}^i \;+\; b \sum_{i=1}^{l} \alpha_i \, y^i \;-\; \sum_{i=1}^{l} \alpha_i \;+\; \sum_{i=1}^{l} \alpha_i \;= \\ &\sum_{i=1}^{l} \alpha_i \, \left(y^i \, \left(\boldsymbol{w}^T \boldsymbol{x}^i \;+\; b \right) \;-\; 1 \right) \;+\; \sum_{i=1}^{l} \alpha_i \;= \\ &\sum_{i=1}^{l} \alpha_i \;, \end{aligned} \tag{3.25}$$

where we used the KKT conditions eq. (3.19).

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

$$\gamma \;=\; \frac{1}{\sqrt{\sum_{i=1}^{l} \alpha_i}} \;. \tag{3.26}$$

3.4 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. 3.6 for a non-linear separable task.

In Fig. 3.7 on the top row two problems are depicted which are not linearly separable. However if data points can be moved then these problems are again linear separable.

What does moving mean? 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

$$\left(\boldsymbol{x}^i \right)^{\text{moved}} \;=\; \boldsymbol{x}^i \;+\; y^i \, \delta_i \, \boldsymbol{w} \tag{3.27}$$

$$\boldsymbol{w}^T \left(\boldsymbol{x}^i \right)^{\text{moved}} \;=\; \boldsymbol{w}^T \boldsymbol{x}^i \;+\; y^i \, \delta_i \, \|\boldsymbol{w}\|^2 \tag{3.28}$$

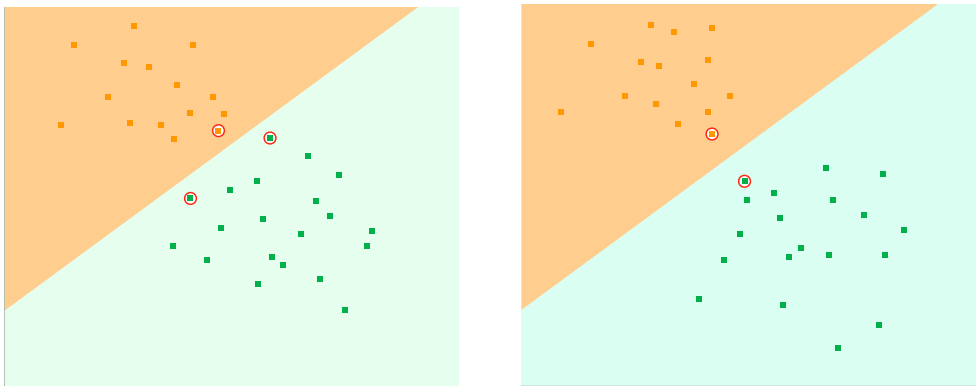


Figure 3.5: Two examples for linear SVMs. The class of points is given by their color. The circled points are support vectors and the color of the regions correspond to the class points in this region would be classified.

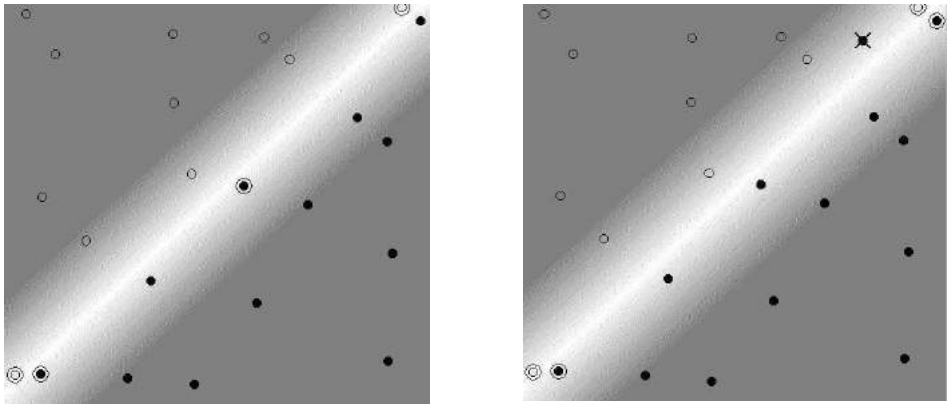


Figure 3.6: Left: linear separable task. Right: a task which is not linearly separable, where the filled point marked with a cross is in the region of the non-filled points. Circled points are support vectors. Copyright © 1998 [Burges, 1998].

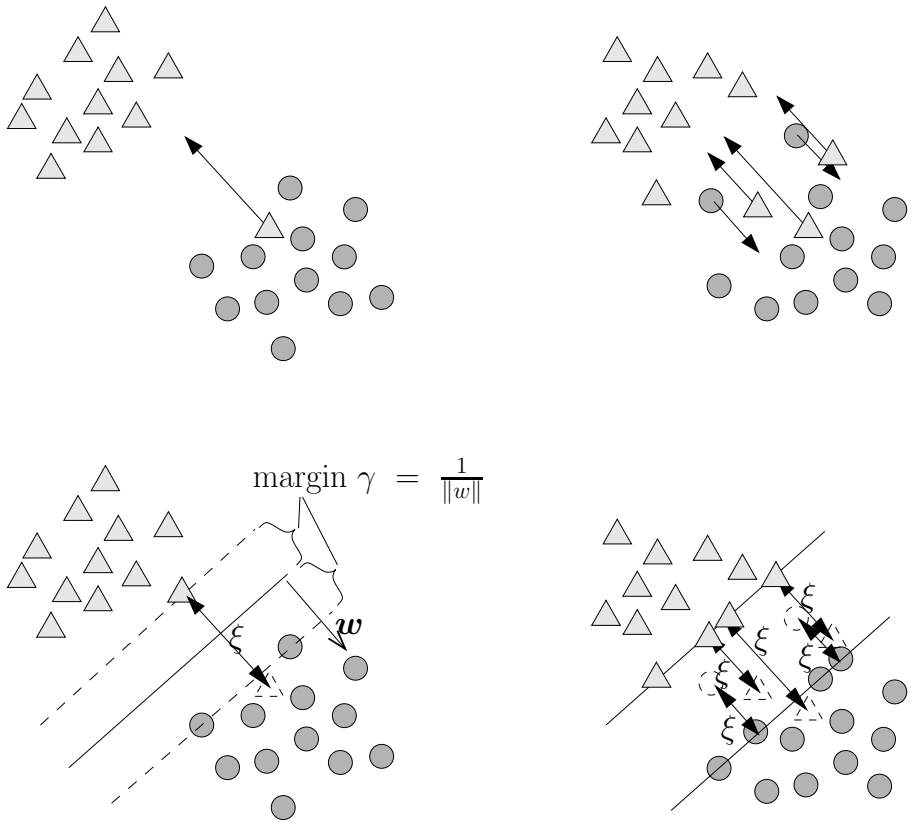


Figure 3.7: Two problems at the top row which are not linearly separable. If data points can be moved then the problem would be linear separable as shown in the bottom row.

and obtain

$$y^i \left(\boldsymbol{w}^T \left(\boldsymbol{x}^i \right)^{\text{moved}} + b \right) \geq 1 \tag{3.29}$$

$$\Leftrightarrow \; y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) \geq 1 - \delta_i \left\| \boldsymbol{w} \right\|^2 . \tag{3.30}$$

If we set $\xi_i = \delta_i \left\| \boldsymbol{w} \right\|^2$ then we obtain

$$y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) \geq 1 - \xi_i . \tag{3.31}$$

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

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; & \frac{1}{2} \left\| \boldsymbol{w} \right\|^2 + C \sum_{i=1}^{l} \xi_i \\ \text{s.t. } & y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) \geq 1 - \xi_i \\ & \xi_i \geq 0 . \end{aligned} \tag{3.32}$$

The Lagrange function is

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu}) = & \frac{1}{2} \left\| \boldsymbol{w} \right\|^2 + C \sum_{i=1}^{l} \xi_i - \\ & \sum_{i=1}^{l} \alpha_i \left(y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) - 1 + \xi_i \right) - \sum_{i=1}^{l} \mu_i \, \xi_i , \end{aligned} \tag{3.33}$$

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:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i = \boldsymbol{0} \tag{3.34}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{l} \alpha_i \, y^i = 0 \tag{3.35}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} = \boldsymbol{1} C - \boldsymbol{\alpha} - \boldsymbol{\mu} = \boldsymbol{0} . \tag{3.36}$$

Again we obtain

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i . \tag{3.37}$$

and

$$\sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \, . \tag{3.38}$$

Additionally, from the last derivative we obtain

$$\alpha_i \; \leq \; C \tag{3.39}$$

because $\mu_i \; \geq \; 0$.

The expression for $\boldsymbol{w}$ can be again substituted into the Lagrangian:

$$\begin{aligned}
\mathcal{L} \; = \; & \frac{1}{2} \left(\sum_{i=1}^{l} \alpha_i \, y^i \; \left(\boldsymbol{x}^i \right)^T \right) \left(\sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \right) \; + \; C \; \sum_{i=1}^{l} \xi_i \; - \\
& \sum_{i=1}^{l} \alpha_i \, y^i \left(\sum_{j=1}^{l} \alpha_j \, y^j \; \left(\boldsymbol{x}^j \right)^T \right) \boldsymbol{x}^i \; - \; b \; \sum_{i=1}^{l} \alpha_i \, y^i \; + \; \sum_{i=1}^{l} \alpha_i \; - \\
& \sum_{i=1}^{l} \alpha_i \, \xi_i \; - \; \sum_{i=1}^{l} \mu_i \, \xi_i \; = \\
& - \; \frac{1}{2} \; \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \; + \; \sum_{i=1}^{l} \xi_i \, (C \; - \; \alpha_i \; - \; \mu_i) \; + \; \sum_{i=1}^{l} \alpha_i \; = \\
& - \; \frac{1}{2} \; \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \; + \; \sum_{i=1}^{l} \alpha_i \, .
\end{aligned} \tag{3.40}$$

We again minimize the negative Lagrangian

$$\frac{1}{2} \; \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \; - \; \sum_{i=1}^{l} \alpha_i \, . \tag{3.41}$$

The dual formulation of the optimization problem with slack variables is

$$\begin{aligned}
& \min_{\boldsymbol{\alpha}} \frac{1}{2} \; \sum_{i,j} \alpha_i \, \alpha_j \, y^i \, y^j \; \left(\boldsymbol{x}^j \right)^T \boldsymbol{x}^i \; - \; \sum_{i=1}^{l} \alpha_i \\
& \text{s.t. } 0 \; \leq \; \alpha_i \; \leq \; C \\
& \quad \sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \, .
\end{aligned} \tag{3.42}$$

The dual formulation with slack variables differs from the dual without slack variables in eq. (3.42) only through the upper bound C on α_i.

$$\begin{aligned}
& \min_{\boldsymbol{\alpha}} \frac{1}{2} \; \boldsymbol{\alpha}^T \boldsymbol{Y}^T \boldsymbol{X}^T \boldsymbol{X} \, \boldsymbol{Y} \, \boldsymbol{\alpha} \; - \; \mathbf{1}^T \boldsymbol{\alpha} \\
& \text{s.t. } \mathbf{0} \; \leq \; \boldsymbol{\alpha} \; \leq \; C \mathbf{1} \\
& \quad \boldsymbol{\alpha}^T \boldsymbol{y} \; = \; 0 \, ,
\end{aligned} \tag{3.43}$$

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

The Karush-Kuhn-Tucker (KKT) conditions require:

$$\alpha_i \left(y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) - 1 + \xi_i \right) = 0 \tag{3.44}$$

$$\mu_i \, \xi_i = 0 \, . \tag{3.45}$$

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

$$\alpha_i > 0 : \tag{3.46}$$

$$\Rightarrow \begin{cases} \xi_i = 1 - y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) > 0 & \text{and} \quad \alpha_i = C \\ y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) = 1 & \text{and} \quad 0 < \alpha_i < C \end{cases}$$

$$\alpha_i = 0 : \tag{3.47}$$

$$\Rightarrow \xi_i = 0 \text{ and } y^i \left(\boldsymbol{w}^T \boldsymbol{x}^i + b \right) > 1 \, .$$

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. 3.8.

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 remain the same.

3.5 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.

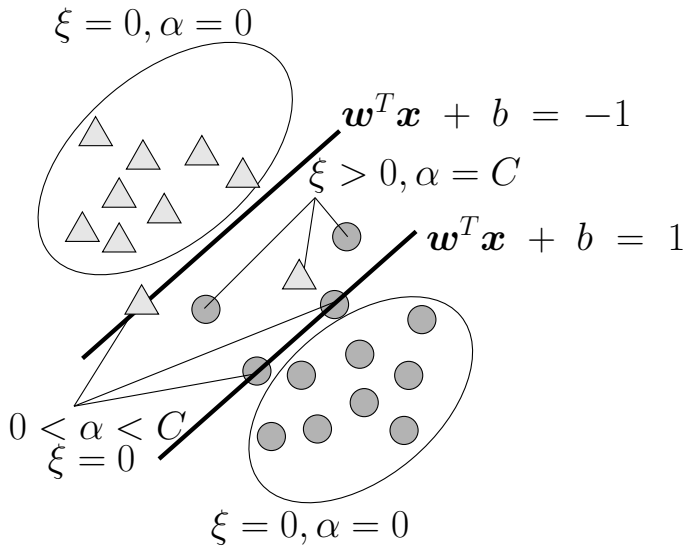


Figure 3.8: Typical situation for the C-SVM. The slack variables for data points within the boundaries $\boldsymbol{w}^T\boldsymbol{x}^i + b = 1$ and $\boldsymbol{w}^T\boldsymbol{x}^i + b = -1$ are $\xi > 0$ with $\alpha = C$, for data points on the boundaries $\xi = 0,\ 0 < \alpha < C$, and for all other data points $\xi = 0,\ alpha = 0$. Support vectors are data points with $\alpha > 0$.

The primal formulation of the ν-SVM is

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \ & \frac{1}{2}\|\boldsymbol{w}\|^2 - \nu\,\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ \text{s.t. } & y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) \geq \rho - \xi_i \\ & \xi_i \geq 0 \text{ and } \rho \geq 0\,. \end{aligned} \tag{3.48}$$

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 negative sign. 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

$$y^i\left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) < \rho\,. \tag{3.49}$$

Theorem 3.1 (ν-SVM Properties)
For the solution of the ν-SVM optimization problem eq. (3.48) 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

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu}, \rho, \delta) \;=\; \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \nu\,\rho \;+\; \frac{1}{l}\,\sum_{i=1}^{l}\xi_i \;- \tag{3.50}$$

$$\sum_{i=1}^{l}\alpha_i\;\left(y^i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;-\; \rho \;+\; \xi_i\right) \;-\; \sum_{i=1}^{l}\mu_i\,\xi_i \;-\; \delta\,\rho\,.$$

At the minimum the derivatives of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu}, \rho, \delta)$ are zero:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} \;=\; \boldsymbol{w} \;-\; \sum_{i=1}^{l}\alpha_i\,y^i\,\boldsymbol{x}^i \;=\; \mathbf{0} \tag{3.51}$$

$$\frac{\partial\mathcal{L}}{\partial b} \;=\; \sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0 \tag{3.52}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} \;=\; \frac{1}{l}\mathbf{1} \;-\; \boldsymbol{\alpha} \;-\; \boldsymbol{\mu} \;=\; \mathbf{0} \tag{3.53}$$

$$\frac{\partial\mathcal{L}}{\partial\rho} \;=\; \sum_{i=1}^{l}\alpha_i \;-\; \delta \;-\; \nu \;=\; 0\,. \tag{3.54}$$

If we substitute these values into the Lagrangian we obtain

$$\mathcal{L} \;=\; -\,\frac{1}{2}\,\sum_{i,j}\alpha_i\,\alpha_j\,y^i\,y^j\;\left(\boldsymbol{x}^j\right)^T\boldsymbol{x}^i\,. \tag{3.55}$$

The dual is

$$\min_{\boldsymbol{\alpha}}\frac{1}{2}\,\sum_{i,j}\alpha_i\,\alpha_j\,y^i\,y^j\;\left(\boldsymbol{x}^j\right)^T\boldsymbol{x}^i \tag{3.56}$$

$$\text{s.t. } 0 \;\leq\; \alpha_i \;\leq\; \frac{1}{l}$$

$$\sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0$$

$$\sum_{i=1}^{l}\alpha_i \;\geq\; \nu\,.$$

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:

$$\alpha_i\;\left(y^i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;-\; \rho \;+\; \xi_i\right) \;=\; 0 \tag{3.57}$$

$$\mu_i\,\xi_i \;=\; 0 \tag{3.58}$$

$$\delta\,\rho \;=\; 0\,. \tag{3.59}$$

The last equation together with eq. (3.54) 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

$$\rho \;=\; \frac{1}{2}\left(\boldsymbol{w}^T \boldsymbol{x}^1 \;-\; \boldsymbol{w}^T \boldsymbol{x}^2\right) \tag{3.60}$$

$$b \;=\; -\,\frac{1}{2}\left(\boldsymbol{w}^T \boldsymbol{x}^1 \;+\; \boldsymbol{w}^T \boldsymbol{x}^2\right) \tag{3.61}$$

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

$$\nu \;\leq\; \frac{2\,k}{l}\,. \tag{3.62}$$

If $k \geq \frac{l}{2}$, we can apply the above argumentation and obtain

$$\nu \;\leq\; \frac{2\,(l-k)}{l}\,. \tag{3.63}$$

Merging these two cases, we obtain

$$\nu \;\leq\; \frac{2\;\min\{k, (l-k)\}}{l}\,. \tag{3.64}$$

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

$$\boldsymbol{w}^* \;=\; \boldsymbol{w}/\rho \tag{3.65}$$

$$b^* \;=\; b/\rho \tag{3.66}$$

$$\boldsymbol{\xi}^* \;=\; \boldsymbol{\xi}/\rho \tag{3.67}$$

and we obtain as primal for the ν-SVM with the new variables:

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*,\rho} \; & \frac{1}{2}\,\rho^2\,\|\boldsymbol{w}^*\|^2 \;-\; \nu\,\rho \;+\; \rho\,\frac{1}{l}\,\sum_{i=1}^{l}\xi_i^* \\ \text{s.t. } & \rho\,y^i\;\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; \rho \;-\; \rho\,\xi_i^* \\ & \rho\,\xi_i^* \;\geq\; 0 \;\text{ and }\; \rho \;\geq\; 0 \end{aligned} \tag{3.68}$$

that is

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*,\rho} \; & \rho^2\,\left(\frac{1}{2}\,\|\boldsymbol{w}^*\|^2 \;-\; \frac{\nu}{\rho} \;+\; \frac{1}{l\,\rho}\,\sum_{i=1}^{l}\xi_i^*\right) \\ \text{s.t. } & y^i\;\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; 1 \;-\; \xi_i^* \\ & \xi_i^* \;\geq\; 0 \;\text{ and }\; \rho \;\geq\; 0\,. \end{aligned} \tag{3.69}$$

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:

$$\begin{aligned} \min_{\boldsymbol{w}^*,b^*,\boldsymbol{\xi}^*} \; & \frac{1}{2}\,\|\boldsymbol{w}^*\|^2 \;+\; C\,\sum_{i=1}^{l}\xi_i^* \\ \text{s.t. } & y^i\;\left((\boldsymbol{w}^*)^T\boldsymbol{x}^i \;+\; b^*\right) \;\geq\; 1 \;-\; \xi_i^* \\ & \xi_i^* \;\geq\; 0\,. \end{aligned} \tag{3.70}$$

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.

3.6 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:

$$\boldsymbol{\Phi}:\;\; \mathbb{R}^d \to \mathbb{R}^{d_\Phi},\; \boldsymbol{x} \;\mapsto\; \boldsymbol{x}_\phi,\; \boldsymbol{x}_\phi \;=\; \boldsymbol{\Phi}(\boldsymbol{x})\,. \tag{3.71}$$

In this feature space we can apply the linear theory of SVMs. Afterwards we can project the results back into the original space. Fig. 3.9 depicts the feature mapping. In the feature space the 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.

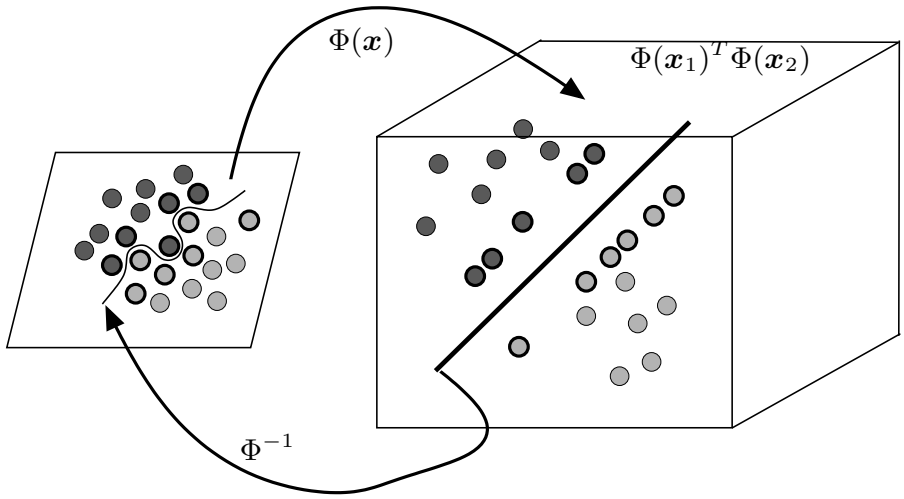


Figure 3.9: Nonlinearly separable data is mapped into a feature space where the data is linear separable. The support vectors in feature space are marked by thicker borders. These vectors as well as the boundary function are shown in the original space where the linear boundary function becomes a nonlinear boundary function.

For example consider

$$\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \left(x_1^2 \,,\; x_2^2 \,,\; \sqrt{2}\, x_1\, x_2\right) \,. \tag{3.72}$$

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. 3.10 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

$$\begin{aligned}
&\boldsymbol{\Phi}^T(\boldsymbol{x}^i)\boldsymbol{\Phi}(\boldsymbol{x}^j) \;=\; x_{i1}^2\, x_{j1}^2 \;+\; x_{i2}^2\, x_{j2}^2 \;+\; 2\, x_{i1}\, x_{i2}\, x_{j1}\, x_{j2} \;= \\
&(x_{i1}\, x_{j1} \;+\; x_{i2}\, x_{j2})^2 \;=\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j\right)^2 \,.
\end{aligned}$$

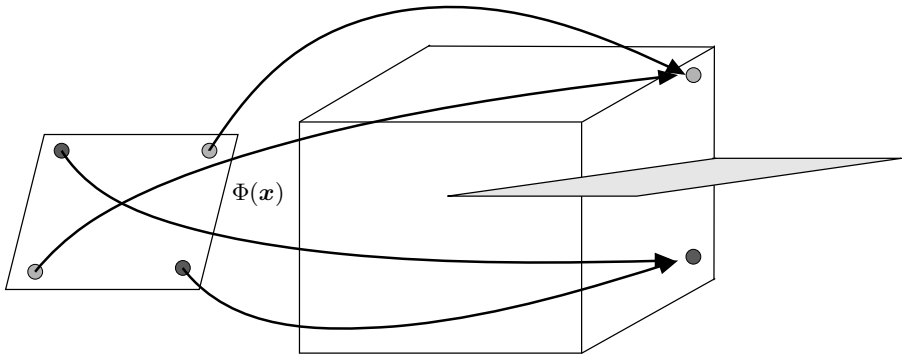


Figure 3.10: An example of a mapping from the two-dimensional space into the three-dimensional space. The data points are not linearly separable in the two-dimensional space but in the three-dimensional space.

In another example we can map the two-dimensional vectors into a 9-dimensional space by

$$\begin{aligned}\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \Big(& x_1^3\,,\; x_2^3\,,\; \sqrt{3}\,x_1^2\,x_2\,,\; \sqrt{3}\,x_2^2\,x_1\,, \\ & \sqrt{3}\,x_1^2\,,\; \sqrt{3}\,x_2^2\,,\; \sqrt{6}\,x_1\,x_2\,,\; \sqrt{3}\,x_1\,,\; \sqrt{3}\,x_2\Big)\;.\end{aligned} \tag{3.73}$$

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

$$\begin{aligned}\boldsymbol{w}^T\boldsymbol{x} \;&=\; \sum_{i=1}^{l}\alpha_i\,y^i\;\big(\boldsymbol{x}^i\big)^T\,\boldsymbol{x} \;= \\ &\sum_{i=1}^{l}\alpha_i\,y^i\;\Big(\big(\boldsymbol{x}^i\big)^T\,\boldsymbol{x} \;+c\Big)\;,\end{aligned} \tag{3.74}$$

for a constant c, because we have $\sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0$.

Fig. 3.11 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 $\Big(\big(\boldsymbol{x}^i\big)^T\,\boldsymbol{x}^j \;+\; 1\Big)^3$.

A function which produces a scalar out of two vectors is called *kernel* k. In our example we have $k\left(\boldsymbol{x}^i,\boldsymbol{x}^j\right) \;=\; \Big(\big(\boldsymbol{x}^i\big)^T\,\boldsymbol{x}^j \;+\; 1\Big)^3$.

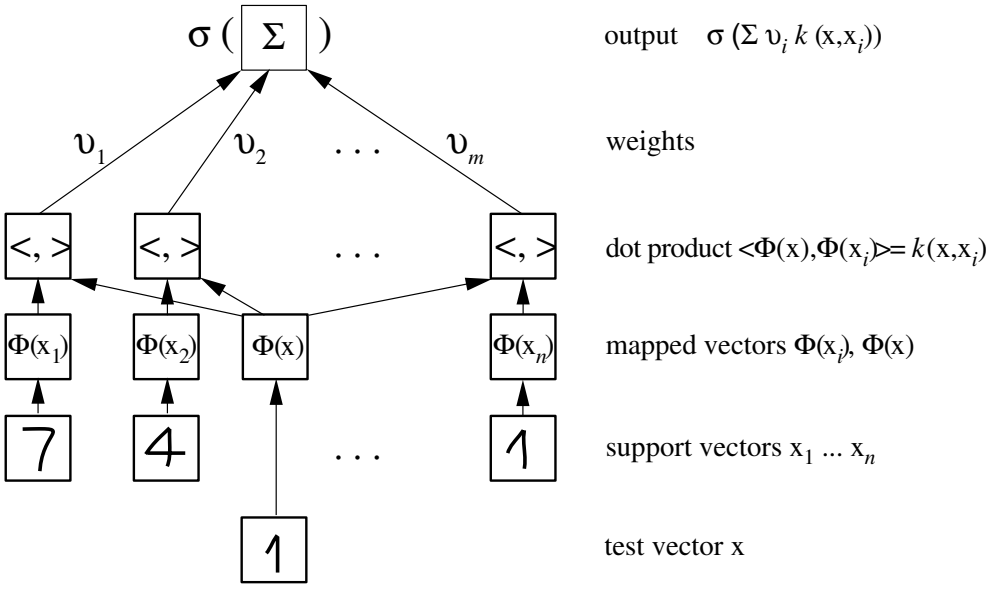


Figure 3.11: The support vector machine with mapping into a feature space is depicted. Images of digits are represented by vectors $\boldsymbol{x}$. These vectors are mapped by $\boldsymbol{\Phi}$ into a feature space in which a dot product is performed. Certain digits are support vectors $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n$. 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.

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 3.2 (Mercer)
Let the kernel k be symmetric and from $L_2(X \times X)$ defining a Hilbert-Schmidt operator

$$T_k(f)(\boldsymbol{x}) \;=\; \int_X k(\boldsymbol{x}, \boldsymbol{x}')\; f(\boldsymbol{x}')\; d\boldsymbol{x}' \;. \tag{3.75}$$

If T_k is positive semi-definite, i.e. for all $f \in L_2(X)$

$$\int_{X \times X} k(\boldsymbol{x}, \boldsymbol{x}')\; f(\boldsymbol{x})\; f(\boldsymbol{x}')\; d\boldsymbol{x}\; d\boldsymbol{x}' \;\geq\; 0 \;, \tag{3.76}$$

then T_k has eigenvalues $\lambda_j \;\geq\; 0$ with associated eigenfunctions $\psi_j \in L_2(X)$. Further

$$(\lambda_1, \lambda_2, \ldots) \in \ell_1 \tag{3.77}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') \;=\; \sum_j \lambda_j\; \psi_j(\boldsymbol{x})\; \psi_j(\boldsymbol{x}') \;, \tag{3.78}$$

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. (3.76) 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 $\alpha_i, 1 \leq i \leq l$

$$\sum_{i,j=1,1}^{l,l} \alpha_i\; \alpha_j\; k(\boldsymbol{x}^i, \boldsymbol{x}^j) \;\geq\; 0 \;. \tag{3.79}$$

We define the *Gram matrix* $\boldsymbol{K}$ as $K_{ij} \;=\; k(\boldsymbol{x}^i, \boldsymbol{x}^j)$. The above inequality is

$$\boldsymbol{\alpha}^T \boldsymbol{K}\; \boldsymbol{\alpha} \;\geq\; 0 \;, \tag{3.80}$$

which means that the Gram matrix has non-negative eigenvalues, i.e. is positive semi-definite.

The mapping $\boldsymbol{\Phi}$ can be explicitly given as

$$\boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \left(\sqrt{\lambda_1}\; \psi_1(\boldsymbol{x}), \sqrt{\lambda_2}\; \psi_2(\boldsymbol{x}), \ldots\right) \tag{3.81}$$

Note that $\boldsymbol{\Phi}(\boldsymbol{x}) \in \ell_2$, 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 with 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 is 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. (3.42) 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. (3.42 is now

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \alpha_i \, y^i \, \alpha_j \, y^j \, k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; - \; \sum_{i=1}^{l} \alpha_i \\ &\text{s.t. } 0 \; \leq \; \alpha_i \; \leq \; C \\ &\qquad \sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \, . \end{aligned} \tag{3.82}$$

The discriminant function from eq. (3.18) can be formulated using a kernel as

$$\boldsymbol{w}^T \boldsymbol{x} \; + b \; = \; \sum_{i=1}^{l} \left(\alpha_i \, y^i \, k(\boldsymbol{x}^i, \boldsymbol{x}) \right) \; + \; b \, . \tag{3.83}$$

The most popular kernels are the following:

linear kernel:
$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; = \; \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \tag{3.84}$$

RBF kernel:
$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; = \; \exp\left(- \frac{\|\boldsymbol{x}^i \; - \; \boldsymbol{x}^j\|^2}{2 \, \sigma^2} \right) \tag{3.85}$$

polynomial kernel:
$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; = \; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \; + \; \beta \right)^{\alpha} \tag{3.86}$$

sigmoid kernel:
$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; = \; \tanh\left(\alpha \; \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \; + \; \beta \right) \tag{3.87}$$

Comments:

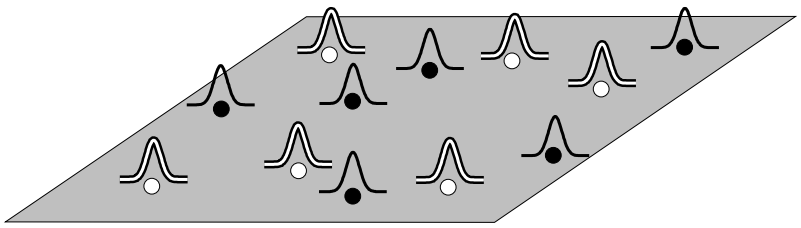


Figure 3.12: An SVM example with RBF kernels. If the width of the kernel is small enough an arbitrary number of training examples can be separated, therefore the VC-dimension is infinity.

- 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. 3.12 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. 3.15 shows again the example from Fig. 3.6 but now with a polynomial kernel of degree 3. The non-linear separable problem can be separated.

The Figures 3.17, 3.18, and 3.19 give SVM examples for the RBF kernel and for the polynomial kernels.

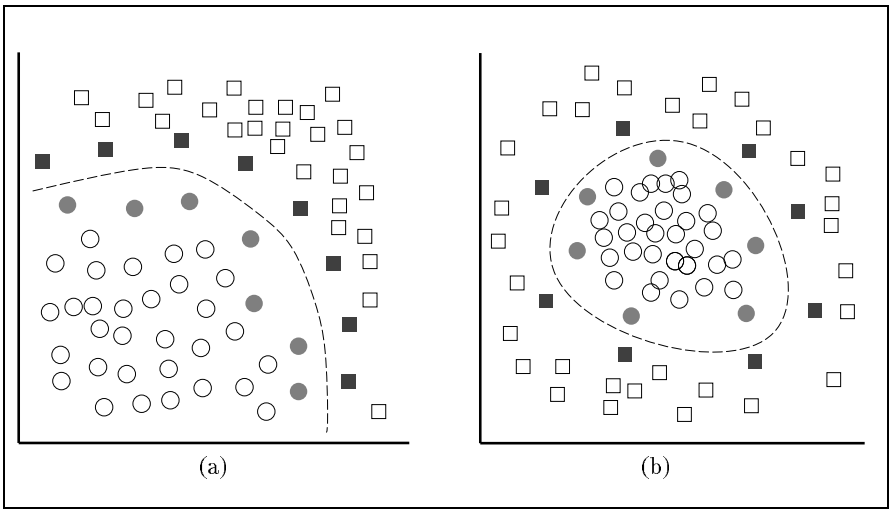


Figure 3.13: Left: An SVM with a polynomial kernel. Right: An SVM with an RBF kernel. Filled points are support vectors. Copyright © 1997 [Osuna et al., 1997].

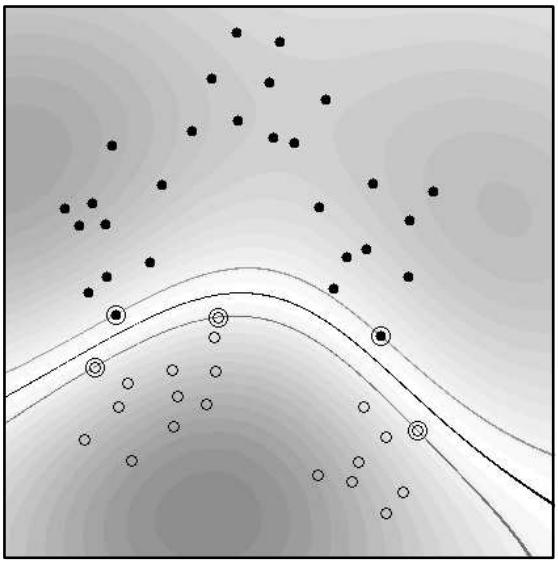


Figure 3.14: SVM classification with an RBF kernel. Support vectors are circled. Copyright ©2002 [Schölkopf and Smola, 2002] MIT Press.

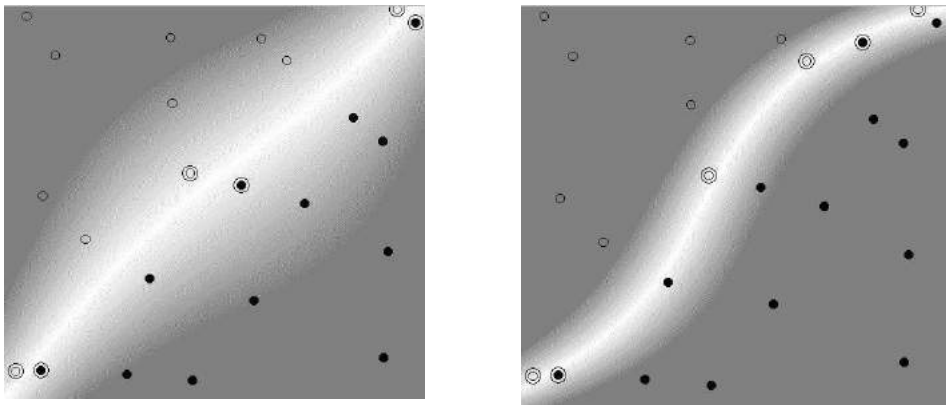


Figure 3.15: The example from Fig. 3.6 but now with polynomial kernel of degree 3. The right task can now be separated. Copyright © 1998 [Burges, 1998].

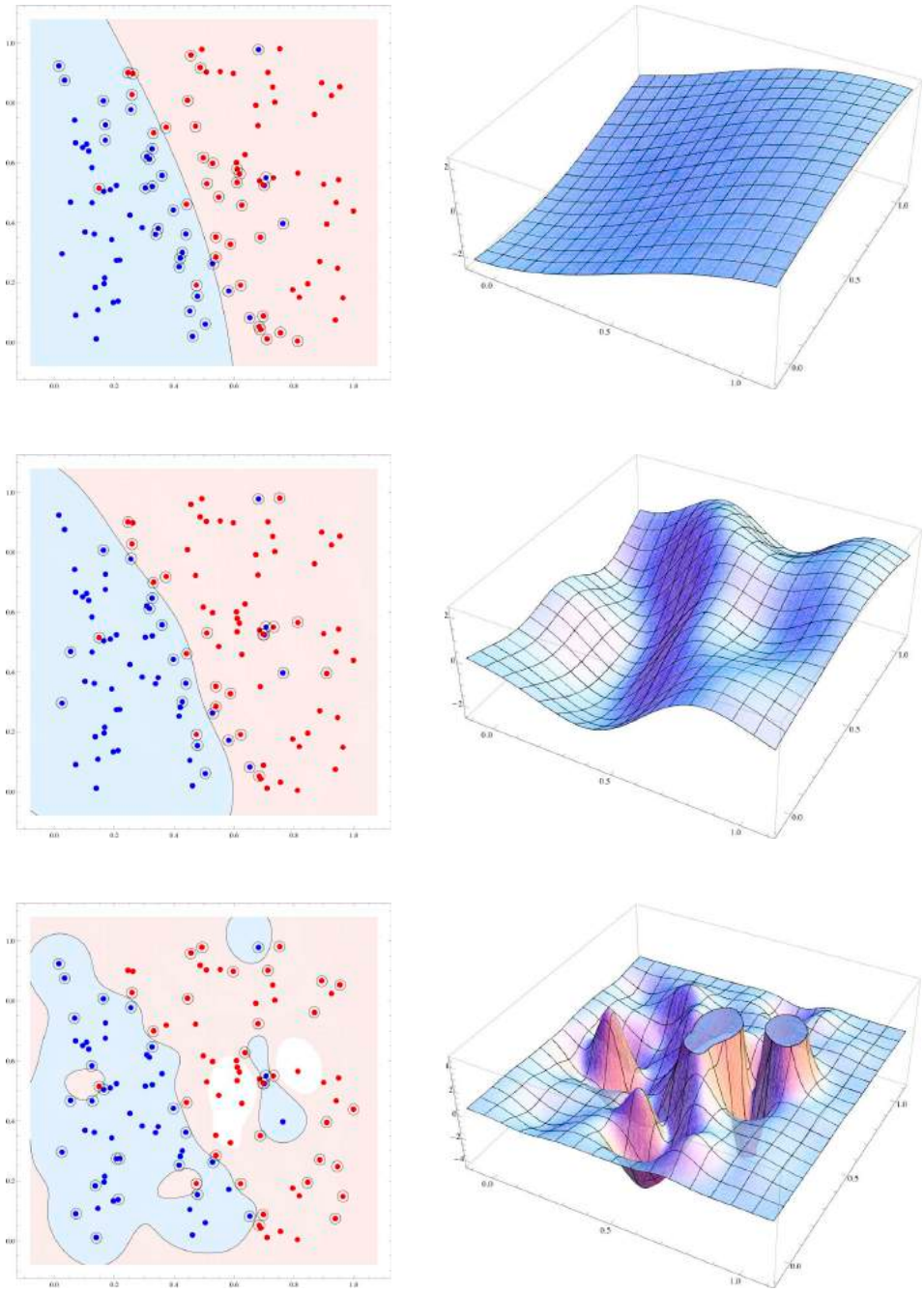


Figure 3.16: SVM with RBF-kernel for different parameter settings. Left: classified datapoints with classification border and areas of the classes. Right: corresponding $g(\boldsymbol{x};\boldsymbol{w})$. The classification border on the left side is the intersection of $g(\boldsymbol{x};\boldsymbol{w})$ with the zero plane on the right side. From the upper to the lower row following parameter pairs were used: $C = 1, \sigma = 0.71; C = 10, \sigma = 0.22; C = 100, \sigma = 0.071$. The class of points is given by their color. The circled points are support vectors and the color of the regions corresponds to the class points in this region would be classified. With increasing cost the SVM tries to avoid classification errors more. With increasing σ $g(\boldsymbol{x};\boldsymbol{w})$ looses its flatness and becomes more spiky which leads to a more complex classification border.

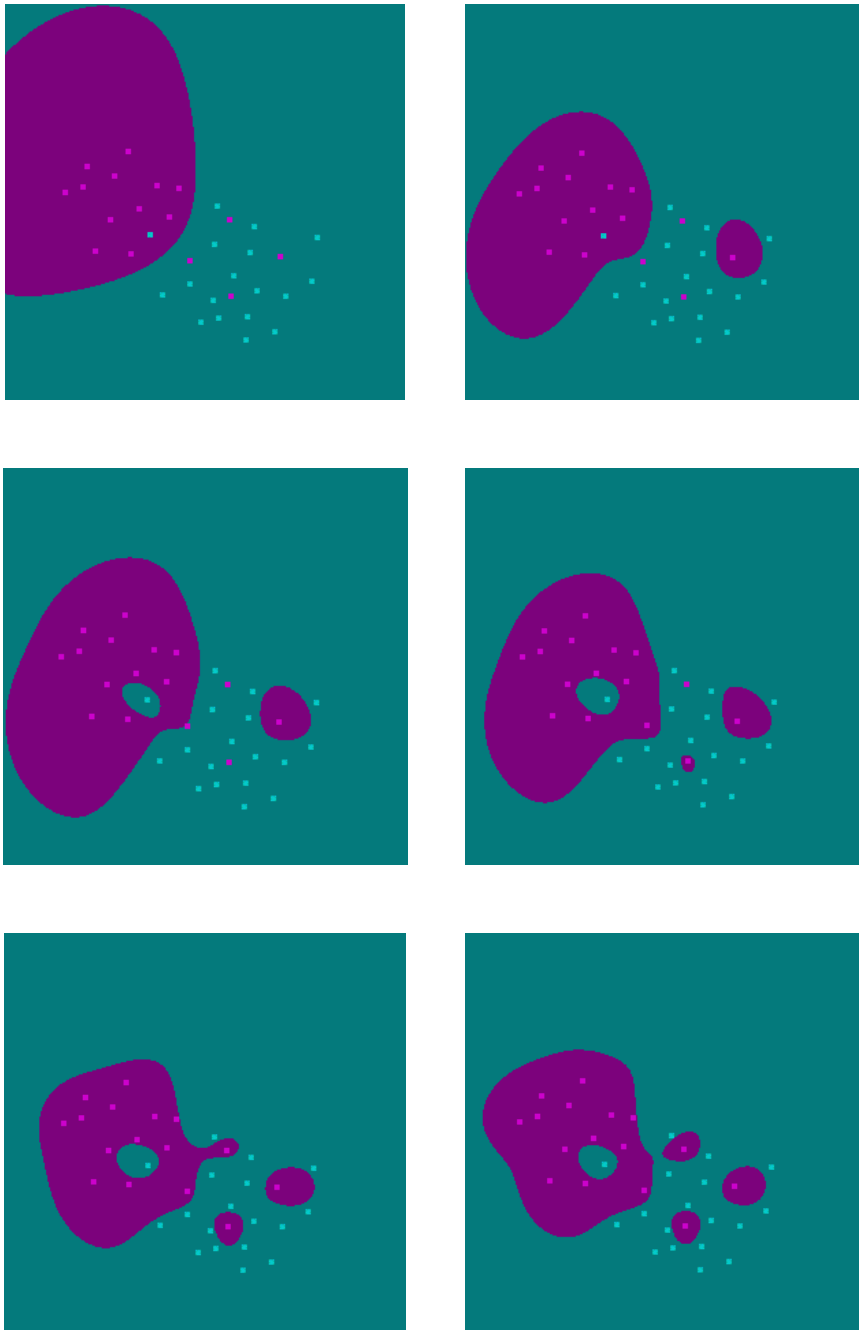


Figure 3.17: SVM with RBF kernel with different σ. From the upper left to the lower right $\sigma = 0.3, 0.18, 0.16, 0.14, 0.12, 0.1$. Screen shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

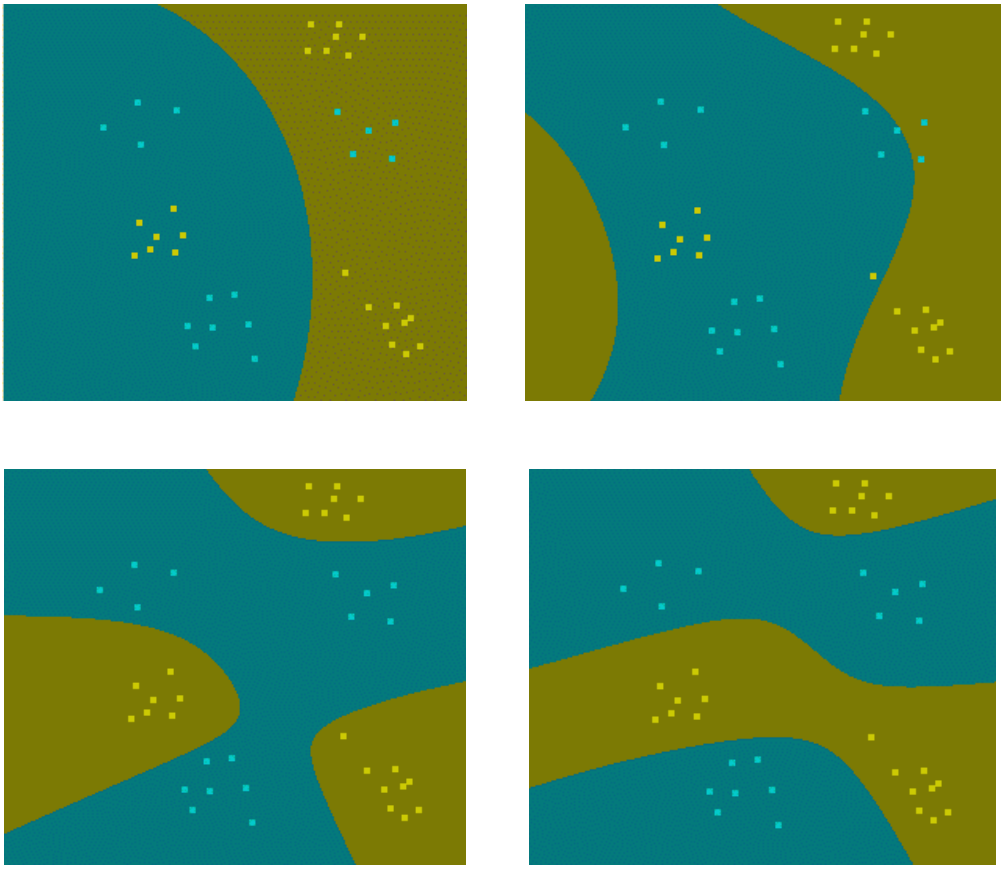


Figure 3.18: SVM with polynomial kernel with different degrees α. From the upper left to the lower right $\alpha = 2, 3, 4, 8$. Screen-shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

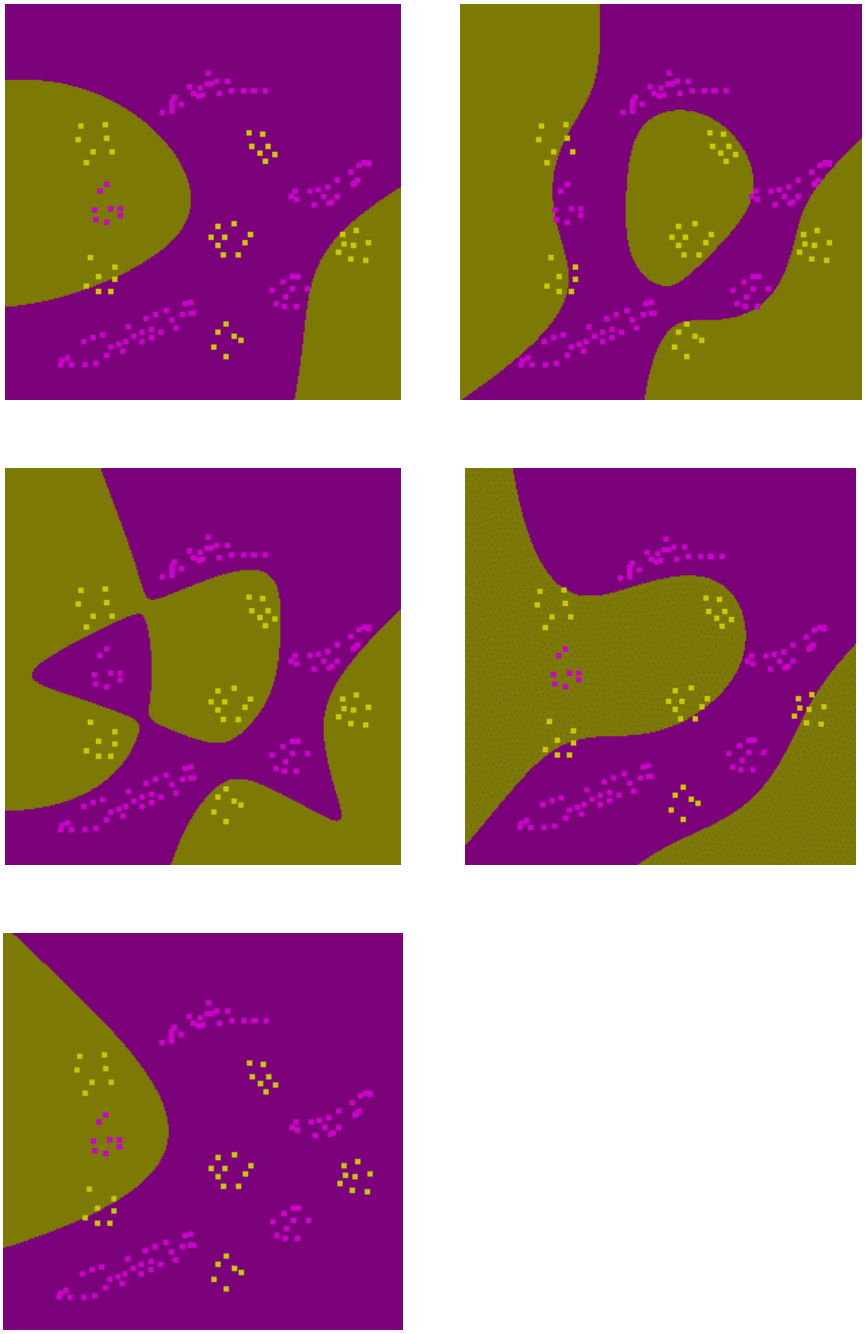


Figure 3.19: SVM with polynomial kernel with degrees $\alpha = 4$ (upper left) and $\alpha = 8$ (upper right) and with RBF kernel with $\sigma = 0.3, 0.6, 1.0$ (from left middle to the bottom). Screen-shots from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

3.7 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 for 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 way 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.
- Classification of the pattern using the SVM.
- If the SVM predicts a face, drawing a rectangle around the face in the output image.

Fig. 3.20 depicts how faces should be separated from non-faces using an SVM. More non-faces are used as support vectors because they form the majority class.

Figures 3.21, 3.22, 3.23, 3.24, 3.25, and 3.26 show the result of the face extraction experiment. Figures 3.25 and 3.26 show more difficult tasks where people are not looking into the camera and false positives and false negatives are present.

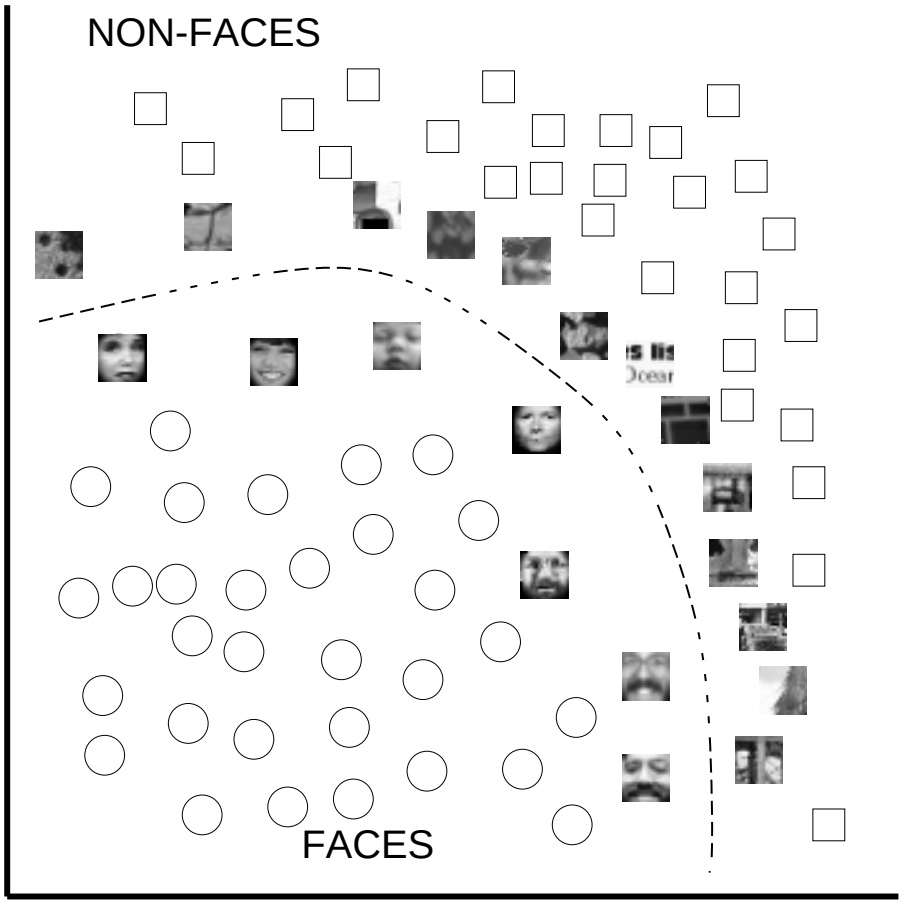


Figure 3.20: Face recognition example. A visualization how the SVM separates faces from non-faces. Support vectors are images of certain faces and images of which are close to face images. The fact that most support vectors are non-faces is depicted in the figure. Copyright © 1997 [Osuna et al., 1997].

Figure 3.21: 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 3.22: Face recognition example. Faces are extracted from an image of the German soccer team and two lab images. Copyright © 1997 [Osuna et al., 1997].

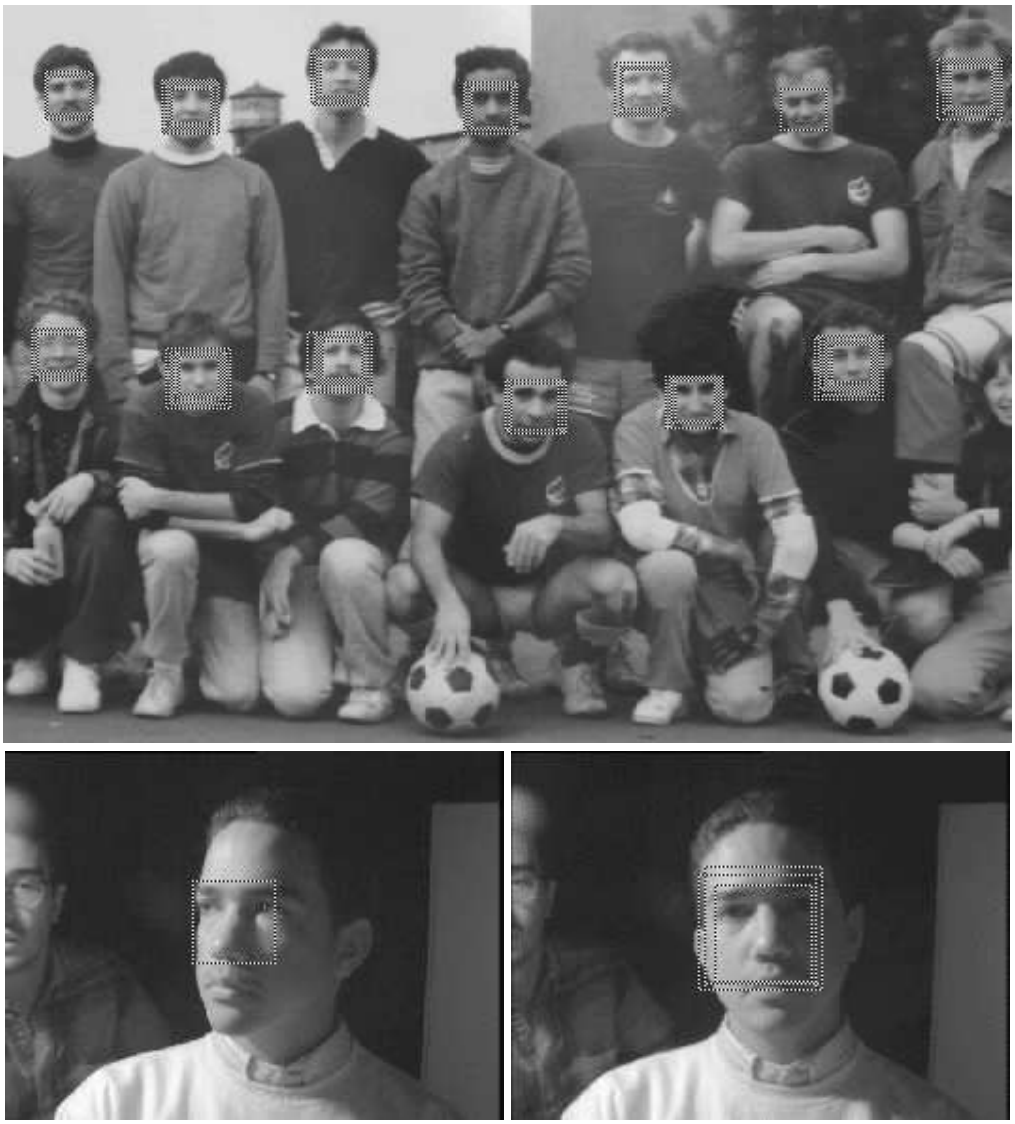


Figure 3.23: Face recognition example. Faces are extracted from another image of a soccer team and two images with lab members. Copyright ©1997 [Osuna et al., 1997].

Figure 3.24: Face recognition example. Faces are extracted from different views and different expressions. Copyright © 1997 [Osuna et al., 1997].

Figure 3.25: 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].

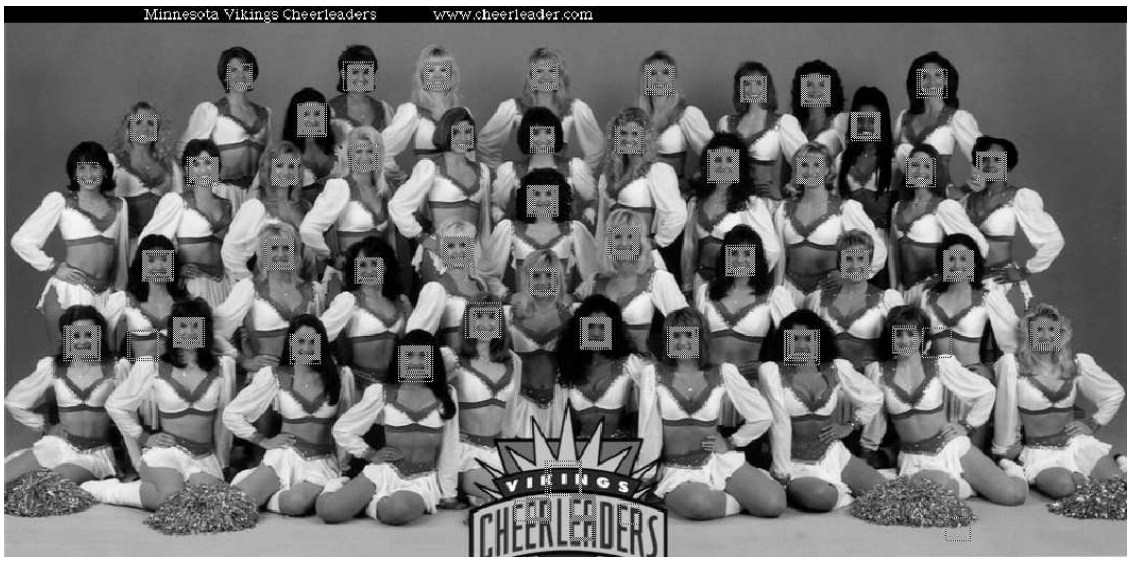


Figure 3.26: Face recognition example. Faces are extracted from a photo of cheerleaders. Here false positives are present. Copyright © 1997 [Osuna et al., 1997].

3.8 Multi-Class SVM

The theory of SVMs has been developed for two classes, i.e. for binary classification. However in real world applications often more than two classes are available. For example in classifying of proteins 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

$$\arg \max_{j=1,\ldots,M} g_j(\boldsymbol{x}) \;=\; \arg \max_{j=1,\ldots,M} \sum_{i=1}^{l} y_j^i \, \alpha_{ij} \, k\left(\boldsymbol{x}, \boldsymbol{x}^i\right) \;+\; b_j \;, \tag{3.88}$$

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 of these classes and the decision is determined by the pairwise classifier of these two classes.

This is 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

$$\begin{aligned} \min_{\boldsymbol{w}_j, b_j, \boldsymbol{\xi}_j} \; & \frac{1}{2} \sum_{j=1}^{M} \|\boldsymbol{w}_j\|^2 \; + \; C \sum_{i=1}^{l} \sum_{j \neq y_i} \xi_{ij} \\ \text{s.t. } & \boldsymbol{w}_{y_i}^T \boldsymbol{x}^i \; + \; b_{y_i} \; \geq \; \boldsymbol{w}_j^T \boldsymbol{x}^i \; + \; b_j \; + \; 2 \; - \; \xi_{ij} \\ & \xi_{ij} \; \geq \; 0 \, , \end{aligned} \tag{3.89}$$

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 of 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.

3.9 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 be 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 if 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.

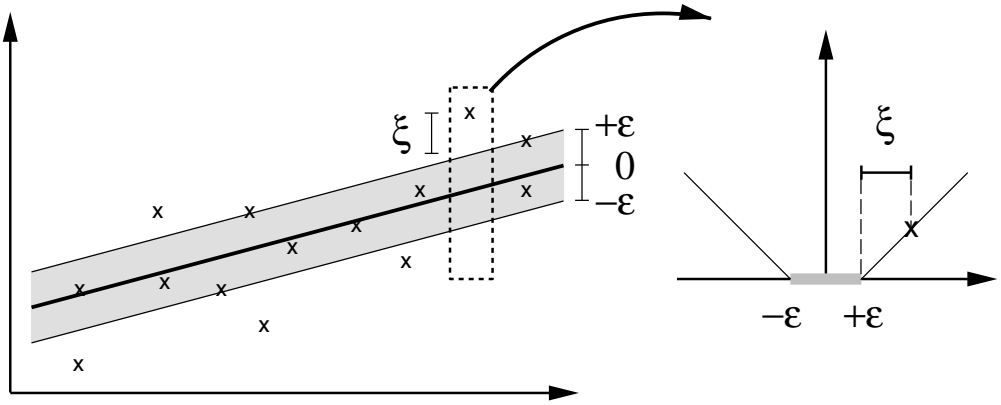


Figure 3.27: Support vector regression. A regression function is found such that a tube with radius ϵ around the regression function contains most data. Points outside the tube are penalized by ξ. Those penalties are traded off against the complexity given by $\|\boldsymbol{w}\|$, which is represented by the slope of a linear function. Copyright © 2002 [Schölkopf and Smola, 2002] MIT Press.

Therefore regression can be transformed into a classification task with $\epsilon > 0$ precision. Reducing the precision allows to describe the regression function with few vectors because all zero-loss 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

$$|y - g(\boldsymbol{x})|_\epsilon = \max\{0\,,\, |y - g(x)| - \epsilon\}\,, \tag{3.90}$$

where $\epsilon > 0$. This loss function was introduced by [Vapnik, 1995]. Any data point lying closer to the regression hyperplane than ϵ does not contribute to the loss. The generally used term ϵ-tube is slightly irritating because the band around the regression function depicted in Fig. 3.27 is actually the space between the two hyperplanes below and above the regression hyperplane.

If we now consider again linear functions $\boldsymbol{w}^T\boldsymbol{x} + b$ then we obtain the situation depicted in Fig. 3.27. 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

$$\begin{aligned} \xi^- &= \max\{0\,,\, g(x) - y - \epsilon\} \\ \xi^+ &= \max\{0\,,\, y - g(x) - \epsilon\} \\ \xi &= \left(\xi^- + \xi^+\right) \end{aligned} \tag{3.91}$$

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.

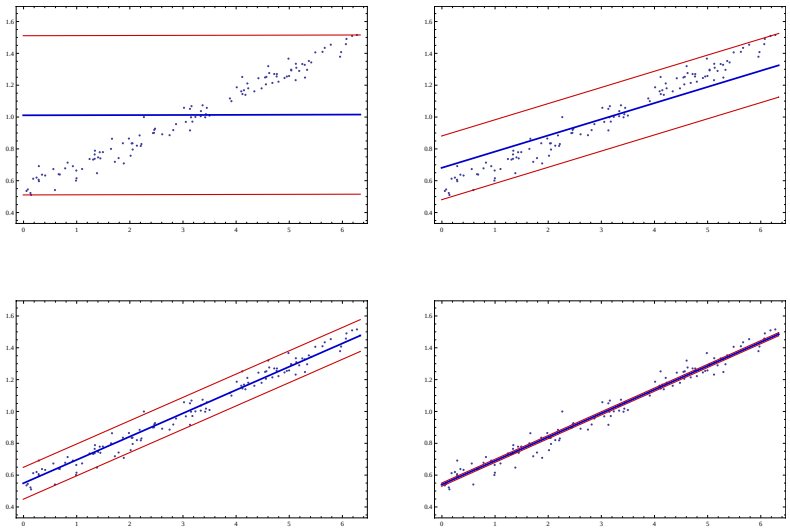


Figure 3.28: Linear support vector regression with different ϵ settings. From the upper left to the lower right $\epsilon = 0.5, 0.2, 0.1, 0.01$. C was set to 1. Support vector regression fits the flattest possible function for the given ϵ-value.

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

$$\left\|\boldsymbol{w}^T\boldsymbol{x}\right\| \leq \|\boldsymbol{w}\| \, \|\boldsymbol{x}\| \leq \|\boldsymbol{w}\| \, R \, . \tag{3.92}$$

This holds also for the kernel version which means for $g(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i \, k(\boldsymbol{x}, \boldsymbol{x}^i) + b$:

$$\|g(\boldsymbol{x}) - b\| \leq \|\boldsymbol{w}\| \, R \leq \sqrt{\sum_{i,j} \alpha_i \, \alpha_j \, k(\boldsymbol{x}^i, \boldsymbol{x}^j)} \, R \leq \tag{3.93}$$
$$\|\boldsymbol{\alpha}\| \, \sqrt{e_{\max}} \, R \, ,$$

where $e_{\max}$ is the maximal eigenvalue of the kernel matrix, i.e. the Gram matrix.

We obtain as optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}^+,\boldsymbol{\xi}^-} \; & \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{l} \sum_{i=1}^{l} \left(\xi_i^+ + \xi_i^-\right) \\ \text{s.t. } & y^i - \left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) \leq \epsilon + \xi_i^+ \\ & \left(\boldsymbol{w}^T\boldsymbol{x}^i + b\right) - y^i \leq \epsilon + \xi_i^- \\ & \xi_i^+ \geq 0 \text{ and } \xi_i^- \geq 0 \, . \end{aligned} \tag{3.94}$$

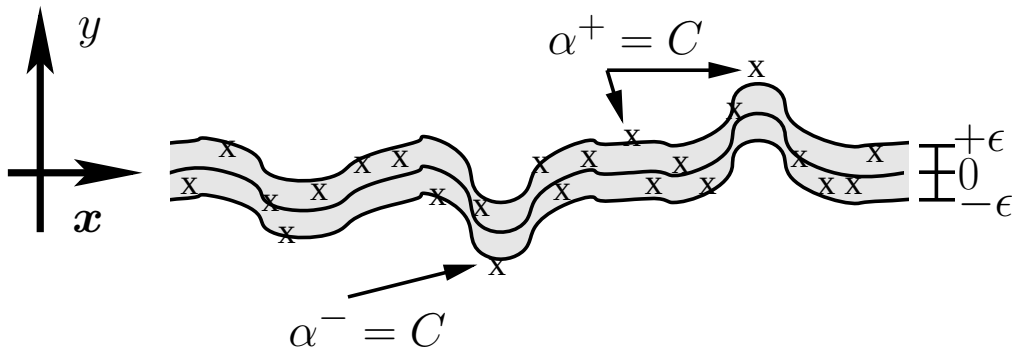


Figure 3.29: Nonlinear support vector regression is depicted. A regression function is found such that a tube with radius ϵ around the regression function contains most data.

The Lagrangian is

$$\begin{aligned}
&\mathcal{L}\left(\boldsymbol{w}, b, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^-\right) = \\
&\frac{1}{2}\left\|\boldsymbol{w}\right\|^2 + \frac{C}{l}\sum_{i=1}^{l}\left(\xi_i^+ + \xi_i^-\right) - \sum_{i=1}^{l}\left(\mu_i^+\,\xi_i^+ + \mu_i^-\,\xi_i^-\right) - \\
&\sum_{i=1}^{l}\alpha_i^+\left(\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T\boldsymbol{x}^i + b\right) - \\
&\sum_{i=1}^{l}\alpha_i^-\left(\epsilon + \xi_i^- + y^i - \boldsymbol{w}^T\boldsymbol{x}^i - b\right),
\end{aligned} \tag{3.95}$$

where $\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^- \geq \mathbf{0}$.

The saddle point conditions require

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial b} &= \sum_{i=1}^{l}\left(\alpha_i^+ - \alpha_i^-\right) = 0 \\
\frac{\partial\mathcal{L}}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{l}\left(\alpha_i^+ - \alpha_i^-\right)\boldsymbol{x}^i = \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial \boldsymbol{\xi}^+} &= \frac{C}{l}\mathbf{1} - \boldsymbol{\alpha}^+ - \boldsymbol{\mu}^+ = \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial \boldsymbol{\xi}^-} &= \frac{C}{l}\mathbf{1} - \boldsymbol{\alpha}^- - \boldsymbol{\mu}^- = \mathbf{0}
\end{aligned} \tag{3.96}$$

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

$$\begin{aligned}\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \frac{1}{2} \sum_{i,j=(1,1)}^{(l,l)} \left(\alpha_i^+ - \alpha_i^-\right) \left(\alpha_j^+ - \alpha_j^-\right) \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j + \\ \epsilon \sum_{i=1}^{l} \left(\alpha_i^+ + \alpha_i^-\right) - \sum_{i=1}^{l} y^i \left(\alpha_i^+ - \alpha_i^-\right) \\ \text{s.t. } 0 \leq \alpha_i^+ \leq \frac{C}{l} \\ 0 \leq \alpha_i^- \leq \frac{C}{l} \\ \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) = 0 \,.\end{aligned} \tag{3.97}$$

The regression function can be written as

$$g(\boldsymbol{x}) = \sum_{i=1}^{l} \left(\alpha_i^+ - \alpha_i^-\right) \left(\boldsymbol{x}^i\right)^T \boldsymbol{x} + b \,. \tag{3.98}$$

The KKT conditions state for the optimal parameters:

$$\begin{aligned}\alpha_i^+ \left(\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T \boldsymbol{x}^i + b\right) &= 0 \\ \alpha_i^- \left(\epsilon + \xi_i^- + y^i - \boldsymbol{w}^T \boldsymbol{x}^i - b\right) &= 0 \\ \xi_i^+ \mu_i^+ &= 0 \\ \xi_i^- \mu_i^- &= 0 \,.\end{aligned} \tag{3.99}$$

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

$$\begin{aligned}\xi_i^+ \left(\frac{C}{l} - \alpha_i^+\right) &= 0 \\ \xi_i^- \left(\frac{C}{l} - \alpha_i^-\right) &= 0 \,.\end{aligned} \tag{3.100}$$

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

$$b = y^i - \boldsymbol{w}^T \boldsymbol{x}^i - \epsilon \,. \tag{3.101}$$

And for $0 < \alpha_i^- < \frac{C}{l}$ the value b can be computed through

$$b = y^i - \boldsymbol{w}^T \boldsymbol{x}^i + \epsilon \,. \tag{3.102}$$

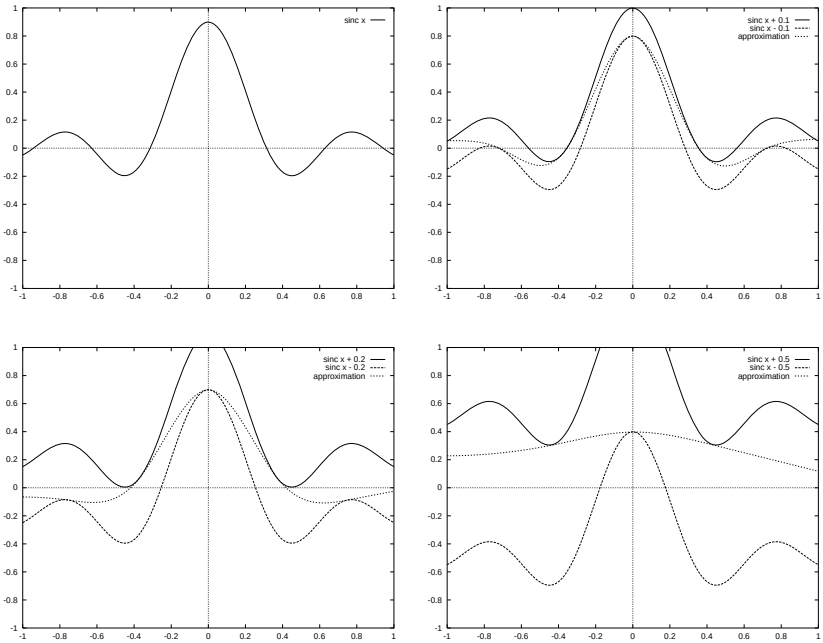


Figure 3.30: Example of SV regression: smoothness effect of different ϵ. Upper left: the target function $\mathrm{sinc}x$, upper right: SV regression with $\epsilon = 0.1$, lower left: $\epsilon = 0.2$, and lower right: $\epsilon = 0.5$. Larger ϵ leads to a smoother approximation function which fits into the broader ϵ-tube.

If $\alpha_i^+ > 0$ and $\alpha_i^- > 0$, then

$$\begin{aligned} \epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T \boldsymbol{x}^i + b = 0 \text{ and} \\ \epsilon + \xi_i^- + y^i - \boldsymbol{w}^T \boldsymbol{x}^i - b = 0 \,. \end{aligned} \tag{3.103}$$

and adding both equations gives

$$2\,\epsilon + \xi_i^+ + \xi_i^- = 0 \tag{3.104}$$

which contradicts $\epsilon > 0$.

Therefore we have

$$\alpha_i^+ \, \alpha_i^- = 0 \,. \tag{3.105}$$

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

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j=(1,1)}^{(l,l)} \alpha_i \, \alpha_j \, \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j - \\ \epsilon \, \|\boldsymbol{\alpha}\|_1 + \sum_{i=1}^{l} y^i \, \alpha_i \\ \text{s.t. } -\frac{C}{l} \leq \alpha_i \leq \frac{C}{l} \\ \sum_{i=1}^{l} \alpha_i = 0 \,. \end{aligned} \tag{3.106}$$

or in vector notation

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \frac{1}{2} \, \boldsymbol{\alpha} \boldsymbol{X}^T \boldsymbol{X} \, \boldsymbol{\alpha} - \epsilon \, \|\boldsymbol{\alpha}\|_1 + \boldsymbol{y}^T \boldsymbol{\alpha} \\ \text{s.t. } -\frac{C}{l} \mathbf{1} \leq \boldsymbol{\alpha} \leq \frac{C}{l} \mathbf{1}, \; 1^T \boldsymbol{\alpha} = 0 \,. \end{aligned} \tag{3.107}$$

An example for support vector regression is given in Fig. 3.30. Larger ϵ leads to a smoother approximation function which fits into the broader ϵ-tube. Fig. 3.31 shows that decreasing ϵ results in an increase of the number of support vectors. As depicted in Fig. 3.32 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.

We obtain as optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}^+,\boldsymbol{\xi}^-,\epsilon} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \left(\nu \, \epsilon + \frac{1}{l} \sum_{i=1}^{l} \left(\xi_i^+ + \xi_i^-\right) \right) \\ \text{s.t. } y^i - \left(\boldsymbol{w}^T \boldsymbol{x}^i + b\right) \leq \epsilon + \xi_i^+ \\ \left(\boldsymbol{w}^T \boldsymbol{x}^i + b\right) - y^i \leq \epsilon + \xi_i^- \\ \xi_i^+ \geq 0, \; \xi_i^- \geq 0 \text{ and } \epsilon \geq 0 \,. \end{aligned} \tag{3.108}$$

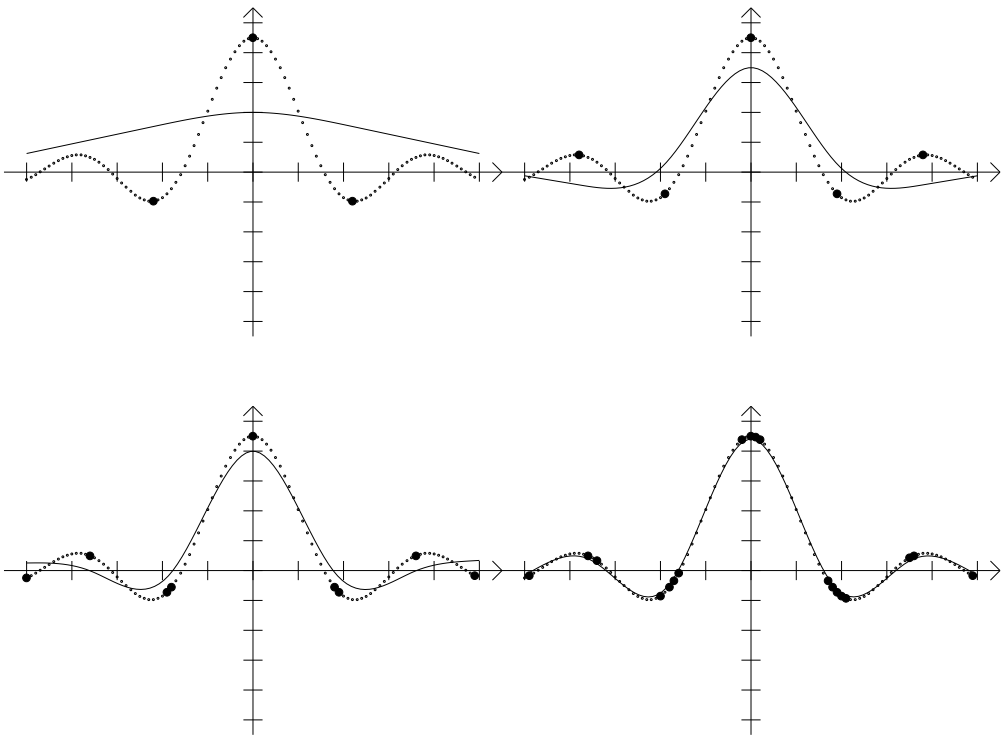


Figure 3.31: Example of SV regression: support vectors for different ϵ. Example from Fig. 3.30. The tiny dots are the data points and the big dots are support vectors. Upper left: $\epsilon = 0.5$, upper right: $\epsilon = 0.2$, lower left: $\epsilon = 0.1$, and lower right: $\epsilon = 0.002$. The number 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].

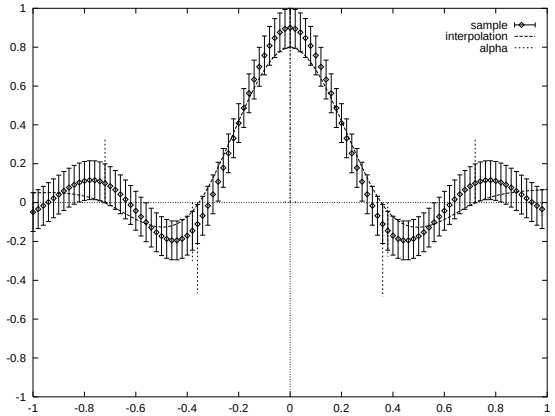


Figure 3.32: Example of SV regression: support vectors pull the approximation curve inside the ϵ-tube. Example from Fig. 3.30. Dash-dotted vertical lines indicate where the approximation curve is pulled inside the ϵ-tube. At these positions support vectors are located. Copyright © 2002 [Smola and Schölkopf, 2002].

The Lagrangian is

$$\begin{aligned}
&\mathcal{L}\left(\boldsymbol{w}, b, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^-, \epsilon, \beta\right) \; = \\
&\frac{1}{2}\,\|\boldsymbol{w}\|^2 \; + \; C\,\nu\,\epsilon \; + \; \frac{C}{l}\sum_{i=1}^{l}\left(\xi_i^+ \; + \; \xi_i^-\right) \; - \; \beta\,\epsilon \; - \\
&\sum_{i=1}^{l}\left(\boldsymbol{\mu}_i^+\,\xi_i^+ \; + \; \boldsymbol{\mu}_i^-\,\xi_i^-\right) \; - \\
&\sum_{i=1}^{l}\alpha_i^+\left(\epsilon \; + \; \xi_i^+ \; - \; y^i \; + \; \boldsymbol{w}^T\boldsymbol{x}^i \; + \; b\right) \; - \\
&\sum_{i=1}^{l}\alpha_i^-\left(\epsilon \; + \; \xi_i^- \; + \; y^i \; - \; \boldsymbol{w}^T\boldsymbol{x}^i \; - \; b\right) \; ,
\end{aligned} \tag{3.109}$$

where $\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^- \geq \mathbf{0}$.

The saddle point conditions require

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial b} \; &= \; \sum_{i=1}^{l}\left(\alpha_i^+ \; + \; \alpha_i^-\right) \; = \; 0 \\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} \; &= \; \boldsymbol{w} \; - \; \sum_{i=1}^{l}\left(\alpha_i^+ \; - \; \alpha_i^-\right)\,\boldsymbol{x}^i \; = \; \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}^+} \; &= \; \frac{C}{l}\,\mathbf{1} \; - \; \boldsymbol{\alpha}^+ \; - \; \boldsymbol{\mu}^+ \; = \; \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}^-} \; &= \; \frac{C}{l}\,\mathbf{1} \; - \; \boldsymbol{\alpha}^- \; - \; \boldsymbol{\mu}^- \; = \; \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial\epsilon} \; &= \; C\,\nu \; - \; \sum_{i=1}^{l}\left(\boldsymbol{\alpha}^+ \; + \; \boldsymbol{\alpha}^-\right) \; - \; \beta \; = \; 0\,.
\end{aligned} \tag{3.110}$$

We obtain as dual formulation

$$\begin{aligned}
\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-}\frac{1}{2}\sum_{i,j=(1,1)}^{(l,l)}\left(\alpha_i^+ \; - \; \alpha_i^-\right)\,\left(\alpha_j^+ \; - \; \alpha_j^-\right)\,\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \; - \\
\sum_{i=1}^{l}y^i\,\left(\alpha_i^+ \; - \; \alpha_i^-\right) \\
\text{s.t. } 0 \; \leq \; \alpha_i^+ \; \leq \; \frac{C}{l} \\
0 \; \leq \; \alpha_i^- \; \leq \; \frac{C}{l} \\
\sum_{i=1}^{l}\left(\alpha_i^+ \; - \; \alpha_i^-\right) \; = \; 0 \\
\sum_{i=1}^{l}\left(\alpha_i^+ \; + \; \alpha_i^-\right) \; \leq \; C\,\nu\,.
\end{aligned} \tag{3.111}$$

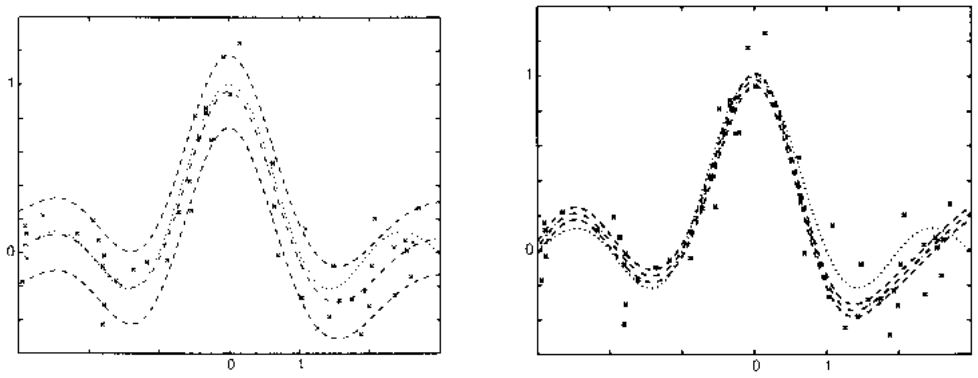


Figure 3.33: ν-SV regression with $\nu = 0.2$ (left) and $\nu = 0.8$ (right). ϵ is automatically adjusted to $\epsilon = 0.22$ (left) and $\epsilon = 0.04$ (right). Large ν allows more points to be outside the tube. Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

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

$$\epsilon + \xi_i^+ - y^i + \boldsymbol{w}^T \boldsymbol{x}^i + b = 0 \tag{3.112}$$

$$\epsilon + \xi_j^- + y^j - \boldsymbol{w}^T \boldsymbol{x}^j - b = 0\,. \tag{3.113}$$

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 3.33, 3.34, and 3.35 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

$$\begin{cases} \frac{1}{2\,\sigma}\left(y - g(x)\right)^2 & \text{if} \quad |y - g(x)| < \sigma \\ |y - g(x)| - \frac{\sigma}{2} & \text{otherwise} \end{cases} \tag{3.114}$$

which is quadratic around zero and linear (Laplacian) elsewhere.

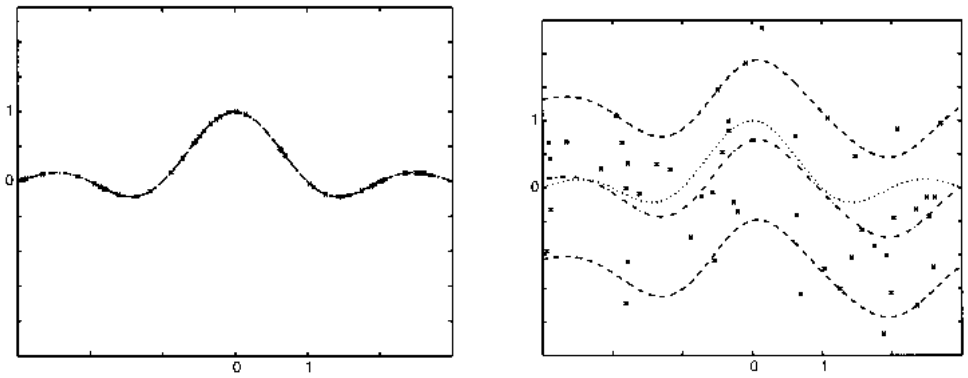


Figure 3.34: ν-SV regression where ϵ is automatically adjusted to the noise level. Noise level $\sigma = 0$ (left) and $\sigma = 1.0$ lead to $\epsilon = 0$ (left) and $\epsilon = 1.19$ (right), respectively. In both cases the same parameter $\nu = 0.2$ was used. Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

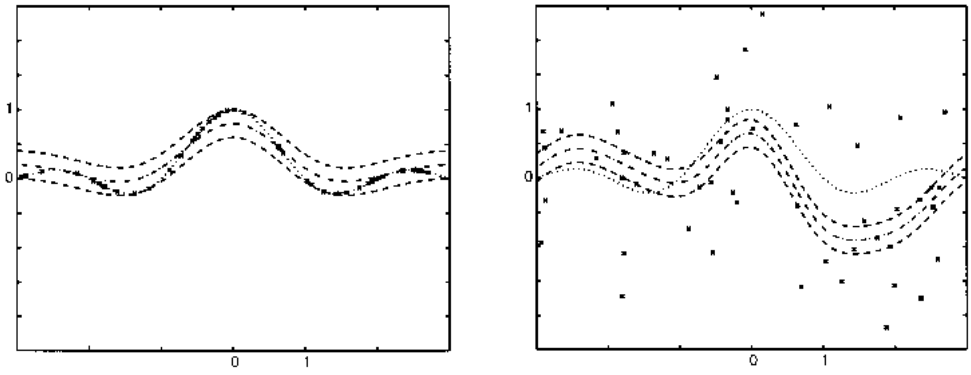


Figure 3.35: Standard SV regression with the example from Fig. 3.34 (noise levels: left $\sigma = 0$ and right $\sigma = 1.0$). $\epsilon = 0.2$ was fixed which is neither optimal for the low noise case (left) nor for the high noise case (right). Copyright © 2000 MIT Press Journals [Schölkopf et al., 2000].

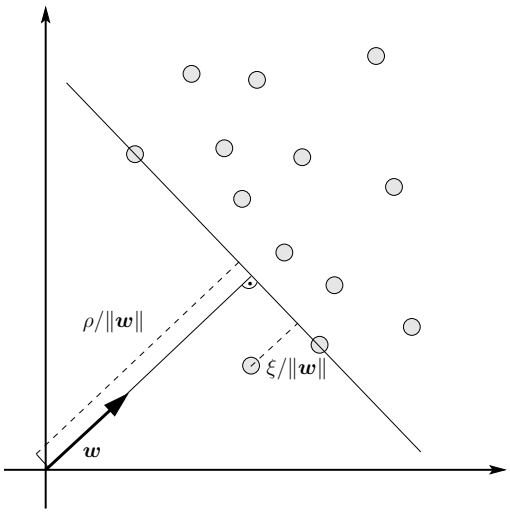


Figure 3.36: The idea of the one-class SVM is depicted. The data points are separated from the origin with the hyperplane which has the largest margin.

3.10 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. 3.36.

We now apply the ν-SVM formulation from eq. (3.48) to this problem:

$$\begin{aligned} \min_{\tilde{\boldsymbol{w}},\tilde{b},\tilde{\boldsymbol{\xi}},\tilde{\rho}} \; & \frac{1}{2}\,\|\tilde{\boldsymbol{w}}\|^2 \;-\; \nu\,\tilde{\rho} \;+\; \frac{1}{l}\,\sum_{i=1}^{l}\tilde{\xi}_i \\ \text{s.t. } & y^i\,\left(\tilde{\boldsymbol{w}}^T\boldsymbol{x}^i \;+\; \tilde{b}\right) \;\geq\; \tilde{\rho} \;-\; \tilde{\xi}_i \\ & \tilde{\xi}_i \;\geq\; 0 \;\text{ and }\; \tilde{\rho} \;\geq\; 0\,. \end{aligned} \tag{3.115}$$

First we re-scale the values because the hyperplane is not supposed to be in its canonical form:

$$\tilde{b} \;=\; \nu\,b \tag{3.116}$$

$$\tilde{\boldsymbol{w}} \;=\; \nu\,\boldsymbol{w} \tag{3.117}$$

$$\tilde{\rho} \;=\; \nu\,\rho \tag{3.118}$$

$$\tilde{\xi} \;=\; \nu\,\xi \tag{3.119}$$

and obtain after scaling

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \; & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;-\; \rho \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \\ \text{s.t. } & y^i\,\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; \rho \;-\; \xi_i \\ & \xi_i \;\geq\; 0 \;\text{ and }\; \rho \;\geq\; 0\,. \end{aligned} \tag{3.120}$$

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

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l-1}\xi_i \;+\; b \\ \text{s.t. } & \boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;\geq\; -\,\xi_i,\;\; \xi_i \;\geq\; 0\,. \end{aligned} \tag{3.121}$$

which gives for l positive points and re-scaling of ν the one class support vector optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; & \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \;+\; b \\ \text{s.t. } & \boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;\geq\; -\,\xi_i,\;\; \xi_i \;\geq\; 0\,. \end{aligned} \tag{3.122}$$

The Lagrangian of the one class SVM formulation is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) \;=\; \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{l\,\nu}\,\sum_{i=1}^{l}\xi_i \;+\; b \;- \tag{3.123}$$
$$\sum_{i=1}^{l}\alpha_i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b \;+\; \xi_i\right) \;-\; \sum_{i=1}^{l}\mu_i\,\xi_i\;.$$

Setting the derivatives of $\mathcal{L}$ with respect to $\boldsymbol{w}$,$\boldsymbol{\xi}$, and b equal to zero gives:

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l}\alpha_i\,\boldsymbol{x}^i \tag{3.124}$$
$$\alpha_i \;=\; \frac{1}{l\,\nu} \;-\; \mu_i \;\leq\; \frac{1}{l\,\nu}$$
$$\sum_{i=1}^{l}\alpha_i \;=\; 1\;.$$

The dual problem for the one-class SVM is

$$\min_{\boldsymbol{\alpha}}\frac{1}{2}\;\sum_{i,j}\alpha_i\,\alpha_j\;\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \tag{3.125}$$
$$\text{s.t. } 0 \;\leq\; \alpha_i \;\leq\; \frac{1}{l\,\nu}$$
$$\sum_{i=1}^{l}\alpha_i \;=\; 1\;.$$

The value for b can again be determined from $0 \;<\; \alpha_j \;<\; \frac{1}{l\,\nu}$ as

$$b \;=\; -\;\sum_{i=1}^{l}\alpha_i\;\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \tag{3.126}$$

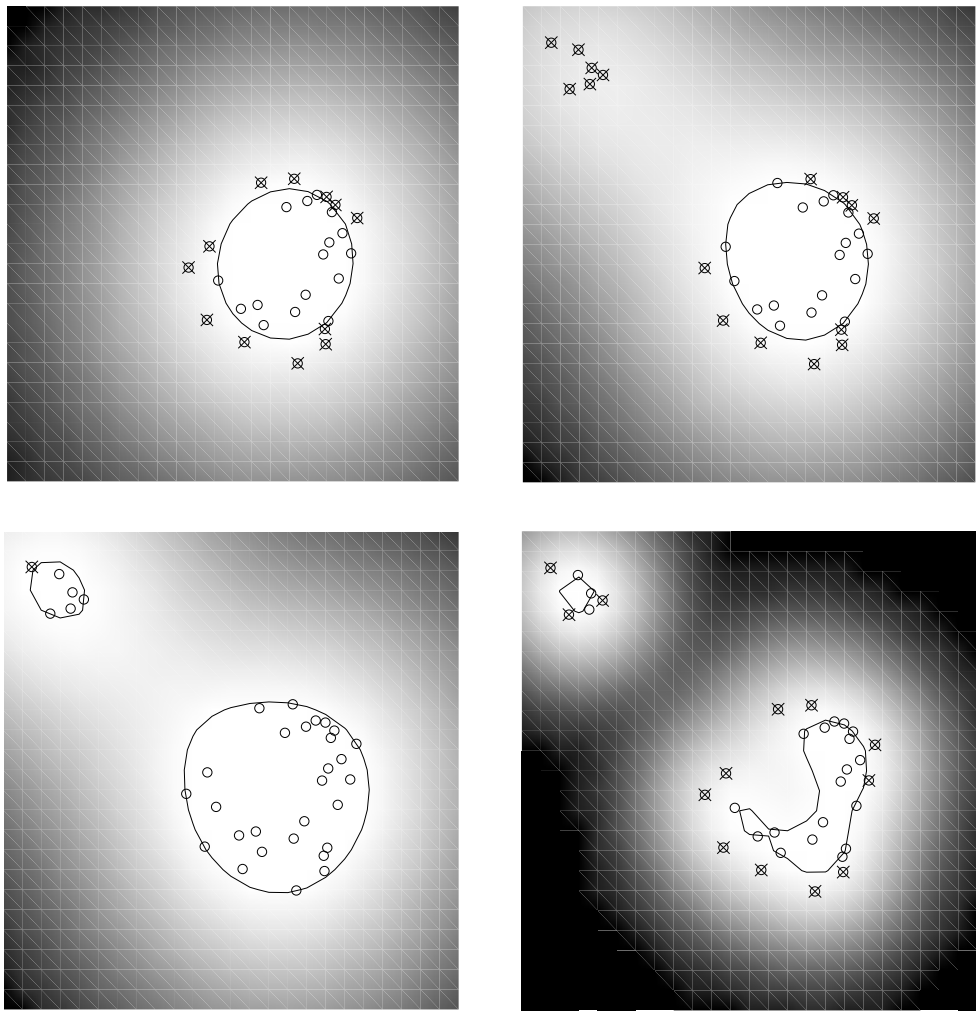
Note for $\nu = 1$ a kernel density estimator is obtained because $\alpha_i = \alpha_j = \frac{1}{l}$ (especially $\nu \leq 1$ must hold).

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 3.37 and 3.38 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 3.37: 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].

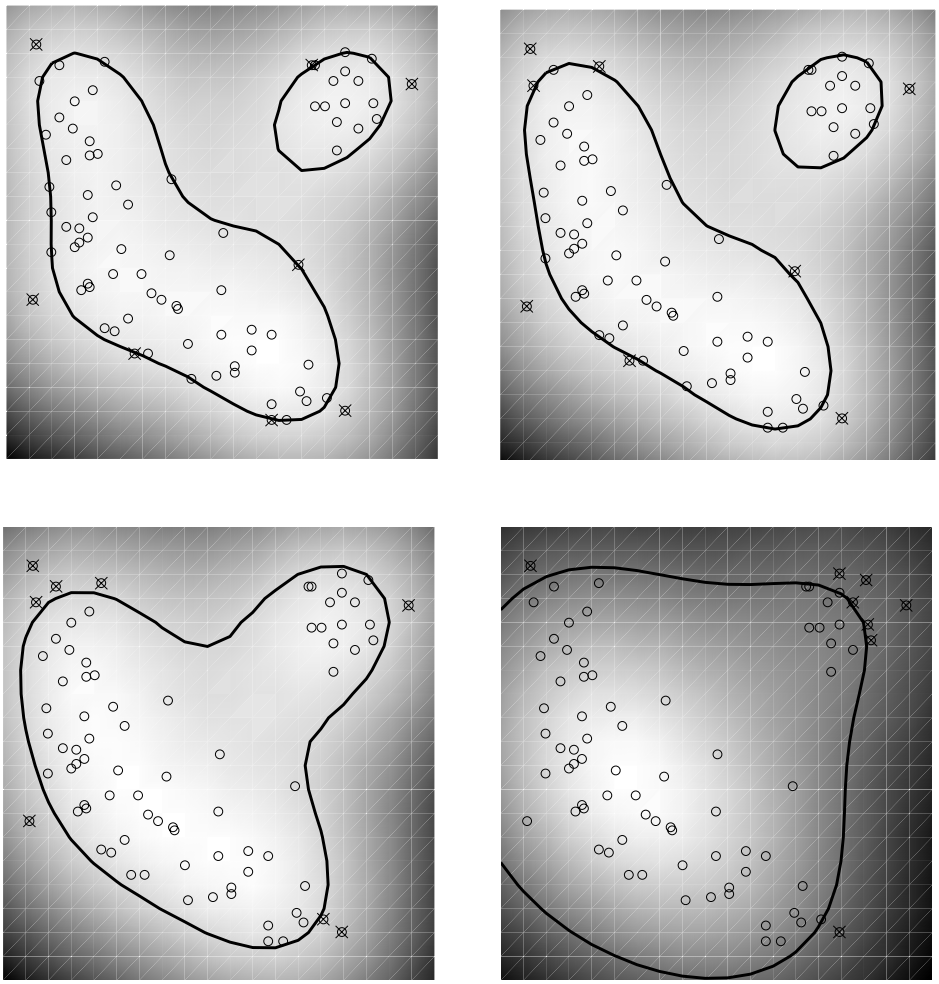


Figure 3.38: A single-class SVM applied to another toy problem with $\sigma = 0.5$ and $\nu \in \{0.1, 0.2, 0.4, 1.0\}$ from upper left to lower right. Note, that $\nu = 1$ (lower right) leads to a kernel density estimate. Copyright © 2001 MIT Press Journals [Schölkopf et al., 2001].

3.11 Least Squares SVM

The C-SVM from eq. (3.32) was

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; & \frac{1}{2} \|\boldsymbol{w}\|^2 \; + \; C \sum_{i=1}^{l} \xi_i \\ \text{s.t. } & y^i \; \left(\boldsymbol{w}^T \boldsymbol{x}^i \; + \; b \right) \; \geq \; 1 \; - \; \xi_i \\ & \xi_i \; \geq \; 0 \, . \end{aligned} \tag{3.127}$$

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.

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; & \frac{1}{2} \|\boldsymbol{w}\|^2 \; + \; C \sum_{i=1}^{l} \xi_i^2 \\ \text{s.t. } & y^i \; \left(\boldsymbol{w}^T \boldsymbol{x}^i \; + \; b \right) \; = \; 1 \; - \; \xi_i \, , \end{aligned} \tag{3.128}$$

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

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} \; = \; 2 \, C \, \boldsymbol{\xi} \; - \; \boldsymbol{\alpha} \; = \; \mathbf{0} \, . \tag{3.129}$$

Therefore

$$\xi_i \; = \; \frac{1}{2 \, C} \, \alpha_i \tag{3.130}$$

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

$$\begin{aligned} \min_{\boldsymbol{\alpha}} & \frac{1}{2} \sum_{i,j} \alpha_i \, y^i \, \alpha_j \, y^j \, \left(\left(\boldsymbol{x}^i \right)^T \boldsymbol{x}^j \; + \; \frac{1}{2 \, C} \delta_{ij} \right) \; - \; \sum_{i=1}^{l} \alpha_i \\ \text{s.t.} & \\ & \sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \, , \end{aligned} \tag{3.131}$$

where δ_{ij} is the Kronecker delta which is 1 for $i = j$ and 0 otherwise.

Again we obtain known facts like

$$\boldsymbol{w} \; = \; \sum_{i=1}^{l} \alpha_i \, y^i \, \boldsymbol{x}^i \, . \tag{3.132}$$

Also the offset b can be regularized which removes the equality constraint from the dual. However the problem is no longer translation invariant.

The primal problem formulation is

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \; \frac{1}{2}\,\|\boldsymbol{w}\|^2 \;+\; \frac{1}{2}\,b^2 \;+\; C\,\sum_{i=1}^{l}\xi_i^2 \\ &\text{s.t. } y^i\;\left(\boldsymbol{w}^T\boldsymbol{x}^i \;+\; b\right) \;\geq\; 1 \;-\; \xi_i\;. \end{aligned} \tag{3.133}$$

The derivative with respect to b is now

$$\frac{\partial\mathcal{L}}{\partial b} \;=\; b \;-\; \sum_{i=1}^{l}\alpha_i\,y^i \;=\; 0 \tag{3.134}$$

$$\tag{3.135}$$

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

$$\begin{aligned} &\min_{\boldsymbol{\alpha}}\frac{1}{2}\;\sum_{i,j}\alpha_i\,y^i\,\alpha_j\,y^j\;\left(\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \;+\; 1 \;+\; \frac{1}{2\,C}\delta_{ij}\right) \;-\; \sum_{i=1}^{l}\alpha_i \\ &\text{s.t. } 0 \;\leq\; \alpha_i\;. \end{aligned} \tag{3.136}$$

The value b can be computed through

$$b \;=\; \sum_{i=1}^{l}\alpha_i\,y^i\;. \tag{3.137}$$

From the KKT conditions it can be inferred that the optimal solution satisfies

$$\boldsymbol{\alpha}^T\left(\boldsymbol{Q}\;\boldsymbol{\alpha}\;\mathbf{1}\right) \;=\; 0\;, \tag{3.138}$$

where

$$Q_{ij} \;=\; y^i\,y^j\;\left(\left(\boldsymbol{x}^i\right)^T\boldsymbol{x}^j \;+\; 1 \;+\; \frac{1}{2\,C}\delta_{ij}\right)\;. \tag{3.139}$$

A linear to the solution converging iteration scheme is

$$\boldsymbol{\alpha}^{t+1} \;=\; \boldsymbol{Q}^{-1}\left(\left(\left(\boldsymbol{Q}\;\boldsymbol{\alpha}^t \;-\; \mathbf{1}\right) \;-\; \gamma\;\boldsymbol{\alpha}^t\right)_+ \;+\; \mathbf{1}\right)\;, \tag{3.140}$$

where $\left(\ldots\right)_+$ 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.

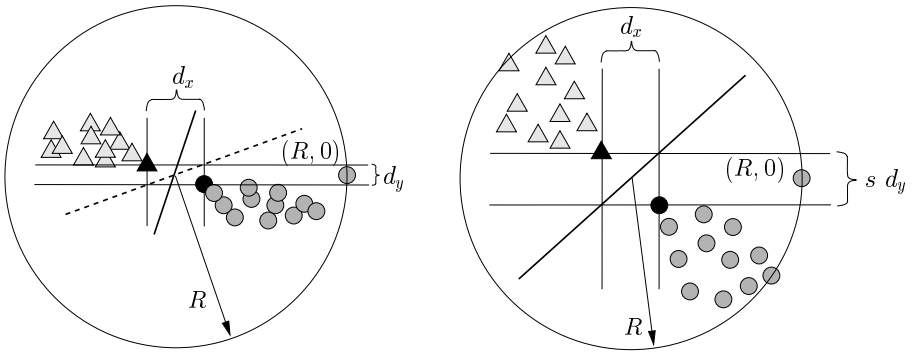


Figure 3.39: LEFT: Data points from two classes (triangles and circles) are separated by the SVM solution. The two support vectors are separated by d_x along the horizontal and by d_y along the vertical axis, from which we obtain the margin $\gamma = \frac{1}{2}\sqrt{d_x^2 + d_y^2}$ and $\frac{R^2}{\gamma^2} = \frac{4\ R^2}{d_x^2 + d_y^2}$. 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.

3.12 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. 3.39, where the optimal hyperplane is not invariant under scaling, hence predictions of class labels may change if the data is re-scaled before learning. The VC-dimension is bounded by

$$d_{\mathrm{VC}} \leq \frac{R^2}{\gamma^2}\ . \tag{3.141}$$

This bound is also not scale invariant as shown in Fig. 3.39.

However if we have 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 the scale factors: which scale factors should be used? Therefore scaling invariance seems to be desirable to obtain robust classification.

For the Potential Support Vector Machine (P-SVM, [Hochreiter and Mozer, 2001b,c, Hochreiter and Obermayer, 2002, Hochreiter et al., 2003, Hochreiter and Obermayer, 2003, 2004a,b, 2005]) 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:

$$R_{\min} \;=\; \max_i \left| \frac{\boldsymbol{w}^T}{\|\boldsymbol{w}\|} \boldsymbol{x}^i \right| \;. \tag{3.142}$$

The new complexity measure instead of the margin would be

$$\frac{R_{\min}^2}{\gamma^2} \;=\; \frac{1}{\|\boldsymbol{w}\|^2} \left(\max_i \left| \boldsymbol{w}^T \boldsymbol{x}^i \right| \right)^2 \|\boldsymbol{w}\|^2 \;=\; \max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;. \tag{3.143}$$

Note that

$$\max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;\leq\; \frac{R^2}{\gamma^2} \tag{3.144}$$

for $R = \max_i \left\| \boldsymbol{x}^i \right\|$ follows from the Cauchy-Schwarz inequality. Therefore the new bound 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 this objective is optimized instead:

$$\max_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;\leq\; \sum_i \left(\boldsymbol{w}^T \boldsymbol{x}^i \right)^2 \;=\; \boldsymbol{w}^T \boldsymbol{X}\, \boldsymbol{X}^T \boldsymbol{w} \;. \tag{3.145}$$

Using the objective

$$\boldsymbol{w}^T \boldsymbol{X}\, \boldsymbol{X}^T \boldsymbol{w} \;=\; \left\| \boldsymbol{X}^T \boldsymbol{w} \right\|^2 \;. \tag{3.146}$$

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 $\left(\boldsymbol{X}\, \boldsymbol{X}^T \right)^{-1}$ appears in various expressions.

Let us assume we have a dot product matrix

$$K_{ij} \;=\; \left(\boldsymbol{x}^i \right)^T \boldsymbol{z}^j \tag{3.147}$$

that is

$$\boldsymbol{K} \;=\; \boldsymbol{X}^T \boldsymbol{Z} \;. \tag{3.148}$$

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.

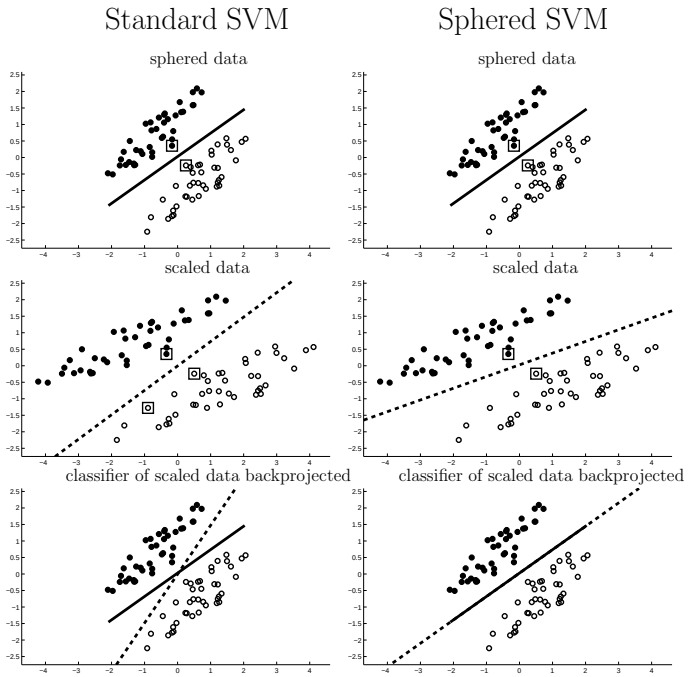


Figure 3.40: The standard SVM (left) in contrast to the sphered SVM (right). Scaling the data and back-scaling gives the original S-SVM solution in contrast to the standard SVM where the solutions differ.

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

$$r_i \;=\; \boldsymbol{w}^T \boldsymbol{x}^i \;+\; b \;-\; y^i \;. \tag{3.149}$$

If we assume

$$\boldsymbol{w} \;=\; \sum_{j=1}^{N} \alpha_j \boldsymbol{z}^j \tag{3.150}$$

then

$$r_i \;=\; \sum_{j=1}^{N} \alpha_j K_{ij} \;+\; b \;-\; y^i \;. \tag{3.151}$$

that is r_i is linear in K_{ij}.

The empirical error, the mean square error, is defined as

$$R_{\text{emp}} \;=\; \frac{1}{l} \sum_{i=1}^{l} r_i^2 \tag{3.152}$$

and it is minimal if

$$\frac{\partial R_{\text{emp}}}{\partial \alpha_j} \;=\; 2\,\frac{1}{l}\,\sum_{i=1}^{l} r_i\,K_{ij} \;=\; 0 \tag{3.153}$$

for every $1 \;\leq\; j \;\leq\; N$.

This motivates a new set of constraints

$$\boldsymbol{K}^T\boldsymbol{r} \;=\; \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;+\; b\mathbf{1} \;-\; \boldsymbol{y}\right) \;=\; \mathbf{0}\,, \tag{3.154}$$

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

$$\begin{aligned}\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;+b\mathbf{1} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;\leq\; \mathbf{0}\,, \\ \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;+b\mathbf{1} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;\geq\; \mathbf{0}\,.\end{aligned} \tag{3.155}$$

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 in terms of absolute values. To make the α_j comparable to one another, the vectors $K_{\cdot j}$ are normalized to zero mean and unit variance:

$$\sum_{i=1}^{m} K_{ij} \;=\; 0 \tag{3.156}$$

$$\frac{1}{m}\sum_{i=1}^{m} K_{ij}^2 \;=\; 1\,. \tag{3.157}$$

in a preprocessing step. That means

$$\boldsymbol{K}^T\mathbf{1} \;=\; 0\,. \tag{3.158}$$

The constraints simplify to

$$\begin{aligned}\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;\leq\; \mathbf{0}\,, \\ \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;\geq\; \mathbf{0}\,.\end{aligned} \tag{3.159}$$

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),

$$\begin{aligned}\min_{\boldsymbol{w},\boldsymbol{\xi}^+,\boldsymbol{\xi}^-}\;\; & \frac{1}{2}\,\|\boldsymbol{X}^T\,\boldsymbol{w}\|_2^2 \;+\; C\,\mathbf{1}^T\left(\boldsymbol{\xi}^+ \;+\; \boldsymbol{\xi}^-\right) \\ \text{s.t. } & \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;+\; \boldsymbol{\xi}^+ \;\geq\; \mathbf{0} \\ & \boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;-\; \boldsymbol{\xi}^- \;\leq\; \mathbf{0} \\ & \boldsymbol{\xi}^+ \;\geq\; \mathbf{0},\;\; \boldsymbol{\xi}^- \;\geq\; \mathbf{0}\;.\end{aligned} \tag{3.160}$$

The Lagrangian $\mathcal{L}$ is given by

$$\begin{aligned}\mathcal{L} \;=\; & \frac{1}{2}\,\boldsymbol{w}^T\,\boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;+\; C\,\mathbf{1}^T\left(\boldsymbol{\xi}^+ \;+\; \boldsymbol{\xi}^-\right) \;- \\ & \left(\boldsymbol{\alpha}^+\right)^T\,\left(\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;+\; \epsilon\,\mathbf{1} \;+\; \boldsymbol{\xi}^+\right) \;+ \\ & \left(\boldsymbol{\alpha}^-\right)^T\,\left(\boldsymbol{K}^T\left(\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{y}\right) \;-\; \epsilon\,\mathbf{1} \;-\; \boldsymbol{\xi}^-\right) \;- \\ & \left(\boldsymbol{\mu}^+\right)^T\,\boldsymbol{\xi}^+ \;-\; \left(\boldsymbol{\mu}^-\right)^T\,\boldsymbol{\xi}^- \;.\end{aligned} \tag{3.161}$$

The optimality conditions require that the following derivatives with respect to the primal variables of the Lagrangian $\mathcal{L}$ are zero:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} \;&=\; \boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;-\; \boldsymbol{X}\,\boldsymbol{K}\,\boldsymbol{\alpha} \;=\; \mathbf{0}\,, \\ \frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}^+} \;&=\; C\mathbf{1} \;-\; \boldsymbol{\alpha}^+ \;-\; \boldsymbol{\mu}^+ \;=\; \mathbf{0}\,, \\ \frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}^+} \;&=\; C\mathbf{1} \;-\; \boldsymbol{\alpha}^- \;-\; \boldsymbol{\mu}^- \;=\; \mathbf{0}\;.\end{aligned} \tag{3.162}$$

In order to ensure the first condition

$$\boldsymbol{X}\,\boldsymbol{X}^T\,\boldsymbol{w} \;=\; \boldsymbol{X}\,\boldsymbol{K}\,\boldsymbol{\alpha} \;=\; \boldsymbol{X}\,\boldsymbol{X}^T\boldsymbol{Z}\,\boldsymbol{\alpha} \tag{3.163}$$

one solution is

$$\boldsymbol{w} \;=\; \boldsymbol{Z}\,\boldsymbol{\alpha} \;=\; \sum_{j=1}^{N}\alpha_j\,\boldsymbol{z}^j\;. \tag{3.164}$$

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

$$\boldsymbol{\alpha}^+ \;\leq\; C\mathbf{1} \quad\text{and}\quad \boldsymbol{\alpha}^- \;\leq\; C\mathbf{1}\;. \tag{3.165}$$

The dual optimization problem of the P-SVM is:

$$\begin{aligned}\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-}\;\; & \frac{1}{2}\left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right)^T\,\boldsymbol{K}^T\,\boldsymbol{K}\,\left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right) \;- \\ & \boldsymbol{y}^T\,\boldsymbol{K}\,\left(\boldsymbol{\alpha}^+ \;-\; \boldsymbol{\alpha}^-\right) \;+\; \epsilon\,\mathbf{1}^T\left(\boldsymbol{\alpha}^+ \;+\; \boldsymbol{\alpha}^-\right) \\ \text{s.t. }\; & \mathbf{0} \;\leq\; \boldsymbol{\alpha}^+ \;\leq\; C\mathbf{1}\,, \\ & \mathbf{0} \;\leq\; \boldsymbol{\alpha}^- \;\leq\; C\mathbf{1}\,.\end{aligned} \tag{3.166}$$

To determine b we find its optimal value by setting the derivative of the empirical error to zero:

$$\frac{\partial R_{\text{emp}}}{\partial b} \;=\; 2\,\frac{1}{l}\sum_{i=1}^{l} r_i \;=\; 0\,, \tag{3.167}$$

therefore

$$b = \frac{1}{l}\sum_{i=1}^{l} y^i - \boldsymbol{w}^T\boldsymbol{x}^i = \tag{3.168}$$
$$\frac{1}{l}\sum_{i=1}^{l} y^i - \frac{1}{l}\boldsymbol{w}^T\boldsymbol{X}\mathbf{1} =$$
$$\frac{1}{l}\sum_{i=1}^{l} y^i - \frac{1}{l}\boldsymbol{\alpha}^T\boldsymbol{K}^T\mathbf{1} =$$
$$\frac{1}{l}\sum_{i=1}^{l} y^i \,.$$

The discriminant function is

$$\sum_{j=1}^{N} \alpha_j \, (\boldsymbol{z}^j)^T\boldsymbol{x} + b = \sum_{j=1}^{N} \alpha_j \, K_{xj} + b \,. \tag{3.169}$$

At the optimum, the value of the objective is

$$\boldsymbol{w}^T \boldsymbol{X} \boldsymbol{X}^T \boldsymbol{w} = \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^T \boldsymbol{K}^T\boldsymbol{y} - \tag{3.170}$$
$$\epsilon \, \mathbf{1}^T \left(\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-\right) - \boldsymbol{\alpha}^+ \boldsymbol{\xi}^+ + \boldsymbol{\alpha}^-\boldsymbol{\xi}^- \,.$$

It can be seen that increasing ϵ or allowing larger values of $\boldsymbol{\xi}^+$ or $\boldsymbol{\xi}^-$ reduces the complexity term.

Note, that $\boldsymbol{K}$ 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.

3.13 SVM Optimization: Sequential Minimal Optimization

Since 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 and extract the support

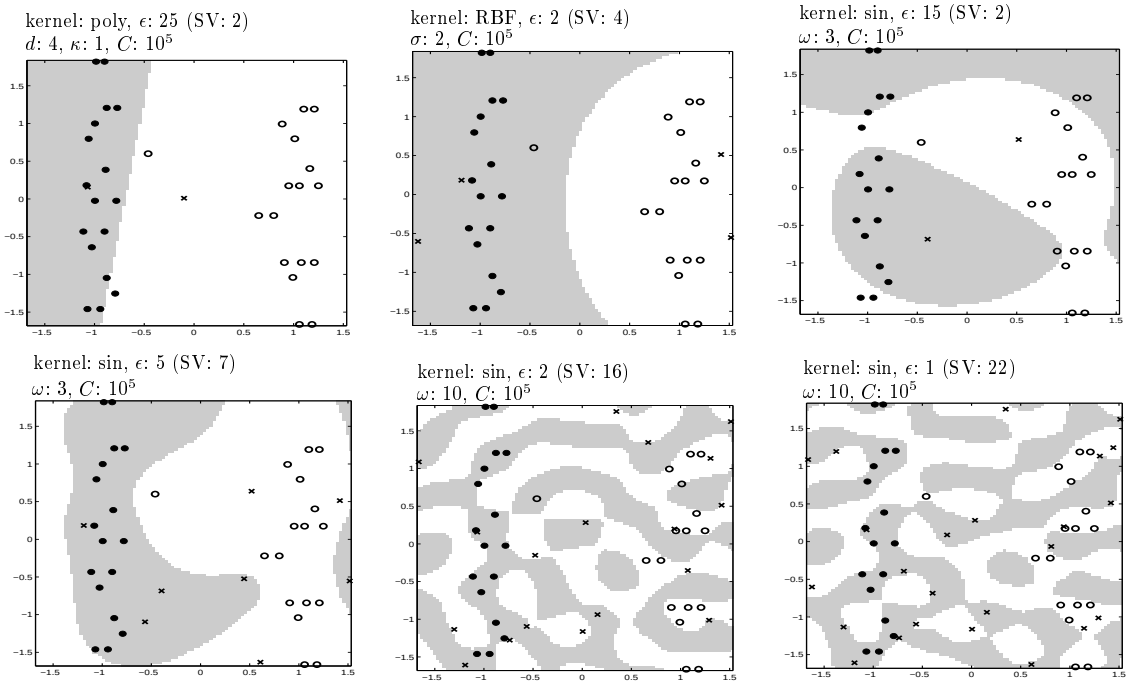


Figure 3.41: Application of the P-SVM method to a toy classification problem. Objects x and complex features z are in a two-dimensional space. The feature vectors $\boldsymbol{x}$ for 34 objects, 17 from each class, were chosen as shown in the figure (open and solid circles), and 50 feature vectors (complex 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.

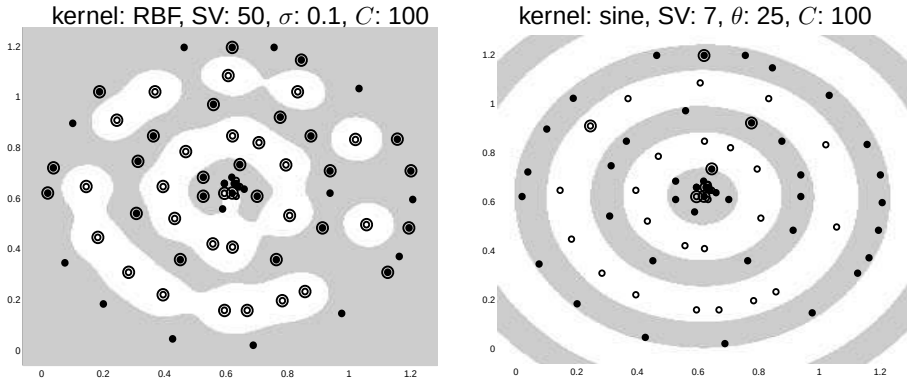


Figure 3.42: Application of the P-SVM method to another toy classification problem. Objects are described by two-dimensional feature vectors $\boldsymbol{x}$, and 70 feature vectors were generated of which 28 belong to class 1 (open circles) and 42 belong to class 2 (solid circles). A Gram matrix was 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.

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. (3.42):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} & \frac{1}{2} \sum_{i,j} \alpha_i \, y^i \, \alpha_j \, y^j \, \left(\boldsymbol{x}^i\right)^T \boldsymbol{x}^j \; - \; \sum_{i=1}^{l} \alpha_i \\ \text{s.t. } & 0 \; \leq \; \alpha_i \; \leq \; C \\ & \sum_{i=1}^{l} \alpha_i \, y^i \; = \; 0 \, . \end{aligned} \tag{3.171}$$

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

$$y^1 \, \alpha_1 \; + \; y^2 \, \alpha_2 \; + \; \sum_{i=3}^{l} \alpha_i \, y^i \; = \; 0 \tag{3.172}$$

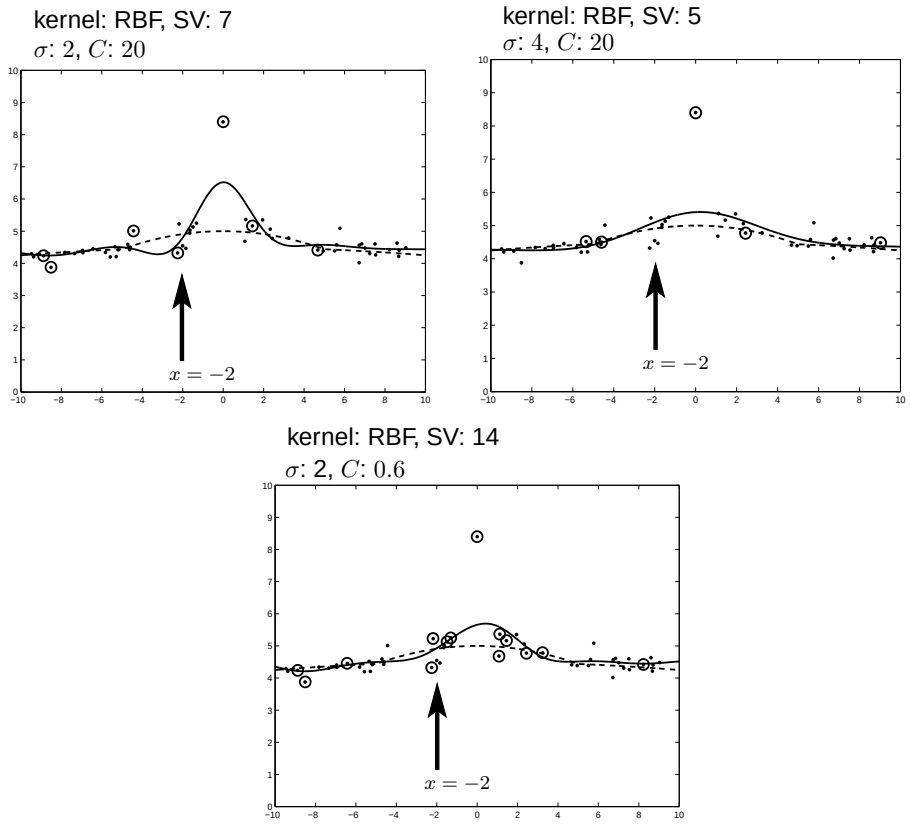


Figure 3.43: Application of the P-SVM method to a toy regression problem. Objects (small dots), described by the x-coordinate, were generated by randomly choosing points from the true function (dashed line) and adding Gaussian noise with mean 0 and standard deviation 0.2 to the y-component of each data 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.

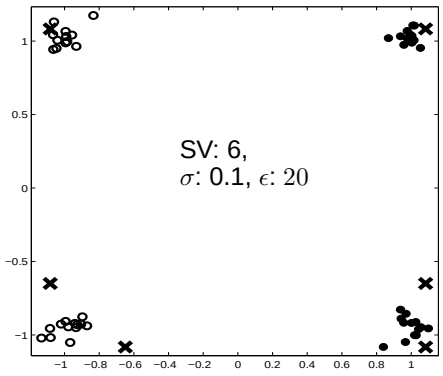


Figure 3.44: Application of the P-SVM method to a toy feature selection problem for a classification task. Objects are described by two-dimensional vectors $\boldsymbol{x}$ and complex features by $\boldsymbol{z}$. Vectors $\boldsymbol{x}$ for 50 objects, 25 from each class (open and solid circles), 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.

which is

$$y^1\,y^2\,\alpha_1\,+\,\alpha_2\,=\,-\,\sum_{i=3}^{l}\alpha_i\,y^i\,y^2 \tag{3.173}$$

If we set $s\,=\,y^1\,y^2$ and $\gamma\,=\,-\,\sum_{i=3}^{l}\alpha_i\,y^i\,y^2$ then we have

$$s\,\alpha_1\,+\,\alpha_2\,=\,\gamma\,. \tag{3.174}$$

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 two variables

$$\begin{aligned}\min_{\alpha_1,\alpha_2}\frac{1}{2}\,&\left(\alpha_1^2\,K_{11}\,+\,\alpha_2^2\,K_{22}\,+\,2\,s\,\alpha_1\,\alpha_2\,K_{12}\right)\,+\\ &c_1\,\alpha_1\,+\,c_2\,\alpha_2\\ \text{s.t. }0\,&\le\,\alpha_1,\alpha_2\,\le\,C\\ &s\,\alpha_1\,+\,\alpha_2\,=\,\gamma\,,\end{aligned} \tag{3.175}$$

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. 3.46.

Let us consider for the moment the optimization problem without the box constraints $0\,\le\,\alpha_1,\alpha_2\,\le\,C$.

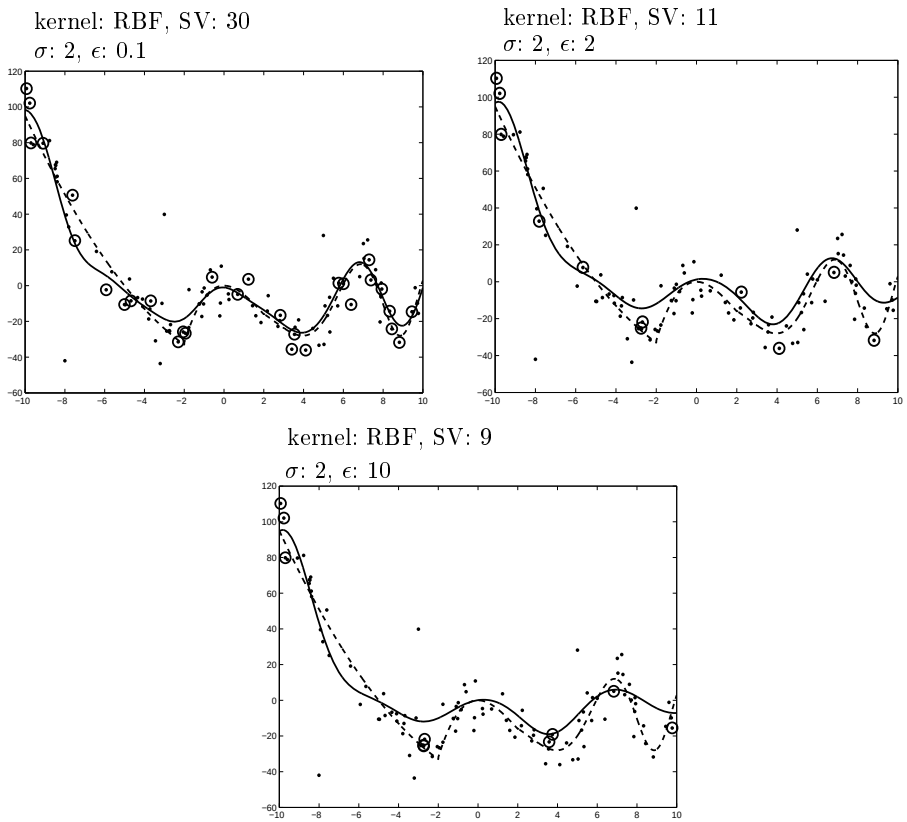


Figure 3.45: Application of the P-SVM to a toy feature selection problem for a regression task. 100 data points are generated from the true function (dashed line) by randomly and uniformly choosing data points from the true function and adding Gaussian noise with mean 0 and standard deviation 10 to the 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.

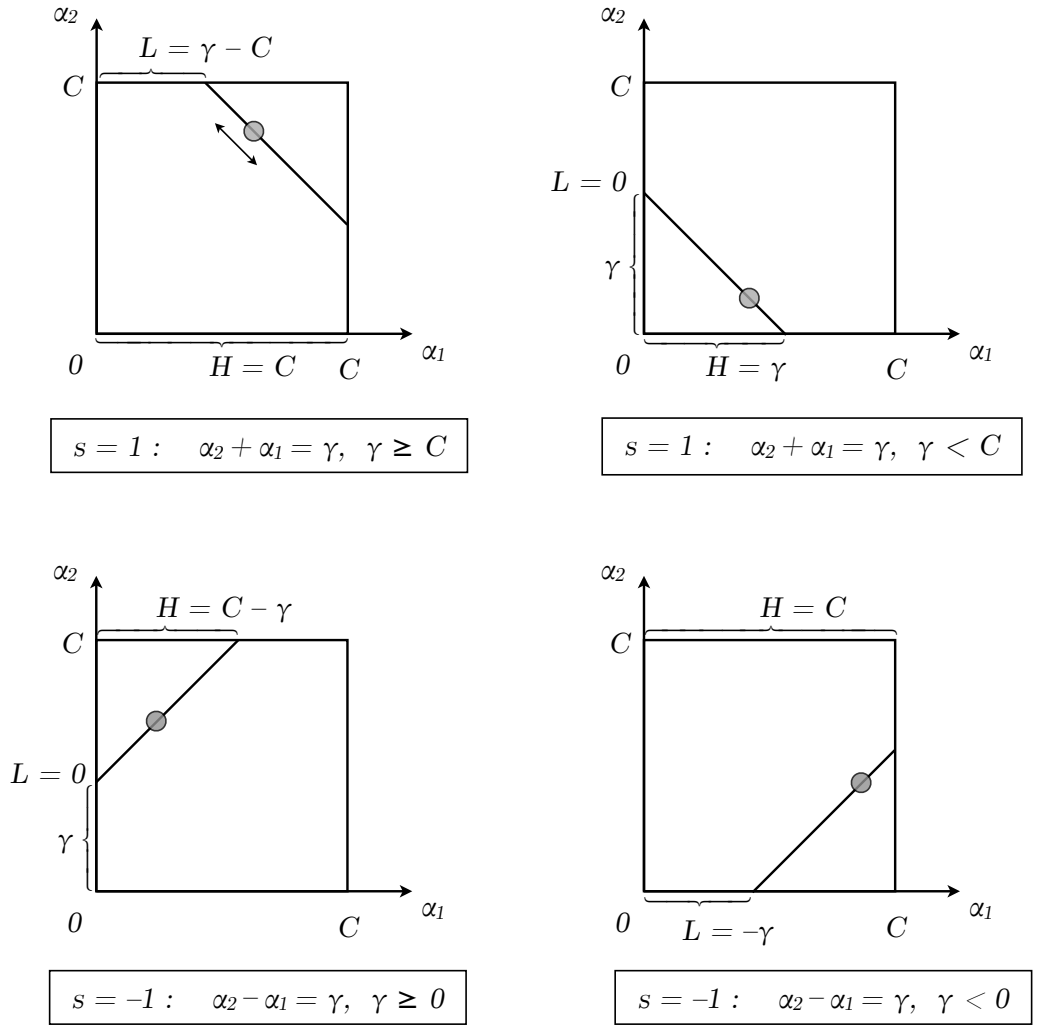


Figure 3.46: The two Lagrange multipliers α_1 and α_2 must fulfill the constraint $s\,\alpha_1 + \alpha_2 = \gamma$. Dependent on values for s and γ four cases are differentiated. L and H show the lower and upper bound for α_1 defined in eq. 3.191 and 3.192.

We insert $\alpha_2 \;=\; \gamma \;-\; s\,\alpha_1$ into the objective and obtain

$$\begin{aligned}
&\frac{1}{2}\,\Big(\alpha_1^2\,K_{11} \;+\; (\gamma \;-\; s\,\alpha_1)^2\,K_{22} \;+\; 2\,s\,\alpha_1\;(\gamma \;-\; s\,\alpha_1)\,K_{12}\Big) \\
&+\; c_1\,\alpha_1 \;+\; c_2\;(\gamma \;-\; s\,\alpha_1) \;= \\
&\frac{1}{2}\;\big((K_{11} \;+\; K_{22} \;-\; 2\,K_{12})\,\alpha_1^2 + \\
&\;(-\,2\,\gamma\,s\,K_{22} \;+\; \;2\,\gamma\,s\,K_{12} \;+\; 2\,c_1 \;-\; 2\,s\,c_2)\,\alpha_1) \;+ \\
&\frac{1}{2}\,\gamma^2\,K_{22} \;+\; c_2\,\gamma
\end{aligned} \tag{3.176}$$

For the optimum, the derivative with respect to α_1 must be zero:

$$\begin{aligned}
&(K_{11} \;+\; K_{22} \;-\; 2\,K_{12})\,\alpha_1 \;+ \\
&(-\,\gamma\,s\,K_{22} \;+\; \gamma\,s\,K_{12} \;+\; c_1 \;-\; s\,c_2) \;=\; 0
\end{aligned} \tag{3.177}$$

The optimal α_1 is

$$\alpha_1 = \frac{\gamma\,s\,K_{22} \;-\; \gamma\,s\,K_{12} \;-\; c_1 \;+\; s\,c_2}{K_{11} \;+\; K_{22} \;-\; 2\,K_{12}}\;. \tag{3.178}$$

We use the discriminant function values

$$f(\boldsymbol{x}^1) \;=\; \sum_{i=1}^{l} y^i\,\alpha_i\,K_{i1} \;+\; b \tag{3.179}$$

$$f(\boldsymbol{x}^2) \;=\; \sum_{i=1}^{l} y^i\,\alpha_i\,K_{i2} \;+\; b\,. \tag{3.180}$$

We now can rewrite c_1 and c_2 as

$$c_1 \;=\; y^1\,f(\boldsymbol{x}^1) \;-\; y^1\,b \;-\; 1 \;-\; K_{11}\,\alpha_1 \;-\; s\,K_{12}\,\alpha_2 \tag{3.181}$$

$$c_2 \;=\; y^2\,f(\boldsymbol{x}^2) \;-\; y^2\,b \;-\; 1 \;-\; K_{22}\,\alpha_2 \;-\; s\,K_{12}\,\alpha_1 \tag{3.182}$$

$$s\,c_2 \;=\; y^1\,f(\boldsymbol{x}^2) \;-\; y^1\,b \;-\; s \;-\; s\,K_{22}\,\alpha_2 \;-\; K_{12}\,\alpha_1 \tag{3.183}$$

$$\tag{3.184}$$

Because of

$$\gamma \;=\; \alpha_2 \;+\; s\,\alpha_1 \tag{3.185}$$

we obtain

$$\begin{aligned}
&\gamma\,s\,K_{22} \;-\; \gamma\,s\,K_{12} \;= \\
&s\,K_{22}\,\alpha_2 \;+\; K_{22}\,\alpha_1 \;-\; s\,K_{12}\,\alpha_2 \;-\; K_{12}\,\alpha_1\,.
\end{aligned} \tag{3.186}$$

Now we can rewrite the enumerator of eq. (3.178):

$$\begin{aligned}
&\gamma\, s\, K_{22} \;-\; \gamma\, s\, K_{12} \;-\; c_1 \;+\; s\, c_2 \;= \\
&s\, K_{22}\alpha_2 \;+\; K_{22}\, \alpha_1 \;-\; s\, K_{12}\, \alpha_2 \;-\; K_{12}\, \alpha_1 \;- \\
&y^1\, f(\boldsymbol{x}^1) \;+\; y^1\, b \;+\; 1 \;+\; K_{11}\, \alpha_1 \;+\; s\, K_{12}\, \alpha_2 \;+ \\
&y^1\, f(\boldsymbol{x}^2) \;-\; y^1\, b \;-\; s \;-\; s\, K_{22}\, \alpha_2 \;-\; K_{12}\, \alpha_1 \;= \\
&y^1 \left(f(\boldsymbol{x}^2) \;-\; f(\boldsymbol{x}^1)\right) \;+ \\
&\alpha_1 \left(K_{22} \;+\; K_{11} \;-\; 2\, K_{12}\right) \;+\; 1 \;-\; s \;= \\
&y^1 \left(\left(f(\boldsymbol{x}^2) \;-\; y^2\right) \;-\; \left(f(\boldsymbol{x}^1) \;-\; y^1\right)\right) \;+ \\
&\alpha_1 \left(K_{22} \;+\; K_{11} \;-\; 2\, K_{12}\right) \;,
\end{aligned} \tag{3.187}$$

where the last equality stems from the fact that $(1 \;-\; s) \;=\; y^1\, (y^1 \;-\; y^2)$.

As update for α_1 we obtain

$$\alpha_1^{\text{new}} = \alpha_1 \;+\; \frac{y^1 \left(\left(f(\boldsymbol{x}^2) \;-\; y^2\right) \;-\; \left(f(\boldsymbol{x}^1) \;-\; y^1\right)\right)}{K_{11} \;+\; K_{22} \;-\; 2\, K_{12}} \;. \tag{3.188}$$

Because

$$s\, \alpha_1^{\text{new}} \;+\; \alpha_2^{\text{new}} \;=\; \gamma \;=\; s\, \alpha_1 \;+\; \alpha_2 \tag{3.189}$$

we obtain for the update of α_2

$$\alpha_2^{\text{new}} = \alpha_2 \;+\; s\; (\alpha_1 \;-\; \alpha_1^{\text{new}}) \;. \tag{3.190}$$

However we did not consider the box constraints until now.

We define as new bounds

$$L \;=\; \begin{cases} \max\{0, \alpha_1 \;-\; \alpha_2\} & \text{if } s \;=\; -1 \\ \max\{0, \alpha_1 \;+\; \alpha_2 \;-\; C\} & \text{otherwise} \end{cases} \tag{3.191}$$

$$H \;=\; \begin{cases} \min\{C, C \;+\; \alpha_1 \;-\; \alpha_2\} & \text{if } s \;=\; -1 \\ \min\{C, \alpha_1 \;+\; \alpha_2\} & \text{otherwise} \end{cases} \;. \tag{3.192}$$

We obtain finally following update rules:

if

$$K_{11} \;+\; K_{22} \;-\; 2\, K_{12} \;=\; 0 \tag{3.193}$$

then

$$\alpha_1^{\text{new}} \;=\; \begin{cases} H & \text{if } y^1 \left(\left(f(\boldsymbol{x}^2) \;-\; y^2\right) \;-\; \left(f(\boldsymbol{x}^1) \;-\; y^1\right)\right) \;>\; 0 \\ L & \text{otherwise} \end{cases} \;. \tag{3.194}$$

else

$$K_{11} \;+\; K_{22} \;-\; 2\, K_{12} \;>\; 0 \tag{3.195}$$

then first compute

$$\alpha_1^{\text{temp}} = \alpha_1 \; + \; \frac{y^1 \left(\left(f(\boldsymbol{x}^2) \; - \; y^2 \right) \; - \; \left(f(\boldsymbol{x}^1) \; - \; y^1 \right) \right)}{K_{11} \; + \; K_{22} \; - \; 2 \, K_{12}} \; . \tag{3.196}$$

and then

$$\alpha_1^{\text{new}} \; = \; \begin{cases} H & \text{if } \; H \; \leq \; \alpha_1^{\text{temp}} \\ \alpha_1^{\text{temp}} & \text{if } L \; < \; \alpha_1^{\text{temp}} \; < \; H \\ L & \text{if } \alpha_1^{\text{temp}} \; \leq \; L \end{cases} \; . \tag{3.197}$$

In both cases ("if" and "else" case) the update for α_2 is

$$\alpha_2^{\text{new}} = \alpha_2 \; + \; s \; (\alpha_1 \; - \; \alpha_1^{\text{new}}) \; . \tag{3.198}$$

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) all non-bound examples are looked at 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.

3.14 Designing Kernels for Bioinformatics Applications

At the beginning of the chapter in Section 3.1, 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 called "feature map" and the feature map combined with dot product is a kernel.

3.14.1 String Kernel

The string kernel describes a string by all its subsequences. Subsequences do not need to be contiguous which means that gaps are allowed.

The string kernel counts how often the subsequence occurs (k) and determines the distance (t) between the first and the last symbol. The contribution of a subsequence is

$$k\,\lambda^t\,, \tag{3.199}$$

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 has 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.

3.14.2 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.

3.14.3 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.

3.14.4 Motif Kernel

A library of motifs from PROSITE or BLOCKS or PRINTS databases is build. The occurrence of a motif is the feature vector.

3.14.5 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.

3.14.6 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).

$$SW(\boldsymbol{x}, \boldsymbol{y}, S, g) \;=\; \max_{i_s, i_e, j_s, j_e} s(\boldsymbol{x}, \boldsymbol{y}, S, g, i_s, i_e, j_s, j_e) \;, \tag{3.200}$$

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"

$$k_{LA}(\boldsymbol{x}, \boldsymbol{y}, S, g, \beta) \;= \tag{3.201}$$
$$\sum_{i_s=1, i_e=i_s+1, j_s=1, j_e=j_s+1} \exp\left(\beta \; s(\boldsymbol{x}, \boldsymbol{y}, S, g, i_s, i_e, j_s, j_e)\right)$$

is a positive definite kernel.

3.14.7 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 no longer necessary.

3.14.8 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 that can be interpreted statistically.

A class of kernels is defined as

$$k_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{y}) \;=\; \frac{\partial \ln p(\boldsymbol{x}; \boldsymbol{w})}{\partial \boldsymbol{w}} \; \boldsymbol{M}^{-1} \; \frac{\partial \ln p(\boldsymbol{y}; \boldsymbol{w})}{\partial \boldsymbol{w}} \;, \tag{3.202}$$

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.

3.14.9 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.

3.14.10 Kernels Based on Chemical Properties

Kernels have been suggested 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.

3.14.11 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.

3.14.12 Salzberg DNA Kernel

Similar as with PSSMs also for DNA log-odd scores can be computed according to the Salzberg method.

3.14.13 Shifted Weighted Degree Kernel

This is a string kernel where positions are important. The "Weighted Degree Kernel" uses fixed position measured with respect to a reference point. The "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.

3.15 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 4

Fisher and 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 $\boldsymbol{w}$:

$$J(\boldsymbol{w}) \;=\; \frac{\boldsymbol{w}^T \boldsymbol{M}\, \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w}}\,, \tag{4.1}$$

where

$$\boldsymbol{M} \;=\; (\boldsymbol{m}_- \;-\; \boldsymbol{m}_+)\; (\boldsymbol{m}_- \;-\; \boldsymbol{m}_+)^T \tag{4.2}$$

is the between class variance and

$$\boldsymbol{N} \;=\; \sum_{i=1;y^i=1}^{l} \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_+\right)\; \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_+\right)^T \;+ \tag{4.3}$$

$$\sum_{i=1;y^i=-1}^{l} \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_-\right)\; \left(\boldsymbol{x}^i \;-\; \boldsymbol{m}_-\right)^T$$

is the within class variance.

The Fisher's discriminant attempts to minimize the variance within one class whilst separating the class centers as 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

$$\boldsymbol{m}_+ \;=\; \frac{1}{l_+} \sum_{i=1;y^i=1}^{l} \boldsymbol{x}^i \tag{4.4}$$

$$\boldsymbol{m}_- \;=\; \frac{1}{l_-} \sum_{i=1;y^i=-1}^{l} \boldsymbol{x}^i \tag{4.5}$$

Note that

$$\boldsymbol{w}^T \boldsymbol{M}\, \boldsymbol{w} \;=\; \left(\boldsymbol{w}^T \boldsymbol{m}_- \;-\; \boldsymbol{w}^T \boldsymbol{m}_+\right)\; \left(\boldsymbol{m}_-^T \boldsymbol{w} \;-\; \boldsymbol{m}_+^T \boldsymbol{w}\right) \tag{4.6}$$

and

$$\begin{aligned} &\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w} \;= \\ &\sum_{i=1;y^i=1}^{l} \left(\boldsymbol{w}^T \boldsymbol{x}^i \;-\; \boldsymbol{w}^T \boldsymbol{m}_+\right)\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{w} \;-\; \boldsymbol{m}_+^T \boldsymbol{w}\right) \;+ \\ &\sum_{i=1;y^i=-1}^{l} \left(\boldsymbol{w}^T \boldsymbol{x}^i \;-\; \boldsymbol{w}^T \boldsymbol{m}_-\right)\; \left(\left(\boldsymbol{x}^i\right)^T \boldsymbol{w} \;-\; \boldsymbol{m}_-^T \boldsymbol{w}\right) \end{aligned} \tag{4.7}$$

Next we kernalize Fisher's discriminant analysis. Toward this end we set

$$\boldsymbol{w} \;=\; \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}(\boldsymbol{x}^i)\;. \tag{4.8}$$

Let $\boldsymbol{K}$ be 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. (4.8) and obtain

$$\begin{aligned} \boldsymbol{w}^T \boldsymbol{m}_- \;=\; \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x}^j) \;= \\ \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \sum_{i=1}^{l} \alpha_i\; \boldsymbol{\Phi}^T(\boldsymbol{x}^i)\; \boldsymbol{\Phi}(\boldsymbol{x}^j) \;= \\ \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \sum_{i=1}^{l} \alpha_i\; k(\boldsymbol{x}^i, \boldsymbol{x}^j)\;. \end{aligned} \tag{4.9}$$

We use again

$$\boldsymbol{k}(\boldsymbol{x}, .) \;=\; \Big(k(\boldsymbol{x}, \boldsymbol{x}^1), \ldots, k(\boldsymbol{x}, \boldsymbol{x}^l)\Big)^T \tag{4.10}$$

and define

$$\boldsymbol{\mu}_- \;=\; \frac{1}{l_-} \sum_{j=1;y^j=-1}^{l} \boldsymbol{k}(\boldsymbol{x}^j, .) \;=\; \frac{1}{l_-}\; \boldsymbol{K}\; \boldsymbol{1}_- \tag{4.11}$$

to obtain

$$\boldsymbol{w}^T \boldsymbol{m}_- \;=\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_-\;. \tag{4.12}$$

Here we used $\boldsymbol{1}_-$ the l-dimensional vector which has ones at positions of class -1 examples and 0 otherwise.

Analog using

$$\boldsymbol{\mu}_+ \;=\; \frac{1}{l_+} \sum_{j=1;y^j=1}^{l} \boldsymbol{k}(\boldsymbol{x}^j,.) \;=\; \frac{1}{l_+}\, \boldsymbol{K}\; \mathbf{1}_+ \tag{4.13}$$

we obtain

$$\boldsymbol{w}^T \boldsymbol{m}_+ \;=\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_+ \,. \tag{4.14}$$

We used $\mathbf{1}_+$ the l-dimensional vector which has ones at positions of class 1 examples and 0 otherwise.

$$\begin{aligned} &\boldsymbol{w}^T \boldsymbol{M}\, \boldsymbol{w} \;=\; \left(\boldsymbol{\alpha}^T \boldsymbol{\mu}_- \;-\; \boldsymbol{\alpha}^T \boldsymbol{\mu}_+\right) \left(\boldsymbol{\mu}_-^T \boldsymbol{\alpha} \;-\; \boldsymbol{\mu}_+^T \boldsymbol{\alpha}\right) \;= \\ &\boldsymbol{\alpha}^T \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+\right) \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+\right)^T \boldsymbol{\alpha} \;=\; \left(\boldsymbol{\alpha}^T \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+\right)\right)^2 \;= \\ &\boldsymbol{\alpha}^T \boldsymbol{M}_k\, \boldsymbol{\alpha}\,, \end{aligned} \tag{4.15}$$

where

$$\boldsymbol{M}_k \;=\; \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+\right) \left(\boldsymbol{\mu}_- \;-\; \boldsymbol{\mu}_+\right)^T \,. \tag{4.16}$$

The expansion of w according to eq. (4.8) 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{4.17}
$$

If we define

$$
\boldsymbol{N}_k \;=\; \boldsymbol{K}\, \boldsymbol{K}^T \;-\; l_+\, \boldsymbol{\mu}_+\, \boldsymbol{\mu}_+^T \;-\; l_-\, \boldsymbol{\mu}_-\, \boldsymbol{\mu}_-^T \tag{4.18}
$$

then

$$
\boldsymbol{w}^T \boldsymbol{N}\, \boldsymbol{w} \;=\; \boldsymbol{\alpha}^T \boldsymbol{N}_k\, \boldsymbol{\alpha}\,. \tag{4.19}
$$

We obtain for the kernel version of Fisher's discriminant the following Rayleigh coefficient to

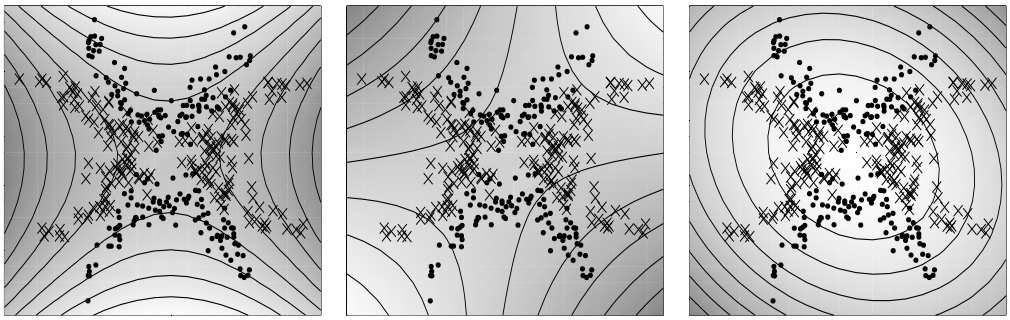


Figure 4.1: Kernel discriminant analysis (KDA) example. Left: KDA contour lines. Middle and Right: first and second component of kernel PCA. With KDA the contour lines are useful to separate the data. From [Mika et al., 1999].

maximize:

$$J(\boldsymbol{\alpha}) \;=\; \frac{\boldsymbol{\alpha}^T \boldsymbol{M}_k \, \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{N}_k \, \boldsymbol{\alpha}} \; . \tag{4.20}$$

The projection of a new vector onto the discriminant direction $\boldsymbol{w}$ is

$$\boldsymbol{w}^T \, \boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i \, \boldsymbol{\Phi}^T(\boldsymbol{x}^i) \boldsymbol{\Phi}(\boldsymbol{x}) \;=\; \sum_{i=1}^{l} \alpha_i \, k(\boldsymbol{x}^i, \boldsymbol{x}) \; . \tag{4.21}$$

The Rayleigh coefficient eq. (4.20) can be maximized by the generalized eigenvalue problem

$$\boldsymbol{M}_k \, \boldsymbol{\alpha} \;=\; \lambda \, \boldsymbol{N}_k \, \boldsymbol{\alpha} \tag{4.22}$$

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

$$\boldsymbol{N}_k \;+\; \epsilon \boldsymbol{I} \; . \tag{4.23}$$

Fig. 4.1 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

$$\boldsymbol{w}^T \boldsymbol{m}_+ \;+\; b \;=\; 1 \tag{4.24}$$

and

$$\boldsymbol{w}^T \boldsymbol{m}_- \;+\; b \;=\; -1 \; . \tag{4.25}$$

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

$$\boldsymbol{w}^T \boldsymbol{x}^i \; + \; b \; = \; y^i \; + \; \xi_i \tag{4.26}$$

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

$$\mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \tag{4.27}$$

$$\mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \, . \tag{4.28}$$

The class 1 variance plus the class 2 variance is given by

$$\boldsymbol{\xi}^T \boldsymbol{\xi} \; = \; \|\boldsymbol{\xi}\|^2 \; . \tag{4.29}$$

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:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\xi},b} \; & \|\boldsymbol{\xi}\|^2 \; + \; C \, \|\boldsymbol{\alpha}\|^2 \\ \text{s.t.} \; & \boldsymbol{K} \, \boldsymbol{\alpha} \; + \; b\mathbf{1} \; = \; \boldsymbol{y} \; + \; \boldsymbol{\xi} \\ & \mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \\ & \mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \end{aligned} \tag{4.30}$$

or

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\xi},b} \; & \|\boldsymbol{\xi}\|^2 \; + \; C \boldsymbol{\alpha}^T \boldsymbol{K} \, \boldsymbol{\alpha} \\ \text{s.t.} \; & \boldsymbol{K} \, \boldsymbol{\alpha} \; + \; b\mathbf{1} \; = \; \boldsymbol{y} \; + \; \boldsymbol{\xi} \\ & \mathbf{1}_+ \, \boldsymbol{\xi} \; = \; 0 \;\; \text{and} \\ & \mathbf{1}_- \, \boldsymbol{\xi} \; = \; 0 \, . \end{aligned} \tag{4.31}$$

Note, that this formulation is very similar to the least square SVM in Subsection 3.11. $\boldsymbol{K} \, \boldsymbol{\alpha} \; = \; \boldsymbol{X}^T \boldsymbol{w}$ 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.

Chapter 5

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 competitions neural networks outperformed support vector machines but needed a lot of computational time. Some of these challenges have been won by Radford Neal who estimated the posterior by Monte Carlo sampling.

5.1 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. 5.1) 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 an 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.

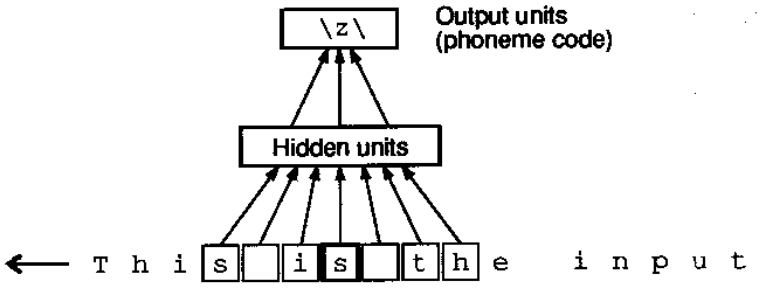


Figure 5.1: The NETTalk neural network architecture is depicted. A window scans the text and the network should predict the pronunciation of the letter in the center of the window. The same architecture was used for protein secondary structure prediction in [Qian and Sejnowski, 1988], where the input 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 they were not superior to neural networks.

Currently the best method is PORTER which is based on a recurrent neural network architecture.

5.2 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 can combine and modulate the incoming information. The incoming information changes 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

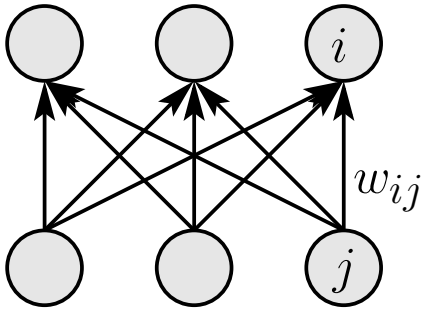


Figure 5.2: Artificial neural networks: units and weights. The weight w_{ij} gives the weight, connection strength, or synaptic weight from units j to unit i.

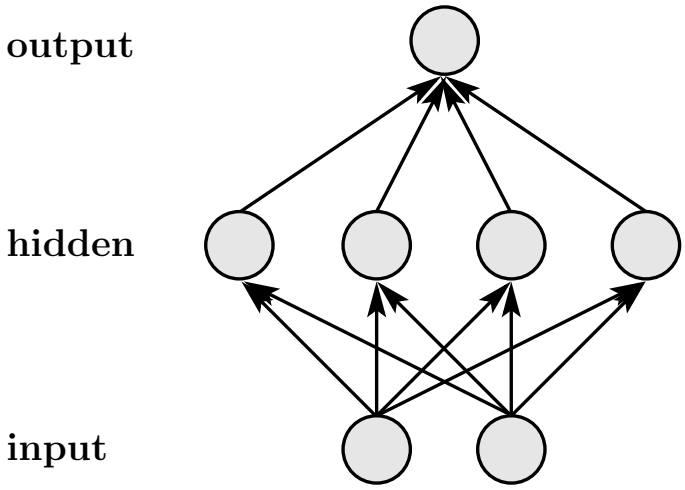


Figure 5.3: Artificial neural networks: a 3-layered net with an input, hidden, and output layer.

in ANNs are parameters w_{ij} giving the strength of the connection from unit j to unit i which are called *weights*. See Fig. 5.2 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. 5.3 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

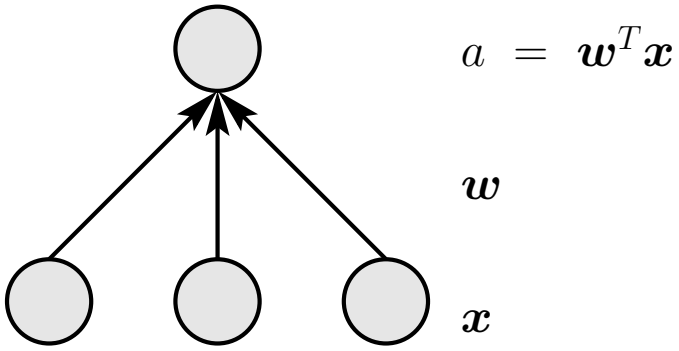


Figure 5.4: A linear network with one output unit.

powerful as Turing machines. Thus, ANNs are powerful enough to model a part of the world to solve a problem.

5.3 Linear Neurons and the Perceptron

In this section we will focus on (artificial) neural networks (NNs) which represent linear discriminant functions.

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 related 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

$$g(\boldsymbol{x}; \boldsymbol{w}) = a = \boldsymbol{w}^T \boldsymbol{x} \,. \tag{5.1}$$

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. 5.4).

If we assume Gaussian noise, then the proper noise model is the least square error. The optimal solution is the least square estimator:

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y} \,. \tag{5.2}$$

This solution may also be obtained by gradient descent on the empirical error function

$$R_{\text{emp}} = \frac{1}{2} \sum_{i=1}^{l} \left(y^i - g(\boldsymbol{x}^i; \boldsymbol{w})\right)^2 = \tag{5.3}$$

$$\frac{1}{2} \sum_{i=1}^{l} \left(y^i - \boldsymbol{w}^T \boldsymbol{x}^i\right)^2 = \frac{1}{2} \left(\boldsymbol{y} - \boldsymbol{X}\, \boldsymbol{w}\right)^T \left(\boldsymbol{y} - \boldsymbol{X}\, \boldsymbol{w}\right) .$$

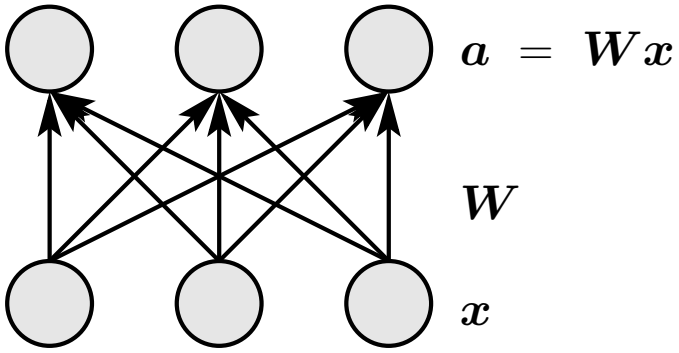


Figure 5.5: A linear network with three output units.

The gradient is

$$\frac{\partial}{\partial \boldsymbol{w}} R_{\text{emp}} \;=\; \sum_{i=1}^{l} \left(y^i \;-\; \boldsymbol{w}^T \boldsymbol{x}^i\right) \boldsymbol{x}^i \;=\; \left(\boldsymbol{y} \;-\; \boldsymbol{X}\, \boldsymbol{w}\right)^T \boldsymbol{X} \;. \tag{5.4}$$

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

$$\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{w}) \;=\; \boldsymbol{a} \;=\; \boldsymbol{W}\boldsymbol{x} \;, \tag{5.5}$$

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. 5.5 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

$$g_i(\boldsymbol{x}; \boldsymbol{w}) \;=\; a_i \;=\; \boldsymbol{w}_i^T \boldsymbol{x} \;, \tag{5.6}$$

where $\boldsymbol{w}_i$ is the i-th line of $\boldsymbol{W}$ written as column vector.

Because the optimal solution is

$$\hat{\boldsymbol{w}}_i \;=\; \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y}^i \;, \tag{5.7}$$

where $\boldsymbol{y}^i$ is the i-th column of $\boldsymbol{Y}$, we obtain as solution for the whole network

$$\hat{\boldsymbol{W}^T} \;=\; \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{Y} \;. \tag{5.8}$$

A linear neuron with a threshold function is called *perceptron*. The output of the perceptron is binary.

The perceptron model is

$$a \;=\; \text{sign}\left(\boldsymbol{w}^T \boldsymbol{x}\right) \;, \tag{5.9}$$

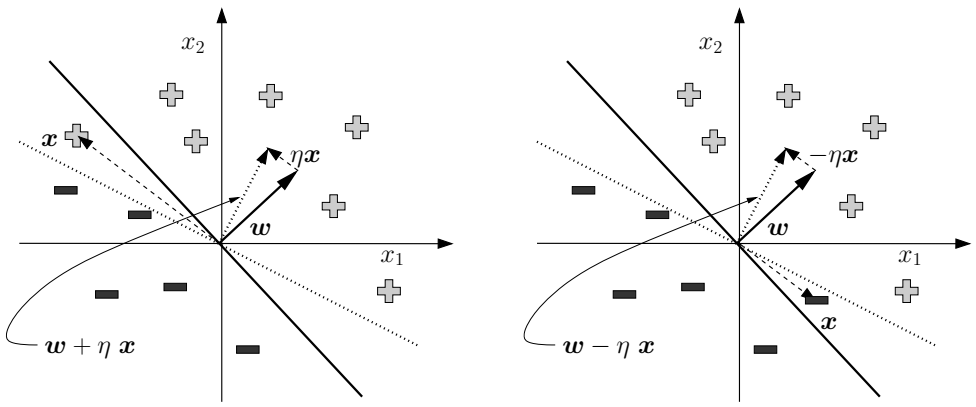


Figure 5.6: The perceptron learning rule. Left: a positive example is wrongly classified and correctly classified after the weight update. Right: a negative example is wrongly classified and correctly processed after weight update.

where sign is called *threshold function*. Note that the perceptron is a linear classifier from Section 3.2 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

$$R_{\text{emp}} \ = \ - \sum_{i=1;\ a^i y^i = -1}^{l} y^i \ \boldsymbol{w}^T \boldsymbol{x}^i \ . \tag{5.10}$$

The error is the margin error of the support vector machine e.g. in Section 3.5 in the eq. (3.48) 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

$$\Delta \boldsymbol{w} \ = \ \eta \sum_{i=1;\ a^i y^i = -1}^{l} y^i \ \boldsymbol{x}^i \tag{5.11}$$

or in an on-line formulation

$$\text{if } a\ y \ = \ -1 : \ \ \Delta \boldsymbol{w} \ = \ \eta\ y\ \boldsymbol{x} \ . \tag{5.12}$$

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. 5.6 depicts the perceptron update rule.

Assume that the problem is linearly separable. The perceptron learning rule converges after finite many steps to a solution which classifies all training examples correctly.

However an arbitrary solution out of all possible solutions is chosen. If the problem in not linearly separable then the algorithms does not stop.

Minsky and Papert showed 1969 that many problems cannot be solved by the perceptron. For example the XOR problem is a nonlinear problem which cannot be solved by the perceptron.

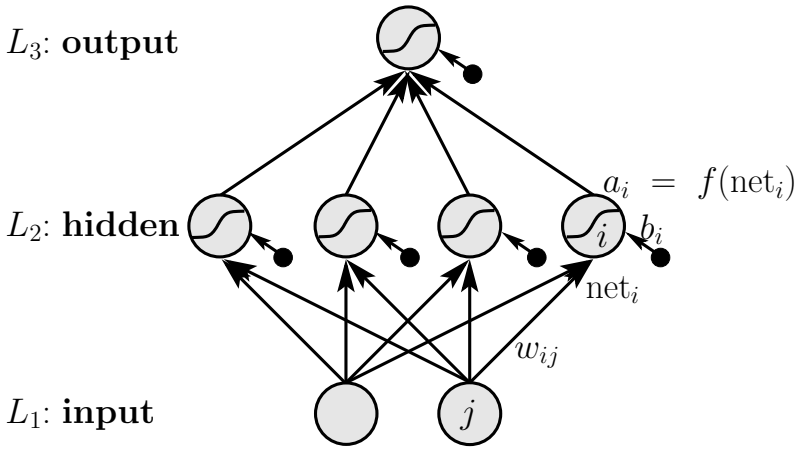


Figure 5.7: Figure of an MLP.

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.

5.4 Multi-Layer Perceptron

With the work of Rumelhart et al. [1986b,a] neural networks and especially the multi-layer perceptron trained with back-propagation became very popular.

5.4.1 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. 5.7):

- 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
- 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*.

- M: 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

$$\mathrm{net}_i \;=\; \sum_{j=0}^{N} w_{ij}\, a_j \tag{5.13}$$

- f: *activation function* with

$$a_i \;=\; f(\mathrm{net}_i) \tag{5.14}$$

  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:

$$i \in L_\nu \text{ and } j \in L_{\nu'} \text{ and } \nu' \;\leq\; \nu \;\Rightarrow\; \boldsymbol{w}_{ij} \;=\; 0\,. \tag{5.15}$$

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 node i in hidden or output layer ν with $\nu > 1$

$$\forall_{i \in L_\nu}:\;\; \mathrm{net}_i \;=\; \sum_{j; j \in L_{\nu-1}}^{N} w_{ij}\, a_j\,. \tag{5.16}$$

Connections between units in layers which are not adjacent are called *shortcut connections*.

The forward pass of a neural network is given in Alg. 5.1.

Activation Functions.

The commonly used activation functions are sigmoid functions.

The logistic function is

$$f(a) \;=\; \frac{1}{1 \;+\; \exp(-\,a)} \tag{5.17}$$

and the $\tanh$ activation function is

$$f(a) \;=\; \tanh(a) \;=\; \frac{\exp(a) \;-\; \exp(-\,a)}{\exp(a) \;+\; \exp(-\,a)}\,. \tag{5.18}$$

Algorithm 5.1 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

Both functions are equivalent because

$$\frac{1}{2}\left(\tanh(a/2) \; + \; 1\right) \; = \; \frac{1}{1 \; + \; \exp(-\, a)} \; . \tag{5.19}$$

(proof as exercise)

That means through weight scaling and through adjusting the bias weights the networks with logistic function 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 an 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 to regions which are not explored in the training phase.

Higher order units.

Higher order units do not use a linear net_i. For example second order units have the form

$$\mathrm{net}_i \; = \sum_{(j_1,j_2)=(0,0)}^{N} w_{ij_1j_2} \; a_{j_1} \; a_{j_2} \; . \tag{5.20}$$

Note that linear and constant terms are considered because of the bias unit $a_0 \; = \; 1$.

We will use higher order units in Section 5.6.6 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.

5.4.2 Universality

In [Funahashi, 1989] it is proven that MLPs are universal function approximators:

Theorem 5.1 (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 arbitrarily exact through a three-layered neural network $o(x; \boldsymbol{w})$:*

$$\max_{x \in K} |o(x; \boldsymbol{w}) \; - \; g(x)| \; < \; \epsilon \; , \tag{5.21}$$

where ϵ is given and the number of hidden units M depends 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.

5.4.3 Learning and Back-Propagation

To train neural network the gradient based methods 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.

We define the empirical error as

$$R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;=\; \frac{1}{l} \sum_{i=1}^{l} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \;, \tag{5.22}$$

where we generalized to multiple outputs. The gradient descent update is

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w}^{\text{old}} \;-\; \eta \, \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;. \tag{5.23}$$

We obtain

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) \;=\; \frac{1}{l} \sum_{i=1}^{l} \nabla_{\boldsymbol{w}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \;. \tag{5.24}$$

We have to compute

$$\frac{\partial}{\partial w_{kl}} \, L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \tag{5.25}$$

for all w_{kl}.

$$\begin{aligned} &\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \\ &\frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; \frac{\partial \mathrm{net}_k}{\partial w_{kl}} \; = \\ &\frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; a_l \; . \end{aligned} \tag{5.26}$$

We define the δ-error at unit k as

$$\delta_k \; = \; \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \tag{5.27}$$

and obtain

$$\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \; \delta_k \; a_l \; . \tag{5.28}$$

The δ-error at the output units is

$$\begin{aligned} &\forall_{1 \leq s \leq O} : \quad \delta_{N-O+s} \; = \\ &\frac{\partial}{\partial g_s} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; f'(\mathrm{net}_{N-O+s}) \; = \\ &\frac{\partial L}{\partial a_{N-O+s}} f'(\mathrm{net}_{N-O+s}) \; , \end{aligned} \tag{5.29}$$

where f' denotes the derivative of the activation function f.

The δ-error at units not in the output layer is

$$\begin{aligned} &\delta_k \; = \; \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; = \\ &\sum_n \frac{\partial}{\partial \mathrm{net}_n} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \; \frac{\partial \mathrm{net}_n}{\partial \mathrm{net}_k} \; = \\ &\\ &f'(\mathrm{net}_k) \; \sum_n \delta_n \; w_{nk} \; , \end{aligned} \tag{5.30}$$

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. 5.8 depicts the back-propagation algorithm for a 4-layer feed-forward network.

Note that for the logistic function

$$f(a) \; = \; \frac{1}{1 \; + \; \exp(-\, a)} \tag{5.31}$$

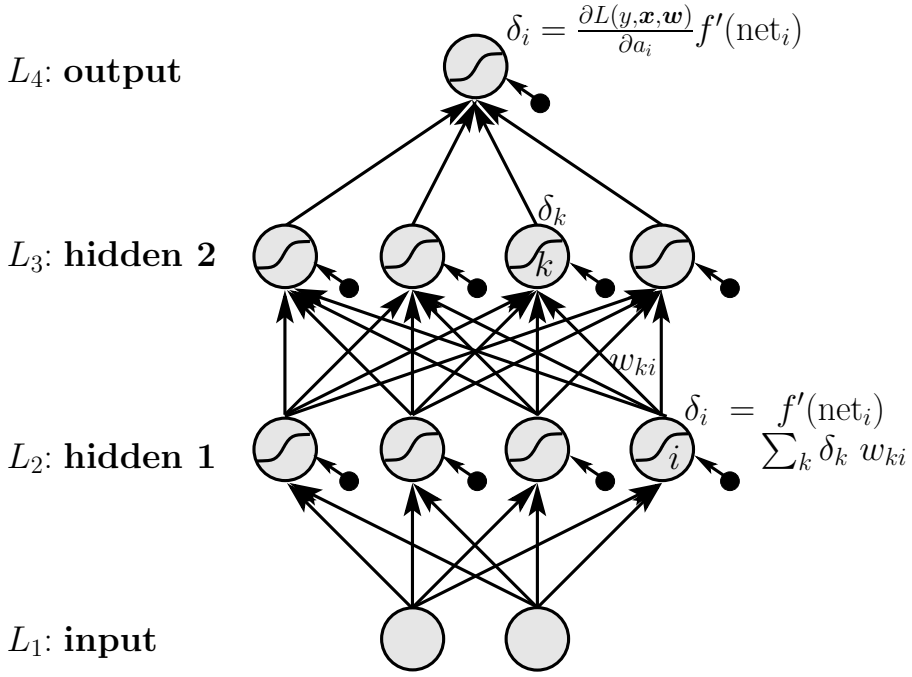


Figure 5.8: 4-layer MLP where the back-propagation algorithm is depicted. The $\delta_k = \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right)$ are computed from the δ's of the higher layer. The back-propagation algorithm starts from the top layer and ends with the layer 2 where the weights to layer 2 are updated.

the derivative is

$$f'(a) \;=\; f(a)\;(1 \;-\; f(a))\;. \tag{5.32}$$

This speeds up the algorithm because the exponential function must only be evaluated once.

Alg. 5.2 gives the backward pass for a single example where the weight update is accumulated in Δw_{ij}.

Algorithm 5.2 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 Alg. 5.2 has complexity of $O(W)$ which is very efficient.

5.4.4 Hessian

The Hessian matrix is useful for training and regularizing neural networks. The Hessian is defined as

$$H_{lj} \;=\; \frac{\partial^2 R(\boldsymbol{w})}{\partial w_l \, \partial w_j} \,. \tag{5.33}$$

For most risk functions it is sufficient to calculate the Hessian of the loss function for one example to compute the Hessian of the risk

$$H_{ij}(y^i, \boldsymbol{x}^i, \boldsymbol{w}) \;=\; \frac{\partial^2 L(y^i, \boldsymbol{x}^i, \boldsymbol{w})}{\partial w_l \, \partial w_j} \,. \tag{5.34}$$

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 use the Hessian or use characteristics of the Hessian to optimize the step-size.

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 5.4.5.3). In other applications the Hessian is used to assign error bars to the outputs or to determine regularization parameters or to compare or average over models.

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. 5.26 we have

$$\frac{\partial^2 L(\boldsymbol{w})}{\partial w_{ij}^2} \;=\; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_i^2} \; a_j^2 \,. \tag{5.35}$$

Further the chain rule gives

$$\begin{aligned} &\frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_i^2} \;= \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (5.36)\\ &\big(f'(\mathrm{net}_i)\big)^2 \sum_l \sum_k \; w_{lj} \; w_{kj} \; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_l \, \partial \mathrm{net}_k} \;+\; f''(\mathrm{net}_i) \sum_l \frac{\partial L\,(\boldsymbol{w})}{\partial \mathrm{net}_l} \;\approx \\ &\big(f'(\mathrm{net}_i)\big)^2 \sum_l \; w_{lj}^2 \; \frac{\partial^2 L\,(\boldsymbol{w})}{\partial \mathrm{net}_l^2} \;+\; f''(\mathrm{net}_i) \sum_l \frac{\partial L\,(\boldsymbol{w})}{\partial \mathrm{net}_l} \,, \end{aligned}$$

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 values.

Outer product / Levenberg-Marquardt Approximation.

The Levenberg-Marquardt algorithm approximates the Hessian through

$$H_{ij,kl} = \frac{\partial^2 R}{\partial w_{ij}\,\partial w_{kl}} = \sum_{i=1}^{l}\left(\frac{\partial e^i}{\partial w_{ij}}\,\frac{\partial e^i}{\partial w_{kl}} + e^i\,\frac{\partial^2 e^i}{\partial w_{ij}\,\partial w_{kl}}\right) \approx \sum_{i=1}^{l}\frac{\partial a_N(\boldsymbol{x}^i)}{\partial w_{ij}}\,\frac{\partial a_N(\boldsymbol{x}^i)}{\partial w_{kl}}\,, \tag{5.37}$$

where

$$e^i = (a_N(\boldsymbol{x}^i;\boldsymbol{w}) - y^i) \tag{5.38}$$

and

$$R(\boldsymbol{w}) = \sum_{i=1}^{l}\left(e^i(\boldsymbol{w})\right)^2\,. \tag{5.39}$$

Attention: that only holds for squared error!

Inverse Hessian Approximation.

We needed the inverse Hessian in Section 5.4.5.3 which can be approximated by an outer product approximation.

If $\boldsymbol{g}^i$ is the weight gradient for example $\boldsymbol{x}^i$ then

$$\boldsymbol{H} = \sum_{i=1}^{l}\boldsymbol{g}^i\,\left(\boldsymbol{g}^i\right)^T\,. \tag{5.40}$$

This means the Hessian can be build sequentially

$$\boldsymbol{H}^k = \sum_{i=1}^{k}\boldsymbol{g}^i\,\left(\boldsymbol{g}^i\right)^T\,, \tag{5.41}$$

where $\boldsymbol{H}^l = \boldsymbol{H}$.

We obtain

$$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k + \boldsymbol{g}^{k+1}\left(\boldsymbol{g}^{k+1}\right)^T\,. \tag{5.42}$$

To this formula the matrix inversion lemma

$$(\boldsymbol{A} + \boldsymbol{B}\,\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{I} + \boldsymbol{C}\,\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\,\boldsymbol{A}^{-1} \tag{5.43}$$

is applied which gives

$$\left(\boldsymbol{H}^{k+1}\right)^{-1} = \left(\boldsymbol{H}^k\right)^{-1} - \frac{\left(\boldsymbol{H}^k\right)^{-1}\boldsymbol{g}^{k+1}\left(\boldsymbol{g}^{k+1}\right)^T\left(\boldsymbol{H}^k\right)^{-1}}{1 + \left(\boldsymbol{g}^{k+1}\right)^T\left(\boldsymbol{H}^k\right)^{-1}\boldsymbol{g}^{k+1}}\,. \tag{5.44}$$

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 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 a complexity of $O(W^3)$.

The finite differences applied to the gradient gives

$$H_{ij,kl}\ \approx\ \frac{1}{2\,\epsilon}\left(\frac{\partial R}{\partial w_{ij}}(w_{kl}\ +\ \epsilon)\ -\ \frac{\partial R}{\partial w_{ij}}(w_{kl}\ -\ \epsilon)\right)\ . \tag{5.45}$$

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 had 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. (5.27) and eq. (5.28) we have

$$\begin{aligned}
\delta_k\ &=\ \frac{\partial}{\partial \mathrm{net}_k}\ L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right) \\
\frac{\partial}{\partial w_{kl}}\ L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\boldsymbol{x}^i; \boldsymbol{w}\right)\right)\ &=\ \delta_k\ a_l\ .
\end{aligned} \tag{5.46}$$

$$\begin{aligned}
H_{ij,kl}\ &=\ \frac{\partial^2 L}{\partial w_{ij}\ \partial w_{kl}}\ = \\
a_j\ \frac{\partial}{\partial \mathrm{net}_i}\frac{\partial L}{\partial w_{kl}}\ &=\ a_j\ \frac{\partial}{\partial \mathrm{net}_i}\left(\delta_k\ a_l\right)\ = \\
\delta_k\ a_j\ f'(\mathrm{net}_l)\ \frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_i}\ &+\ a_j\ a_l\ \frac{\partial \delta_k}{\partial \mathrm{net}_i}\ = \\
\delta_k\ a_j\ f'(\mathrm{net}_l)\ V_{li}\ &+\ \ a_j\ a_l\ b_{ki}\ ,
\end{aligned} \tag{5.47}$$

where we used the definitions

$$\begin{aligned}
V_{li}\ &=\ \frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_i} \\
b_{ki}\ &=\ \frac{\partial \delta_k}{\partial \mathrm{net}_i}\ .
\end{aligned} \tag{5.48}$$

The V_{li} are computed by forward propagation through

$$V_{li} \;=\; \sum_{n:w_{ln}} \frac{\partial \mathrm{net}_l}{\partial \mathrm{net}_n} \frac{\partial \mathrm{net}_n}{\partial \mathrm{net}_i} \;=\; \sum_{n:w_{ln}} f'(\mathrm{net}_l)\; w_{ln}\; V_{ni}\;. \tag{5.49}$$

Note, that for input units the V_{li} are not required, so that the forward propagation can be initialized by

$$V_{ii} \;=\; 1 \tag{5.50}$$

and for i in a higher than or the same layer as l set

$$V_{li} \;=\; 0\;. \tag{5.51}$$

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. (5.30):

$$\delta_k \;=\; f'(\mathrm{net}_k) \sum_n \delta_n\; w_{nk} \tag{5.52}$$

which leads to

$$\begin{aligned} b_{ki} \;&=\; \frac{\partial}{\partial \mathrm{net}_i} f'(\mathrm{net}_k) \sum_n \delta_n\; w_{nk} \;= \\ & f''(\mathrm{net}_k)\; V_{ki} \sum_n \delta_n\; w_{nk} \;+\; f'(\mathrm{net}_k) \sum_n w_{nk}\; b_{ni}\;. \end{aligned} \tag{5.53}$$

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

$$b_{ki} \;=\; \sum_n S_{kn}\; V_{ni} \;=\; [\boldsymbol{S}\boldsymbol{V}_i]_k \;=\; [\boldsymbol{S}\boldsymbol{V}]_{ki}\;, \tag{5.54}$$

where

$$S_{kj} \;=\; \frac{\partial^2 L}{\partial \mathrm{net}_k\; \partial \mathrm{net}_j}\;. \tag{5.55}$$

The initial conditions for the backward pass are for output unit i

$$\delta_i \;=\; \frac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f'(\mathrm{net}_i) \tag{5.56}$$

and for i and j output units (where output units are not interconnected):

$$S_{ij} \;=\; \frac{\partial^2 L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i\; \partial a_j} f'(\mathrm{net}_i)\; f'(\mathrm{net}_j)\;. \tag{5.57}$$

Algorithm 5.3 Hessian Computation

forward pass according to Alg. 5.1
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. 5.2
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

The algorithm Alg. 5.3 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 algorithms 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 [Møller, 1993]) we define a differential operator as

$$\mathcal{R}\{.\} \;=\; \boldsymbol{v}^T \nabla_{\boldsymbol{w}} \tag{5.58}$$

in order to derive an efficient way to compute

$$\boldsymbol{v}^T \boldsymbol{H} \;=\; \mathcal{R}\{\nabla_{\boldsymbol{w}} L\} \;=\; \boldsymbol{v}^T \nabla^2_{\boldsymbol{w}} L \;. \tag{5.59}$$

We have

$$\mathcal{R}\{\boldsymbol{w}\} \;=\; \boldsymbol{v}^T \tag{5.60}$$

and

$$\mathcal{R}\{a_i\} \;=\; \mathbf{0} \tag{5.61}$$

for a_i input unit.

We obtain

$$\mathcal{R}\{a_i\} \;=\; f'(\mathrm{net}_i)\, \mathcal{R}\{\mathrm{net}_i\} \tag{5.62}$$

and

$$\mathcal{R}\{\mathrm{net}_i\} \;=\; \sum_j w_{ij} \mathcal{R}\{a_j\} \;+\; \sum_j v_{ij}\, a_j \;, \tag{5.63}$$

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

$$\delta_i \;=\; \frac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f'(\mathrm{net}_i) \tag{5.64}$$

which gives

$$\begin{aligned} &\mathcal{R}\{\delta_i\} \;= \\ &\left(\frac{\partial^2 L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i^2} \left(f'(\mathrm{net}_i)\right)^2 \;+\; \frac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f''(\mathrm{net}_i) \right) \mathcal{R}\{\mathrm{net}_i\} \end{aligned} \tag{5.65}$$

For non-output units i we have

$$\begin{aligned} \mathcal{R}\{\delta_i\} \;=\; & f''(\mathrm{net}_i) \mathcal{R}\{\mathrm{net}_i\} \sum_k w_{ki}\, \delta_k \;+ \\ & f'(\mathrm{net}_i) \sum_k v_{ki}\, \delta_k \;+\; f'(\mathrm{net}_i) \sum_k w_{ki}\, \mathcal{R}\{\delta_k\} \end{aligned} \tag{5.66}$$

The Hessian can be computed as

$$\mathcal{R}\left\{\frac{\partial L}{\partial w_{ij}}\right\} \;=\; \mathcal{R}\{\delta_i\, a_j\} \;=\; \mathcal{R}\{\delta_i\}\, a_j \;+\; \mathcal{R}\{a_j\}\delta_i\;, \tag{5.67}$$

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. 5.3 is more efficient.

5.4.5 Regularization

The trade-off between complexity and training error (empirical risk) is given by the bound on the risk:

$$R \;\leq\; R_{\text{emp}} \;+\; \text{complexity}\;. \tag{5.68}$$

Increasing complexity leads to smaller training error but the test error, the risk, increases at some point.

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

Algorithm 5.4 Hessian-Vector Multiplication

forward pass according to Alg. 5.1
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**

$$\begin{aligned}\mathcal{R}\{\mathrm{net}_i\} \;&=\; \sum_j w_{ij}\mathcal{R}\{a_j\} \;+\; \sum_j v_{ij}\, a_j \\ \mathcal{R}\{a_i\} \;&=\; f'(\mathrm{net}_i)\mathcal{R}\{\mathrm{net}_i\}\end{aligned}$$

end for
end for
END Hessian-Vector Forward Pass
backward pass according to Alg. 5.2
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**

$$\begin{aligned}\mathcal{R}\{\delta_i\} \;&=\; f''(\mathrm{net}_i)\mathcal{R}\{\mathrm{net}_i\}\sum_k w_{ki}\;\delta_k \;+ \\ &f'(\mathrm{net}_i)\sum_k v_{ki}\;\delta_k \;+\; f'(\mathrm{net}_i)\sum_k w_{ki}\;\mathcal{R}\{\delta_k\}\end{aligned}$$

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

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 increases 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 of 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.

5.4.5.1 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 to 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 a 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 the earlier the learning is stopped the more linear the network function will be and the lower is the complexity.

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.

5.4.5.2 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 obtained 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

$$C_k \;=\; \sum_{j=N-O+1}^{N} \left| \sum_{i=1}^{l} (a_k \;-\; \bar{a}_k)\;(\epsilon_j \;-\; \bar{\epsilon}_j) \right| \;, \tag{5.69}$$

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

$$\frac{\partial C_k}{\partial w_{kj}} \;=\; \sum_{j=N-O+1}^{N} \pm \sum_{i=1}^{l} (\epsilon_j \;-\; \bar{\epsilon}_j)\;\; f'(\mathrm{net}_k)\; a_j \;, \tag{5.70}$$

where the sign is given by the sign of the correlation in eq. (5.69).

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 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. 5.9 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.

5.4.5.3 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 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.

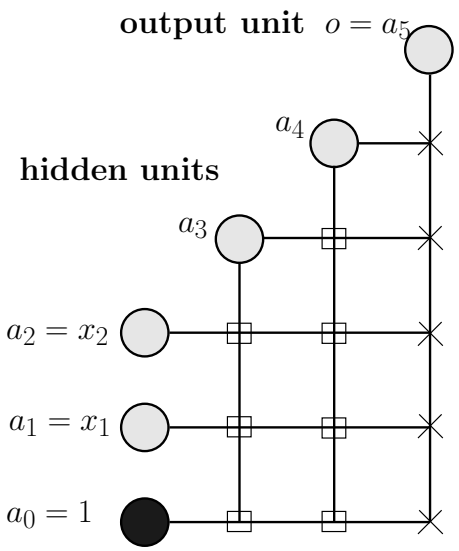


Figure 5.9: Cascade-correlation: architecture of the network. Squares are weights which are trained whereas crosses represent weights which retrained only after the addition of a hidden unit. First the hidden unit 3 with activation a_3 was added and then hidden unit 4 with activation a_4.

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 unit 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. The gradient vanishes in the minimum $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) = \mathbf{0}$, therefore we obtain with $\Delta \boldsymbol{w} = (\boldsymbol{w} - \boldsymbol{w}^*)$

$$R(\boldsymbol{w}) = R(\boldsymbol{w}^*) + \frac{1}{2} \Delta \boldsymbol{w}^T \boldsymbol{H}(\boldsymbol{w}^*) \, \Delta \boldsymbol{w} + O\left((\Delta \boldsymbol{w})^3\right) , \tag{5.71}$$

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

$$\boldsymbol{e}_i^T \, \Delta \boldsymbol{w} + w_i = 0 \, , \tag{5.72}$$

where $\boldsymbol{e}_i$ is the unit vector with a 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. (5.36). The error increase is determined by

$$\sum_i H_{ii} \, (\Delta w_i)^2 \, . \tag{5.73}$$

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 $\boldsymbol{v}$

$$\boldsymbol{v}^T \boldsymbol{H}\, \boldsymbol{v} \;\geq\; 0\,, \tag{5.74}$$

therefore

$$\boldsymbol{e}_i^T \boldsymbol{H}\, \boldsymbol{e}_i \;=\; H_{ii} \;\geq\; 0\,. \tag{5.75}$$

Therefore the vector $\Delta \boldsymbol{w}$ is chosen as

$$\Delta \boldsymbol{w} \;=\; -\, w_i\, \boldsymbol{e}_i \tag{5.76}$$

so that all components except the i-th are zero. The minimal error increase is given by

$$\min_i H_{ii}\, w_i^2 \tag{5.77}$$

and the weight w_k with

$$k \;=\; \arg\min_i H_{ii}\, w_i^2 \tag{5.78}$$

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 perform the same tasks which can be done by one weight alone: assume each of the two weights has a 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 or at least be reduced.

Above Taylor expansion and the constraint of removing weight i can be stated as a quadratic optimization problem

$$\begin{aligned} &\min_{\Delta \boldsymbol{w}} \;\frac{1}{2}\, \Delta \boldsymbol{w}^T \boldsymbol{H}\, \Delta \boldsymbol{w} \\ &\text{s.t. } \boldsymbol{e}_i^T\, \Delta \boldsymbol{w} \;+\; w_i \;=\; 0 \end{aligned} \tag{5.79}$$

The Lagrangian is

$$L \;=\; \frac{1}{2}\, \Delta \boldsymbol{w}^T \boldsymbol{H}\, \Delta \boldsymbol{w} \;+\; \alpha \left(\boldsymbol{e}_i^T\, \Delta \boldsymbol{w} \;+\; w_i \right)\,. \tag{5.80}$$

The derivative has to be zero

$$\frac{\partial L}{\partial \Delta \boldsymbol{w}} \;=\; \boldsymbol{H}\, \Delta \boldsymbol{w} \;+\; \alpha\, \boldsymbol{e}_i \;= \boldsymbol{0} \tag{5.81}$$

which is

$$\Delta \boldsymbol{w} \;=\; -\,\alpha\, \boldsymbol{H}^{-1}\, \boldsymbol{e}_i \tag{5.82}$$

Further we have

$$w_i \;=\; -\,\boldsymbol{e}_i^T\, \Delta \boldsymbol{w} \;=\; \alpha\, \boldsymbol{e}_i^T \boldsymbol{H}^{-1}\, \boldsymbol{e}_i \;=\; \alpha\, \boldsymbol{H}_{ii}^{-1} \tag{5.83}$$

which gives

$$\alpha \;=\; \frac{w_i}{\boldsymbol{H}_{ii}^{-1}} \tag{5.84}$$

and results in

$$\Delta \boldsymbol{w} \;=\; -\,\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\, \boldsymbol{H}^{-1}\, \boldsymbol{e}_i\;. \tag{5.85}$$

The objective is

$$\frac{1}{2}\, \Delta \boldsymbol{w}^T \boldsymbol{H}\, \Delta \boldsymbol{w} \;=\; \frac{1}{2} \frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}}\;, \tag{5.86}$$

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

$$\frac{1}{2} \frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}} \tag{5.87}$$

and the correction is done by

$$\Delta \boldsymbol{w} \;=\; -\,\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\, \boldsymbol{H}^{-1}\, \boldsymbol{e}_i\;. \tag{5.88}$$

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 5.4.4.

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 $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) \;=\; \boldsymbol{0}$; 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. $\gamma = 0.5$), 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 problem. Therefore some structures are distributed in a larger network which avoids to prune the network to an optimal size.

5.4.5.4 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:

$$R(\boldsymbol{w}) \;=\; R_{\text{emp}}(\boldsymbol{w}) \;+\; \lambda\, \Omega(\boldsymbol{w})\;. \tag{5.89}$$

Gradient descent is now performed on this extended error term.

$$\nabla_{\boldsymbol{w}} R(\boldsymbol{w}) \;=\; \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}) \;+\; \lambda\, \nabla_{\boldsymbol{w}} \Omega(\boldsymbol{w})\;. \tag{5.90}$$

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 keep 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, a term based on $\log\left(1 \;+\; \boldsymbol{w}^T\boldsymbol{w}\right)$ [Williams, 1994], or the 2-norm $\|\boldsymbol{w}\|^2 \;=\; \boldsymbol{w}^T\boldsymbol{w}$ [Krogh and Hertz, 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

$$\Omega(\boldsymbol{w}) \;=\; \sum_i \frac{w_i^2/w_0^2}{1 \;+\; w_i^2/w_0^2}\;, \tag{5.91}$$

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.

5.4.5.5 Training with Noise

To regularize and to avoid that the network extracts individual characteristics of few training examples either noise is added to the training examples 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 algorithm 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.

Denoising autoassociators Vincent et al. [2008], Vincent [2011] are autoassociator for which the input is corrupted by noise. However the full uncorrupted input must be restored at the output. The goal is to learn to reconstruct the clean input from a corrupted version. The corruption may be additive isotropic Gaussian noise, salt and pepper noise for gray-scale images, or masking noise (salt or pepper only).

5.4.5.6 Dropout

Recently a new technique for training with noise was introduced: **dropout** Hinton et al. [2012], Krizhevsky et al. [2012], Goodfellow et al. [2013].

To be extended.

5.4.5.7 Weight Sharing

Nowlan 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

$$G(\sigma_j, \mu_j) \;=\; \frac{1}{(2\,\pi)^{1/2}\;\sigma_j}\;\exp\left(-\frac{1}{2\,\sigma_j^2}\;(w\;-\;\mu_j)^2\right)\;. \tag{5.92}$$

Each group j has a prior size α_j which estimates what percentage of weights will belong to this group.

We obtain for the probability of a weight belonging to group j

$$p(w \mid j) \;=\; \frac{\alpha_j\;G(\sigma_j, \mu_j)}{\sum_k \alpha_k\;G(\sigma_k, \mu_k)} \tag{5.93}$$

and the probability of observing this weight value

$$p(w) \;=\; \sum_j p(w \mid j)\;. \tag{5.94}$$

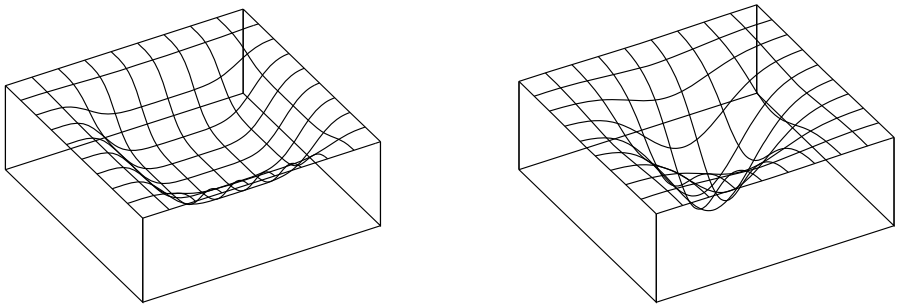


Figure 5.10: Left: example of a flat minimum. Right: example of a steep minimum.

The value $p(w)$ should be maximized and we obtain for the derivative of $R(\boldsymbol{w})$ with respect to a single weight

$$\frac{\partial R(\boldsymbol{w})}{\partial w_i} \;=\; \frac{\partial R_{\text{emp}}(\boldsymbol{w})}{\partial w_i} \;+\; \lambda \sum_j p(w_i \mid j)\, \frac{(w_i \;-\; \mu_j)}{\sigma_j^2}\,. \tag{5.95}$$

Also the centers μ_j and the variances σ_j^2 can be adaptively adjusted during learning.

5.4.5.8 Flat Minimum Search

Another algorithm for regularization during training is the "Flat Minimum Search" (FMS) algorithms [Hochreiter and Schmidhuber, 1997h,e,f,i,b,d,a]. 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. 5.10).

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 given 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 it 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. 5.4.

The FMS error term is

$$\Omega(\boldsymbol{w}) \;=\; \frac{1}{2}\left(-L\log\epsilon + \sum_{i,j}\log\sum_{k=N-O+1}^{N}\left(\frac{\partial a_k}{\partial w_{ij}}\right)^2 + W\log\sum_{k=N-O+1}^{N}\left(\sum_{i,j}\frac{\left|\frac{\partial a_k}{\partial w_{ij}}\right|}{\sqrt{\sum_{k=N-O+1}^{N}(\frac{\partial a_k}{\partial w_{ij}})^2}}\right)^2\right), \tag{5.96}$$

where ϵ gives the tolerable output change as Euclidian distance (output changes below ϵ are considered to be equal).

The derivative is

$$\frac{\partial\Omega(\boldsymbol{w})}{\partial w_{uv}} \;=\; \sum_{k=N-O+1}^{N}\sum_{i,j}\frac{\partial\Omega(\boldsymbol{w})}{\partial\left(\frac{\partial a_k}{\partial w_{ij}}\right)}\frac{\partial^2 a_k}{\partial w_{ij}\,\partial w_{uv}} \tag{5.97}$$

which is with $\frac{\partial a_k}{\partial w_{ij}}$ as new variables

$$\nabla_{\boldsymbol{w}}\Omega(\boldsymbol{w}) \;=\; \sum_{k=N-O+1}^{N}\boldsymbol{H}^k\left(\nabla_{\frac{\partial a_k}{\partial w_{ij}}}\Omega(\boldsymbol{w})\right), \tag{5.98}$$

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).

Similar to weight decay FMS prefers 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 (incoming weights can be given with low precision).

5.4.5.9 Regularization for Structure Extraction

Regularization of neural networks can be used to extract structures from data.

The idea is to build an *auto-associator* network, where the input and the output should be identical (see Fig. 5.11). 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.

However using regularization the identity must be built 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. 5.12 for an example. A good coding strategy would be to code bars and not the single pixels.

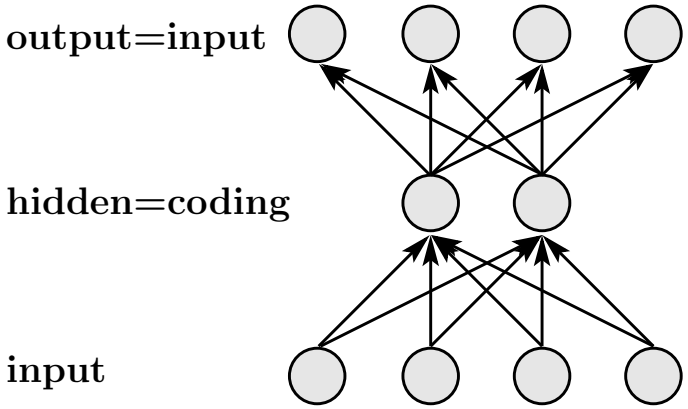
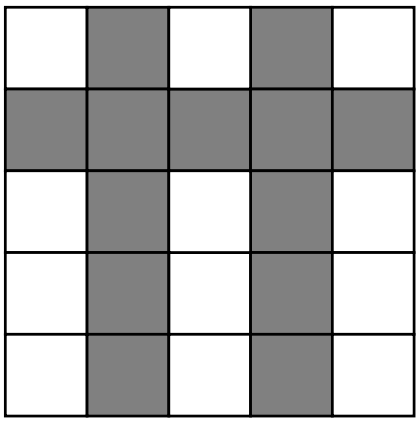


Figure 5.11: An auto-associator network where the output must be identical to the input. The hidden layer must code the input information to supply it to the output layer.



-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 5.12: 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.

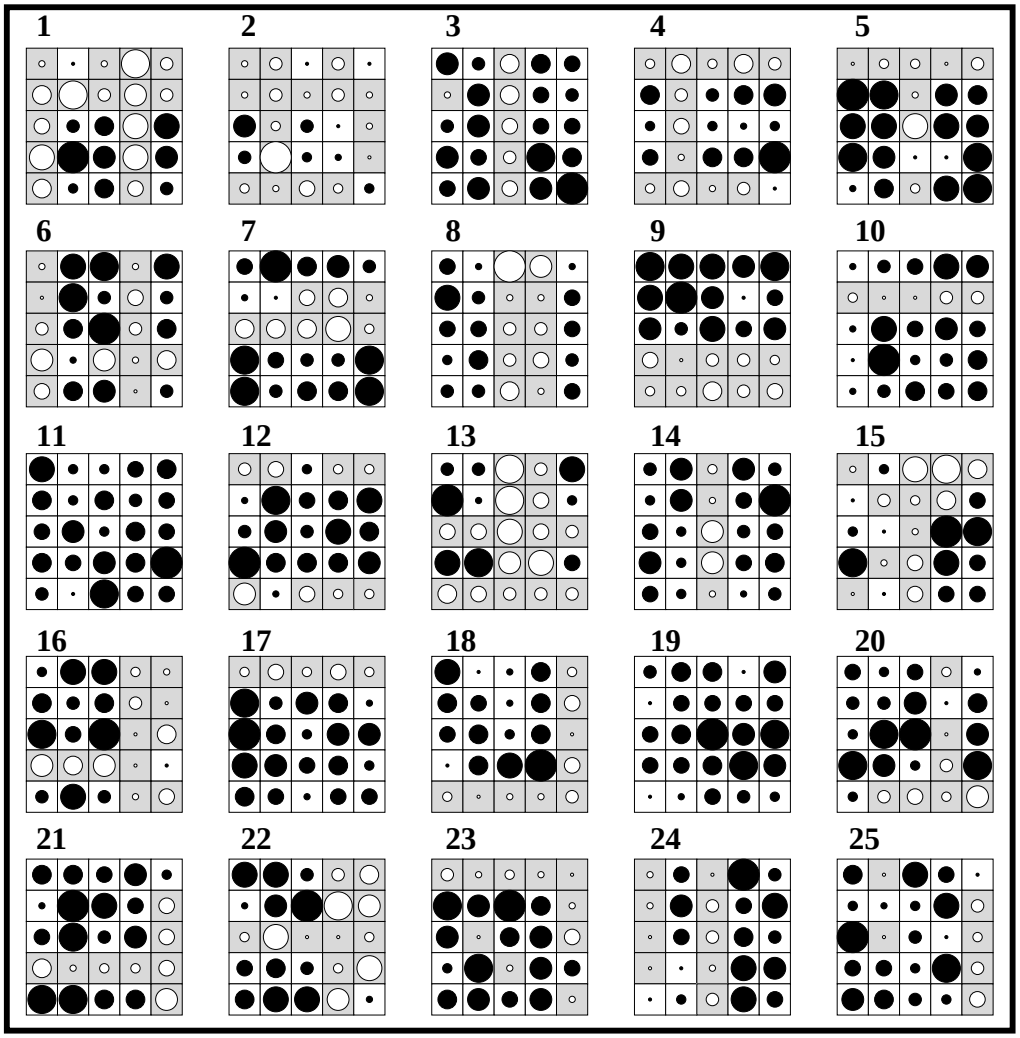


Figure 5.13: 25 examples for noise training examples of the bars problem where each example is a 5×5 matrix. The white circles are positive and the black circles are negative values. The radius of the circle is proportional to the absolute value. The values are normalized for each 5×5 matrix so that one maximal circle is present.

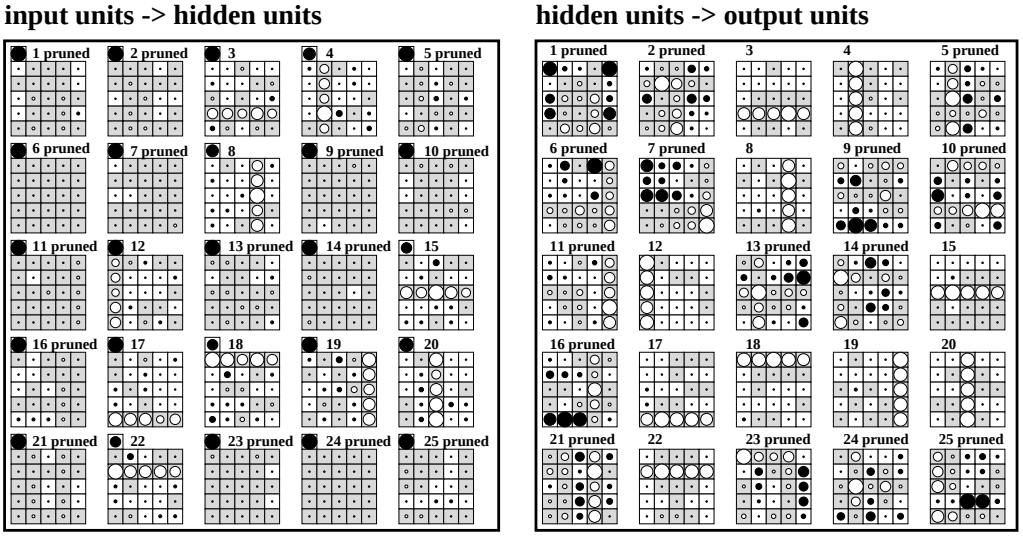


Figure 5.14: Noise bars results for FMS. Left: weights from inputs to the hidden units. Each 5×5 square represents one hidden unit with its ingoing weights. At the upper left corner the bias weights are given. The color coding is as in Fig. 5.13 but also the bias weights is used for normalization. Units which are marked by "pruned" have been removed by FMS. Right: weights from hidden units to output units.

To the input noise is added which gives training examples as shown in Fig. 5.13.

The result of the auto-associator trained with FMS is shown in Fig. 5.14. 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. 5.15. The image is black except for special white regions which show buildings or streets. The result is shown in Fig. 5.15. The black background with white structures was extracted by localized white spot detectors.

We analyze the image of wood cells as in Fig. 5.17. Fig. 5.18 shows the results after training with FMS.

We analyze the image of a wood piece with grain as shown in Fig. 5.19. The results after training with FMS are shown in Fig. 5.20.

In all these experiments a regularized auto-associator was able to extract structures in the input data.

5.4.6 Tricks of the Trade

5.4.6.1 Number of Training Examples

According to Baum & Haussler 1989 if from

$$l \geq \frac{W}{\epsilon} \log_2 \left(\frac{N}{\epsilon} \right) \tag{5.99}$$

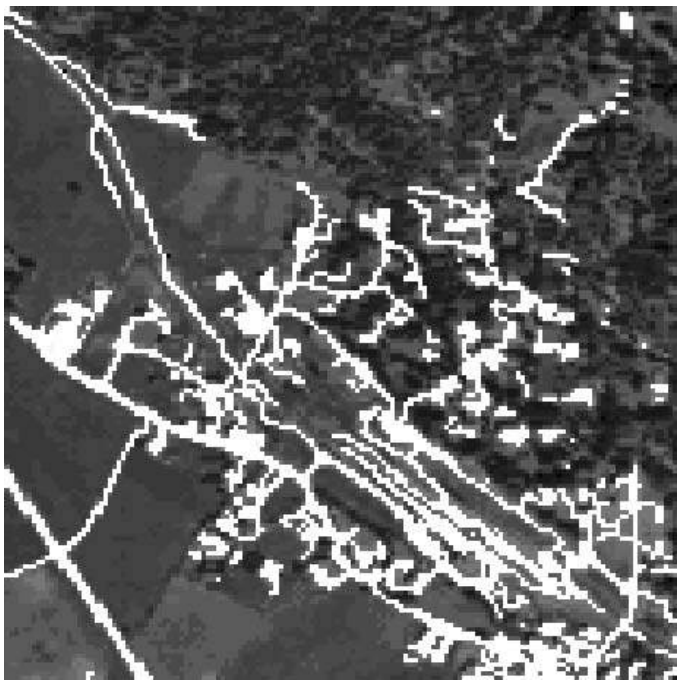


Figure 5.15: An image of a village from air.

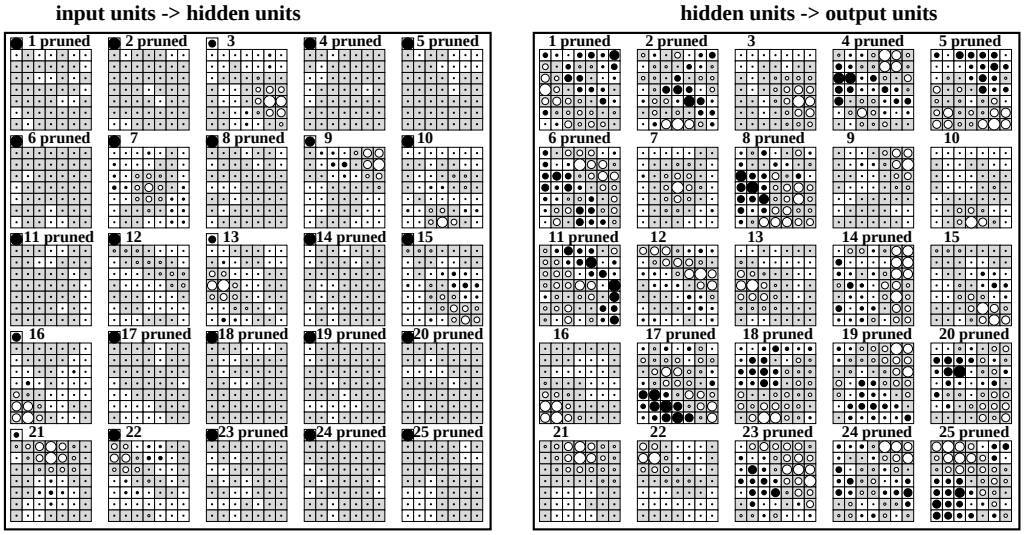


Figure 5.16: Result of FMS trained on the village image. Left: weights from input units to hidden units. Most units are deleted. Right: weights from hidden units to output units.

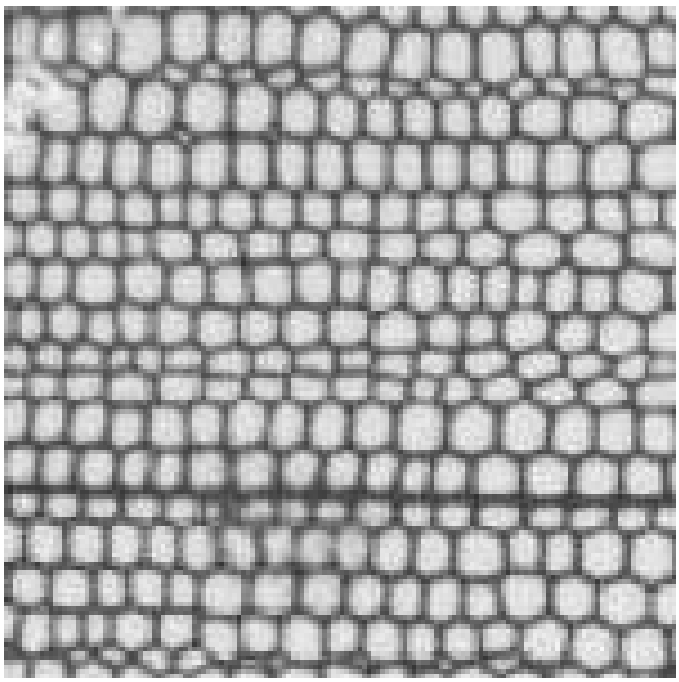


Figure 5.17: An image of wood cells.

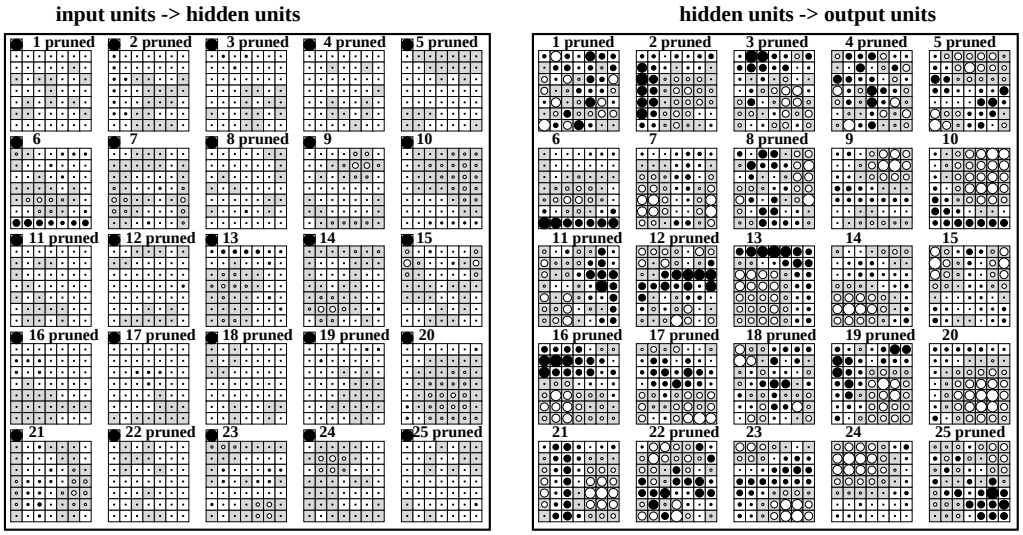


Figure 5.18: Result of FMS trained on the wood cell image. Left: weights from input units to hidden units. Most units are deleted. Right: weights from hidden units to output units.

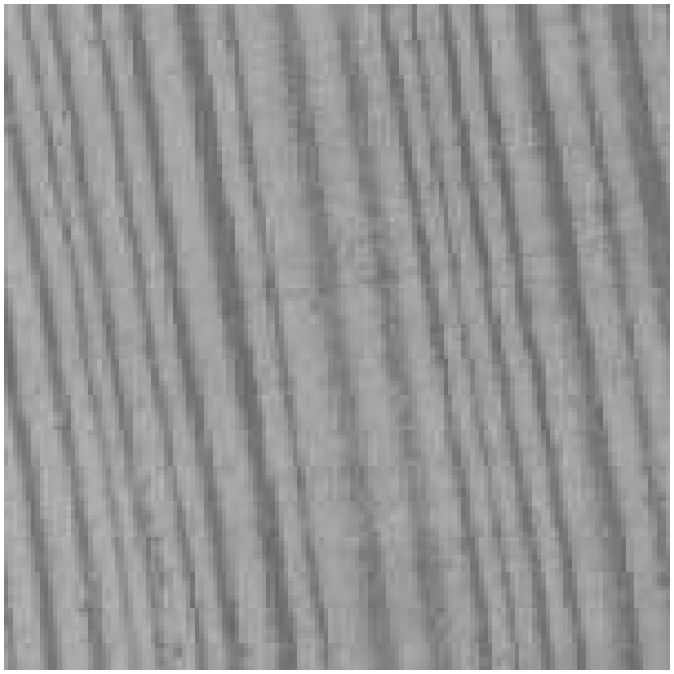


Figure 5.19: An image of a wood piece with grain.

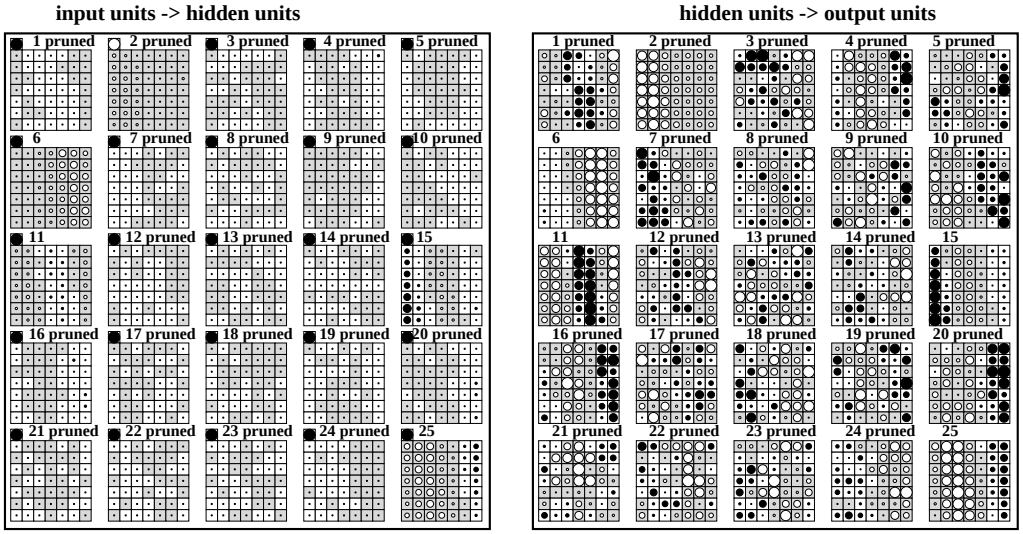


Figure 5.20: Result of FMS trained on the wood piece image. Left: weights from input units to hidden units. Most units are deleted. Right: weights from hidden units to output units.

examples a fraction $(1 - \epsilon/2)$ are correctly classified then a fraction of $(1 - \epsilon)$ of future examples are classified correctly. Here N is the number of units.

Further they found that

$$d_{\mathrm{VC}} \geq (N - 2)\, d\,, \tag{5.100}$$

where d is the number of inputs. For a two-layered network with one output unit we have $d\,(N - d - 1)$ weights from input to the $(N - d - 1)$ hidden units, $(N - d - 1)$ weights from the hidden units to the output, $(N - d - 1)$ hidden bias weights, and one output bias which gives

$$W = (d + 2)\,(N - d - 1) + 1\,. \tag{5.101}$$

Approximatively we have

$$W = N\, d\,. \tag{5.102}$$

For $\epsilon = 0.1$ we have

$$l \geq W/\epsilon = 10\, W\,. \tag{5.103}$$

That amounts to 10 times more examples than weights in the network.

5.4.6.2 Committees

In many applications the performance is more robust if the average over different networks is used.

Let g_j be the function which is represented by the j-th network then as a committee output with C members we obtain

$$g(\boldsymbol{x}) = \frac{1}{C} \sum_{j=1}^{C} g_j(\boldsymbol{x})\,. \tag{5.104}$$

We assume that

$$g_j(\boldsymbol{x}^i) = y^i + \epsilon_j^i\,, \tag{5.105}$$

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

$$\mathrm{E}\left(\left(\frac{1}{C}\sum_{j=1}^{C} g_j(\boldsymbol{x}) - y\right)^2\right) = \mathrm{E}\left(\left(\frac{1}{C}\sum_{j=1}^{C} (g_j(\boldsymbol{x}) - y)\right)^2\right) = \tag{5.106}$$

$$\mathrm{E}\left(\left(\frac{1}{C}\sum_{j=1}^{C} \epsilon_j\right)^2\right) = \frac{1}{C^2}\sum_{j=1}^{C} \mathrm{E}\left(\epsilon_j^2\right) =$$

$$\frac{1}{C}\,\mathrm{E}\left(\epsilon_j^2\right)\,,$$

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 errors 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

$$g(\boldsymbol{x}) \; = \; \sum_{j=1}^{C} \alpha_j \; g_j(\boldsymbol{x}) \tag{5.107}$$

with $\sum_{j=1}^{C} \alpha_j = 1$ is possible. The α_j can be determined by using the correlation matrix.

5.4.6.3 Local Minima

In principle the optimization techniques for neural networks are prone to find 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.

5.4.6.4 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 distributed 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.

5.4.6.5 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.

5.4.6.6 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 an input component positive and negative values express different facts then this should be kept and normalization to zero mean should be avoided.

If an input component contains outliers e.g. extreme large absolute values then these values should be mapped to a maximum or a squashing function should be used. For example if 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 examples except one 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 a unit in the first hidden layer. That means one example overwrites the information of many other examples.

5.4.6.7 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.

5.4.6.8 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 cumulative update signal is not the sum of all absolute update values. The updates for one weight are positive and negative depending on the example.

5.4.6.9 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 5.6.5) 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.

5.4.6.10 Momentum and Weight Decay

The momentum term helps in most cases to speed up the back-propagation algorithm. For regularization the weight decay method is easy to realize.

5.4.6.11 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 criteria include the maximal weight change of the last epochs, the error improvement of the last epochs.

5.4.6.12 Batch vs. On-line

In many application learning can be sped up by using an on-line update, 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.

5.5 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.

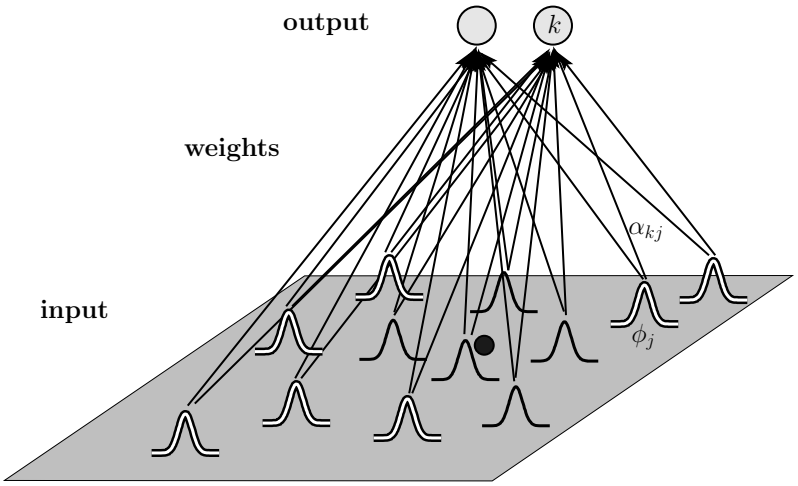


Figure 5.21: A radial basis function network is depicted. Inputs activate the Gaussians according how close the input is to the center. For the current input (dark point) the dark Gaussians are activated.

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

$$g_k(\boldsymbol{x}) \;=\; \sum_{j=1}^{C} \alpha_{kj}\; \phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;+\; \alpha_{k0}\;. \tag{5.108}$$

For example Gaussian radial basis functions are

$$\phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;=\; \exp\left(-\frac{1}{2\;\sigma_j^2}\,\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|^2\right)\;. \tag{5.109}$$

Fig. 5.21 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

$$\phi_j\left(\|\boldsymbol{x} \;-\; \boldsymbol{\mu}_j\|\right) \;=\; \exp\left(-\frac{1}{2}\,(\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)^T\,\boldsymbol{\Sigma}_j^{-1}\;(\boldsymbol{x} \;-\; \boldsymbol{\mu}_j)\right)\;. \tag{5.110}$$

5.5.1 Clustering and Least Squares Estimate

To adjust the parameters of RBF networks there exist different approaches.

First we consider a two-stage procedure, where 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).

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

$$\boldsymbol{Y} = \boldsymbol{\Lambda}\, \boldsymbol{\Phi} \tag{5.111}$$

which can be solved by a least-square estimate if a squared error is used:

$$\boldsymbol{\Lambda}^T = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{Y} \, . \tag{5.112}$$

5.5.2 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

$$\begin{aligned}
\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{y})}{\partial \alpha_{kj}} &= \left(g_k(\boldsymbol{x}) - y_k \right) \phi_j = \left(g_k(\boldsymbol{x}) - y_k \right) \\
&\exp\left(- \frac{1}{2\, \sigma_j^2} \| \boldsymbol{x} - \boldsymbol{\mu}_j \|^2 \right) \\
\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{y})}{\partial \sigma_j} &= \sum_k \left(g_k(\boldsymbol{x}) - y_k \right) \alpha_{kj} \\
&\exp\left(- \frac{1}{2\, \sigma_j^2} \| \boldsymbol{x} - \boldsymbol{\mu}_j \|^2 \right) \frac{\| \boldsymbol{x} - \boldsymbol{\mu}_j \|^2}{\sigma_j^3} \\
\frac{\partial L(\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{y})}{\partial \mu_{jl}} &= \sum_k \left(g_k(\boldsymbol{x}) - y_k \right) \alpha_{kj} \\
&\exp\left(- \frac{1}{2\, \sigma_j^2} \| \boldsymbol{x} - \boldsymbol{\mu}_j \|^2 \right) \frac{(x_l - \mu_{jl})}{\sigma_j^2} \, .
\end{aligned} \tag{5.113}$$

5.5.3 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 points at this region.

5.6 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

$$\boldsymbol{g}(t) \;=\; \boldsymbol{g}\left(\boldsymbol{a}_0, \boldsymbol{x}(1), \ldots, \boldsymbol{x}(t)\right) \;, \tag{5.114}$$

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. 5.22 shows an architecture of a recurrent network and Fig. 5.23 shows how a sequences is processed.

In bioinformatics recurrent networks are however used for sequence processing.

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].

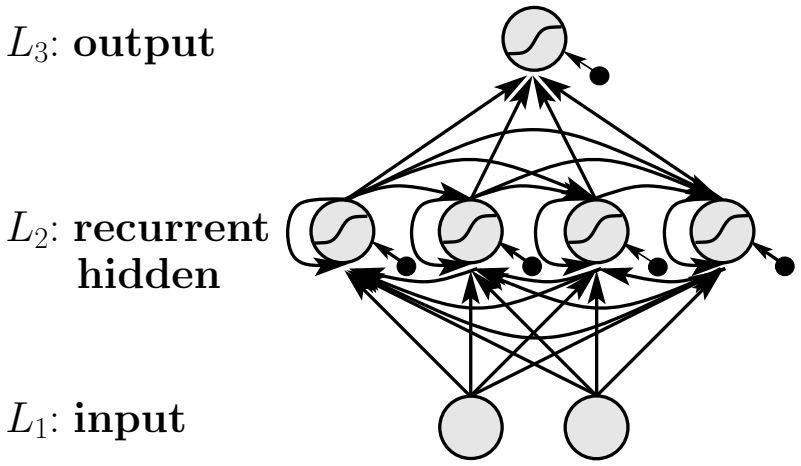


Figure 5.22: An architecture of a recurrent network. The recurrent hidden layer is fully connected, i.e. all units are interconnected. The hidden units are able to store information, i.e. information from previous inputs is kept in the hidden units.

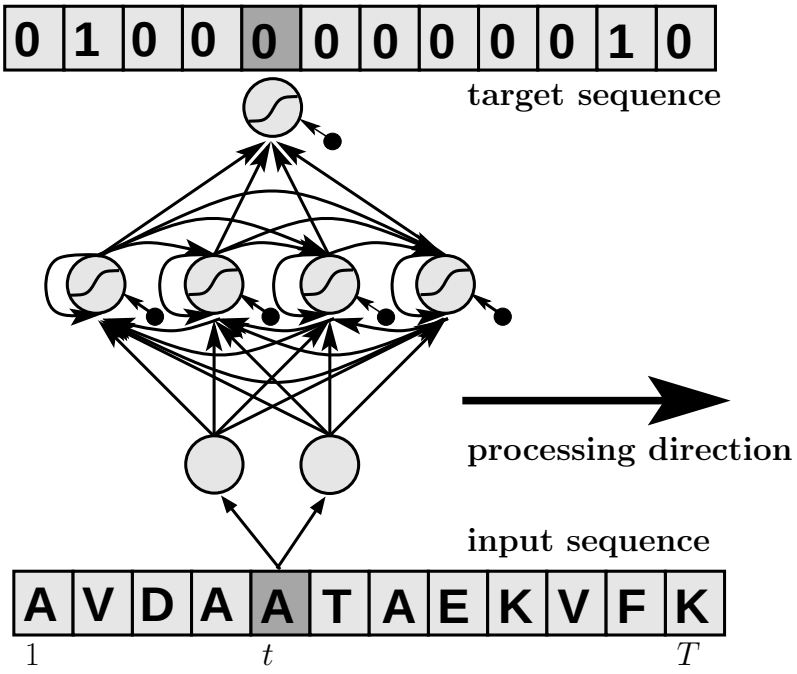


Figure 5.23: The processing of a sequence with a recurrent neural network. At each step the current input element is fed to the input units and the output unit should supply the target value.

5.6.1 Sequence Processing with RNNs

The activation functions and network inputs are now time dependent and are computed from the activations of previous time step:

$$\text{net}_i(t) \;=\; \sum_{j=0}^{N} w_{ij}\; a_j(t-1) \tag{5.115}$$

$$a_i(t) \;=\; f(\text{net}_i(t))\;. \tag{5.116}$$

Further we need initial activations $a_i(0)$ 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 sequence called 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

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \{\boldsymbol{X}(T)\}, \{\boldsymbol{Y}(T)\}) \;= \tag{5.117}$$
$$\frac{1}{l} \sum_{i=1}^{l} \sum_{t=1}^{T} \nabla_{\boldsymbol{w}} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right)\;.$$

The next subsections discuss how recurrent networks can be learned.

5.6.2 Real-Time Recurrent Learning

For performing gradient descent we have to compute

$$\frac{\partial}{\partial w_{uv}}\, L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \tag{5.118}$$

for all w_{uv}.

Using the chain rule this can be expanded to

$$\frac{\partial}{\partial w_{uv}}\, L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \;= \tag{5.119}$$
$$\sum_{k, k \text{ output unit}} \frac{\partial}{\partial a_k(t)} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right)\; \frac{\partial a_k(t)}{\partial w_{uv}}\;.$$

For all units k we can compute

$$\frac{\partial a_k(t+1)}{\partial w_{uv}} \;=\; f'(\text{net}_k(t+1))\; \left(\sum_l w_{kl} \frac{\partial a_l(t)}{\partial w_{uv}} \;+\; \delta_{ku} a_v(t)\right)\;, \tag{5.120}$$

where δ is the Kronecker delta with $\delta_{ku} = 1$ for $k = u$ and $\delta_{ku} = 0$ otherwise.

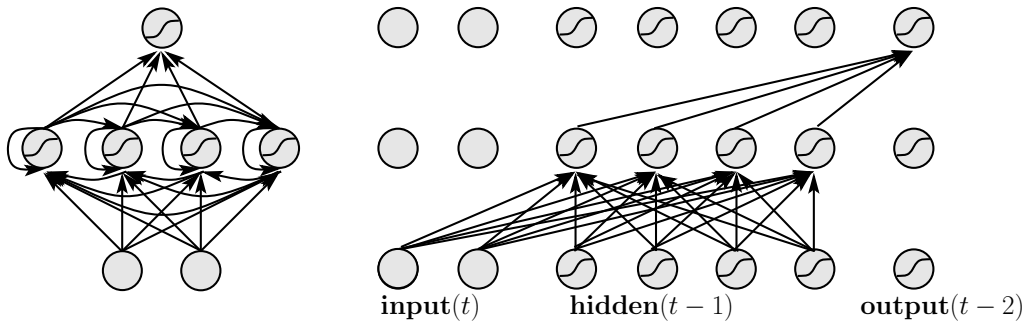


Figure 5.24: Left: A recurrent network. Right: the left network in feed-forward formalism, where all units have a copy (a clone) for each times step.

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*.

5.6.3 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. 5.24 for one step and the network unfolded over the whole time is depicted in Fig. 5.25.

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. 5.26.

The network of Fig. 5.26 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

$$L\left(\boldsymbol{y}^i, \boldsymbol{g}\left(\{\boldsymbol{x}^i\}; \boldsymbol{w}\right)\right) \;=\; \sum_{t=1}^{T} L\left(\boldsymbol{y}^i(t), \boldsymbol{g}\left(\{\boldsymbol{x}^i(t)\}; \boldsymbol{w}\right)\right) \tag{5.121}$$

for which we write for short

$$L \;=\; \sum_{t=1}^{T} L(t)\;. \tag{5.122}$$

The delta error is

$$\delta_j(t) \;=\; -\,\frac{\partial L}{\partial(net_j(t))} = -\,\frac{\partial\left(\sum_{\tau=t}^{T} L(\tau)\right)}{\partial(net_j(t))} \tag{5.123}$$

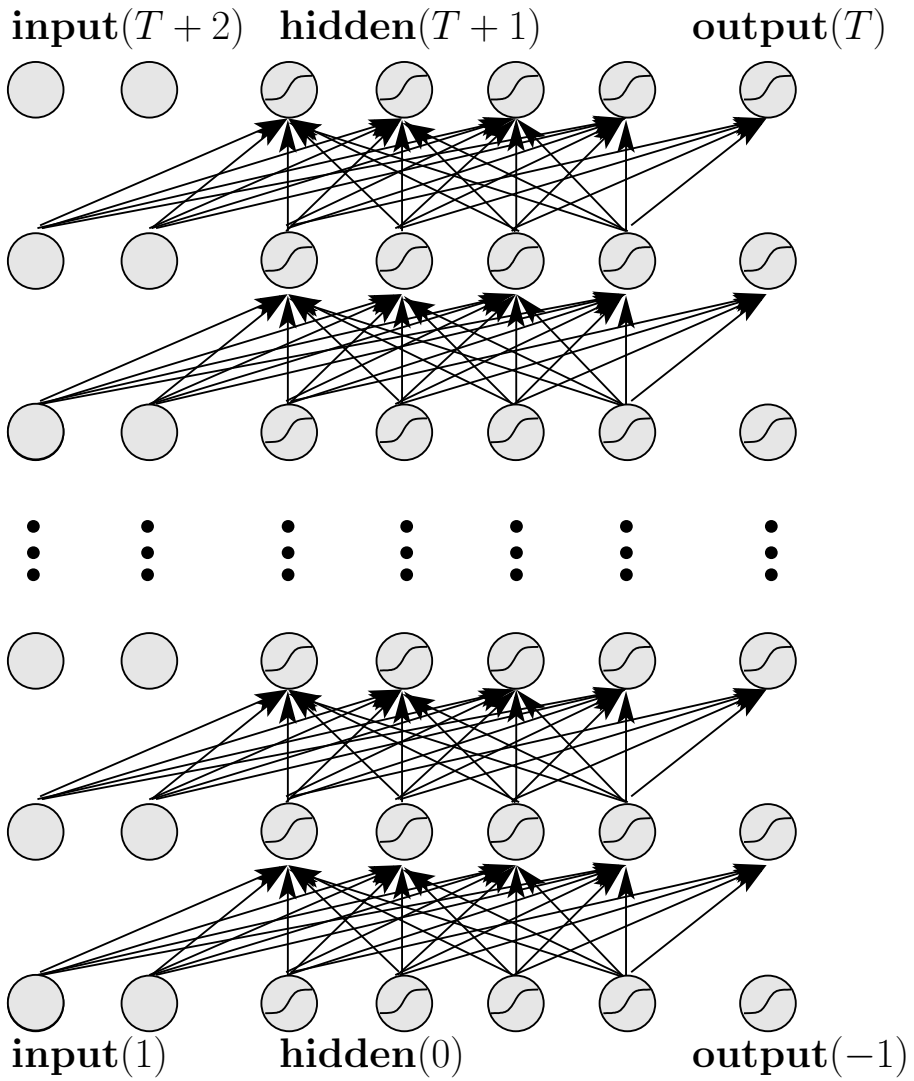


Figure 5.25: The recurrent network from Fig. 5.24 left unfolded in time. Dummy inputs at time $(t+1)$ and $(t+2)$ and dummy outputs at $(t-2)$ and $(t-1)$ are used. The input is shifted two time steps against the output to allow the input information be propagated through the network.

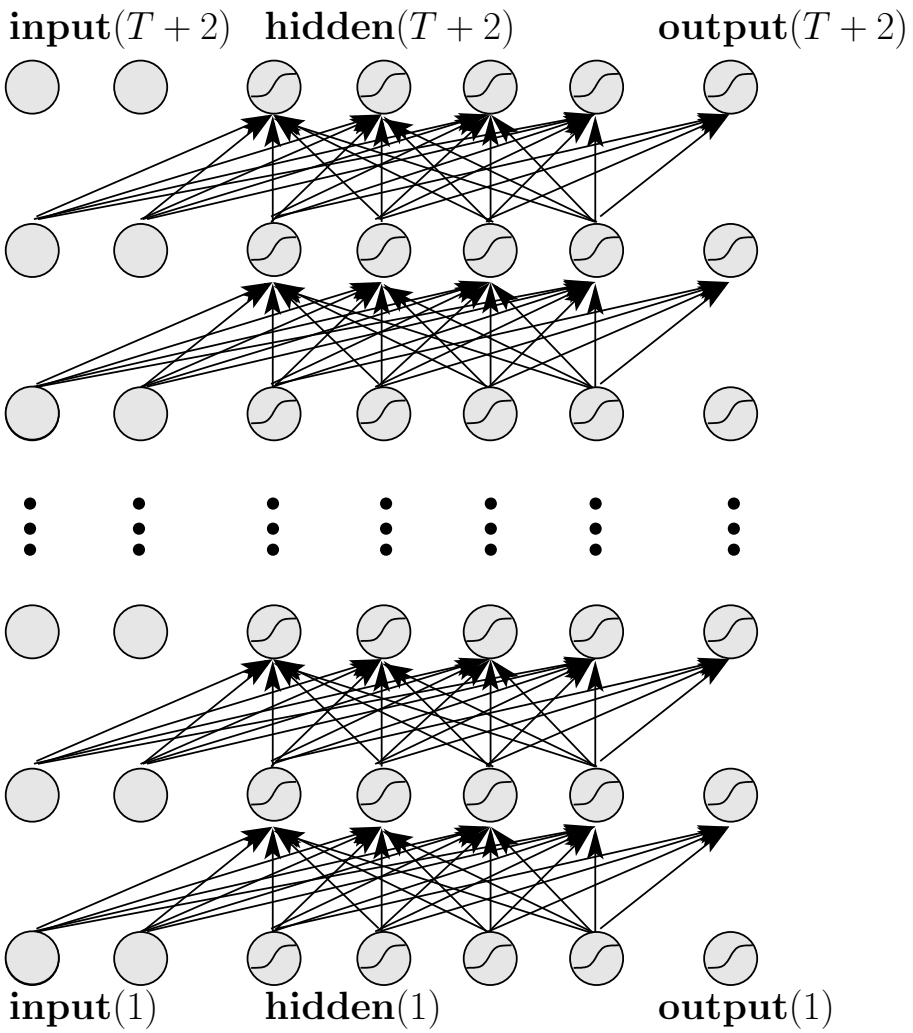


Figure 5.26: The recurrent network from Fig. 5.25 after re-indexing the hidden and output.

The back-propagation algorithm starts with

$$\delta_j(T) \;=\; f'(\mathrm{net}_j(T))\,\frac{\partial L(T)}{\partial a_j(T)}\;, \tag{5.124}$$

where in above reformulation T has to be replaced by $(T+2)$.

For $t = T-1$ to $t = 1$:

$$\delta_j(t) \;=\; f'_j(\mathrm{net}_j(t))\,\left(\frac{\partial L(t)}{\partial a_j(t)} \;+\; \sum_l w_{lj}\delta_l(t+1)\right)\;, \tag{5.125}$$

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 our architecture in Fig. 5.26 these errors are separated.

The derivative of the loss with respect to a weight in a layer t of the unfolded network in Fig. 5.26 is

$$-\,\frac{\partial L}{\partial w_{jl}(t)} \;=\; -\,\frac{\partial L}{\partial(\mathrm{net}_j(t))}\,\frac{\mathrm{net}_j(t)}{\partial w_{jl}(t)} \;=\; \delta_j(t)\;a_l(t-1)\;. \tag{5.126}$$

Because the corresponding weights in different layers are identical, the derivative of the loss with respect to a weight is

$$\frac{\partial L}{\partial w_{jl}} \;=\; \sum_{t=1}^{T}\frac{\partial L}{\partial w_{jl}(t)} \;=\; -\sum_{t=1}^{T}\left(\delta_j(t)\;a_l(t-1)\right)\;. \tag{5.127}$$

The on-line weight update is

$$w_{jl}^{\mathrm{new}} \;=\; w_{jl}^{\mathrm{old}} \;-\; \eta\,\frac{\partial L}{\partial w_{jl}}\;, \tag{5.128}$$

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 5.6.5).

Therefore BPTT is in most cases faster than RTRL without loss of performance.

5.6.4 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 e.g. 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, 1992] 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 $O(W^2/N) = O(W\ N)$.

5.6.5 Vanishing Gradient

The advantage of recurrent networks 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 a problem in storing the relevant information over time. If recurrent networks are not able to store information over time then their 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

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} = \begin{cases} f'\left(\mathrm{net}_v(t-1)\right)\ w_{uv} & q=1 \\ f'\left(\mathrm{net}_v(t-q)\right)\ \sum_{l=I}^{N} \frac{\partial \delta_l(t-q+1)}{\partial \delta_u(t)} w_{lv} & q>1 \end{cases} . \tag{5.129}$$

If we set $l_q = v$ and $l_0 = u$ then we obtain

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} = \sum_{l_1=I}^{N} \cdots \sum_{l_{q-1}=I}^{N} \prod_{r=1}^{q} f'\left(\mathrm{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}} . \tag{5.130}$$

Because of

$$\delta_v(t-q) = -\frac{\partial L}{\partial \mathrm{net}_v(t-q)} = -\sum_l \frac{\partial L}{\partial \mathrm{net}_l(t)} \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_v(t-q)} = \tag{5.131}$$

$$\sum_l \delta_l(t)\ \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_v(t-q)}$$

holds

$$\frac{\partial \delta_v(t-q)}{\partial \delta_u(t)} = \frac{\partial \mathrm{net}_u(t)}{\partial \mathrm{net}_v(t-q)} \tag{5.132}$$

true.

This leads to

$$\frac{\partial L(t)}{\partial w_{ij}(t-q)} = \tag{5.133}$$

$$\sum_{l:\text{output unit}} f'(\mathrm{net}_u(t))\ \frac{\partial L(t)}{\partial a_j(t)}\ \frac{\partial \mathrm{net}_l(t)}{\partial \mathrm{net}_i(t-q)} \frac{\partial \mathrm{net}_i(t-q)}{\partial w_{ij}(t-q)} =$$

$$\sum_{l:\text{output unit}} f'(\mathrm{net}_u(t))\ \frac{\partial L(t)}{\partial a_j(t)}\ \frac{\partial \vartheta_i(t-q)}{\partial \vartheta_l(t)} a_j(t-q)$$

which shows that $\frac{\partial L(t)}{\partial w_{ij}(t-q)}$ is governed by the factor in eq. (5.130).

The eq. (5.130) contains N^{q-1} terms of the form

$$\prod_{r=1}^{q} f'\left(\mathrm{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}} . \tag{5.134}$$

If the multipliers

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ >\ 1\ , \tag{5.135}$$

the learning is instable because derivatives grow over time and the weight updates are too large.

If the multipliers

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ <\ 1\ , \tag{5.136}$$

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 Hochreiter [1997b,a], Hochreiter and Schmidhuber [1997g,c], Hochreiter [1991].

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.

5.6.6 Long Short-Term Memory

From previous considerations we know that to avoid both instable learning and the vanishing gradient, we have to enforce

$$f'\left(\text{net}_{l_r}(t-r)\right)\ w_{l_r l_{r-1}}\ =\ 1\ . \tag{5.137}$$

For simplicity we consider only one unit j with a self-recurrent connection w_{jj} and obtain

$$f'\left(\text{net}_j(t-r)\right)\ w_{jj}\ =\ 1\ . \tag{5.138}$$

Solving this differential equation gives:

$$f(x)\ =\ \frac{1}{w_{jj}}\ x\ . \tag{5.139}$$

Because f depends on the self-recurrent connection we can set $w_{jj} = 1$ to fix f and obtain

$$f(x)\ =\ x\ . \tag{5.140}$$

Fig. 5.27 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.

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. 5.28 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.

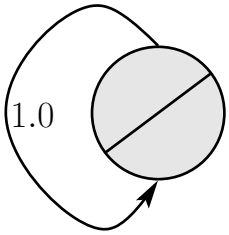


Figure 5.27: A single unit with self-recurrent connection which avoids the vanishing gradient.

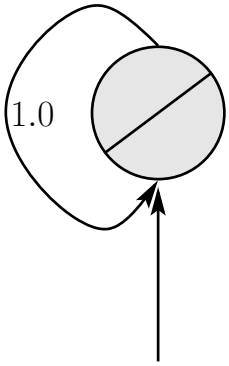


Figure 5.28: A single unit with self-recurrent connection which avoids the vanishing gradient and which has an input.

Only relevant information should be stored. In order to realize that an *input gate* a_{in} (basically a sigma-pi unit) is used. If we further assume that the network input $\mathrm{net_c}$ to the single unit is squashed then we obtain following dynamics:

$$a(t+1) \;=\; a(t) \;+\; a_{\mathrm{in}}(t)\; g\left(\mathrm{net_c}(t)\right) \;, \tag{5.141}$$

If a_{in} is a sigmoid unit active in $[0, b]$ then it 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

$$a_{\mathrm{c}}(t+1) \;=\; a_{\mathrm{out}}(t)\; h\left(a(t+1)\right) \;. \tag{5.142}$$

The whole dynamics is

$$a_{\mathrm{c}}(t+1) \;=\; a_{\mathrm{out}}(t)\; h\left(a(t) \;+\; a_{\mathrm{in}}(t)\; g\left(\mathrm{net_c}(t)\right)\right) \;. \tag{5.143}$$

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. 5.29.

The input to the storing unit is controlled by an "input gating" or attention unit (Fig. 5.29, unit marked "a_{in}") which blocks class-irrelevant information, so that only class-relevant information

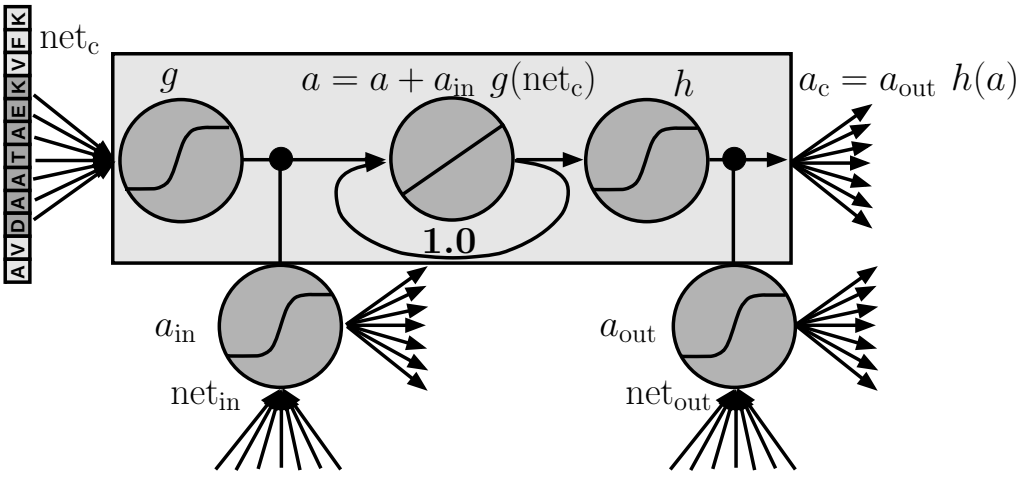


Figure 5.29: The LSTM memory cell. Arrows represent weighted connections of the neural network. Circles represent units (neurons), where the activation function (linear or sigmoid) is indicated in the circle.

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. 5.29, center) is bounded by the sigmoid function h (Fig. 5.29, unit labeled as "h"). Memory readout is controlled by an "output gate" (Fig. 5.29, unit labeled "a_{out}"). The cell's output a_{c} is then computed according to eq. (5.142) and eq. (5.143).

The recurrent network built from memory cells is called "Long Short-Term Memory" (LSTM, [Hochreiter and Schmidhuber, 1994, 1996, 1997c,g]). It 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. 5.30.

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. 5.31.

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.

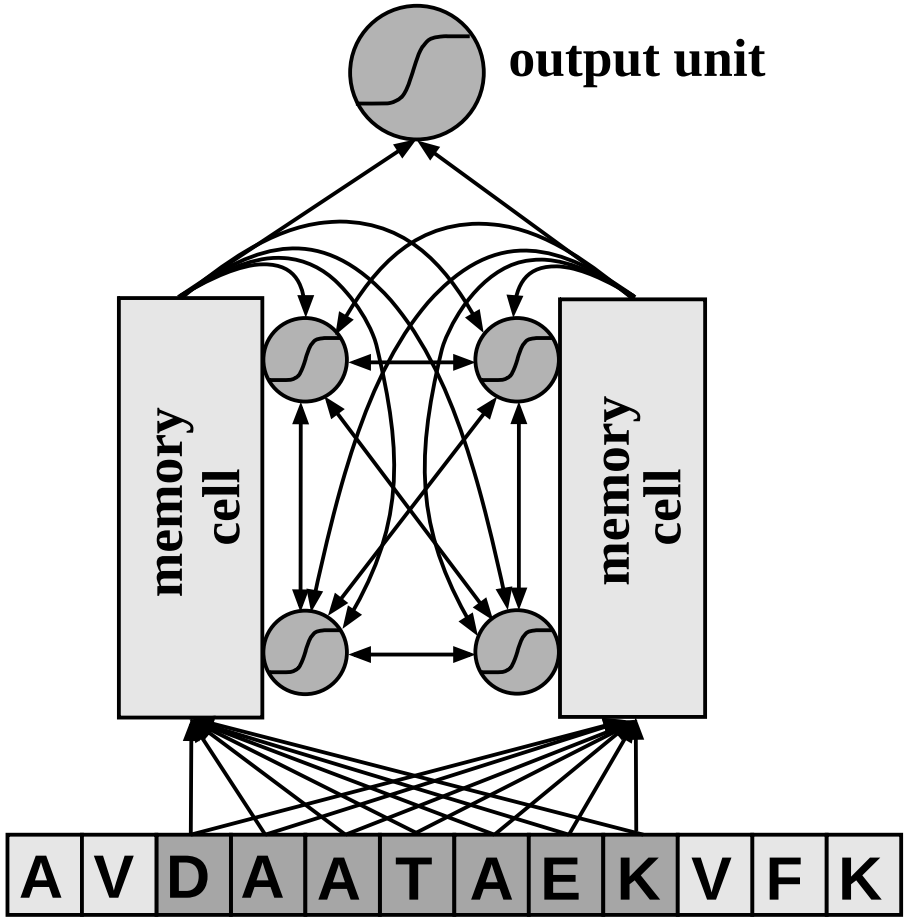


Figure 5.30: LSTM network with three layers: input layer (window of the amino acid sequence – shaded elements), hidden layer (with memory cells – see Fig. 5.29), and output layer. Arrows represent weighted connections; the hidden layer is fully connected. The subnetworks denoted by "memory cell" can store information and receive three kinds of inputs, which act as pattern detectors (input $\rightarrow$ 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.

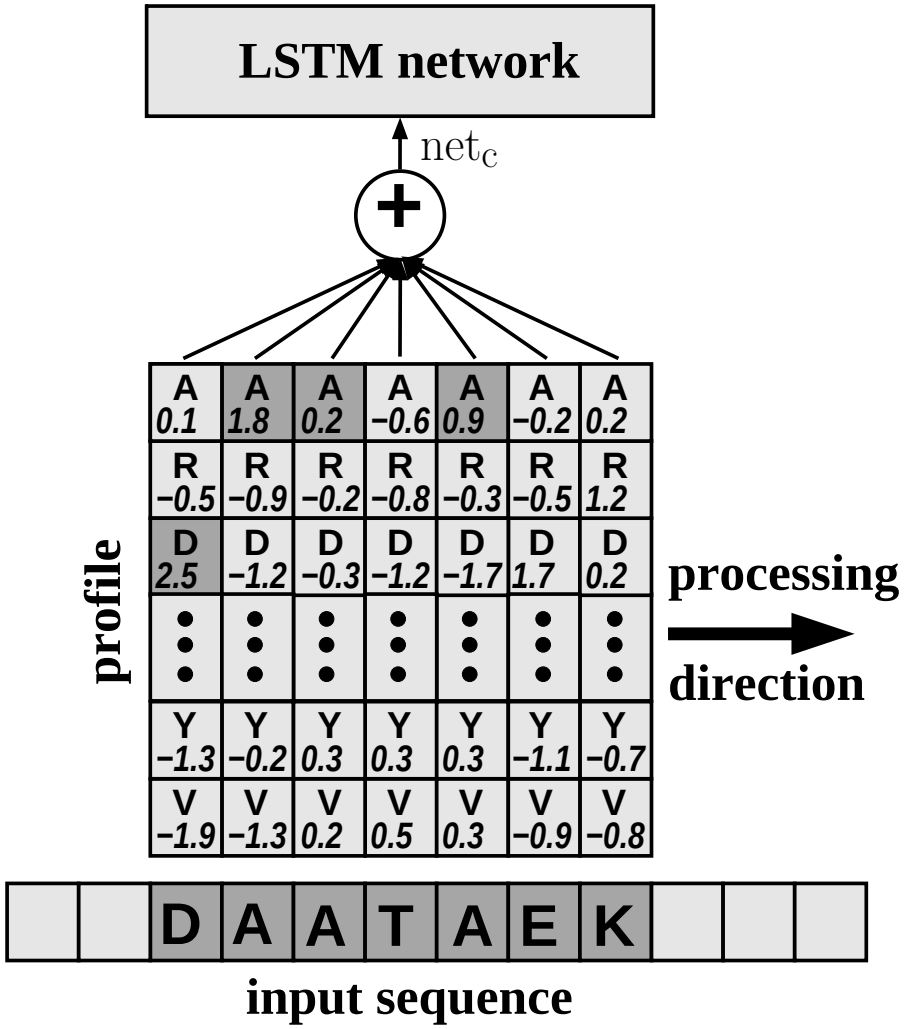


Figure 5.31: A profile as input to the LSTM network which scans the input from left to right. Amino acids are represented by the one letter code. Shaded squares in the matrix match the input and contribute to the sum $\mathrm{net_c}$.

Chapter 6

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.

6.1 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.

6.1.1 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 presence 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 were achieved. 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) 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 from PCA components. [Tibshirani et al., 2003] achieved 72.5% balanced accuracy with only seven features.

Prostate Cancer

Classification of healthy and cancerous persons based on SELDI TOF MS data using decision trees achieved 81% sensitivity and 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 97% specificity and 83% sensitivity (balanced accuracy of 90%).

These results show varying results if using feature selection which was discussed in [Diamandis, 2003].

6.1.2 Protein Sequences

The most classical task of bioinformatics is sequence processing and comparing new sequences with known sequences in order to detect evolutionary relationships. The later 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., 1994, 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 be 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.

6.1.3 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 different 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. 6.1 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.

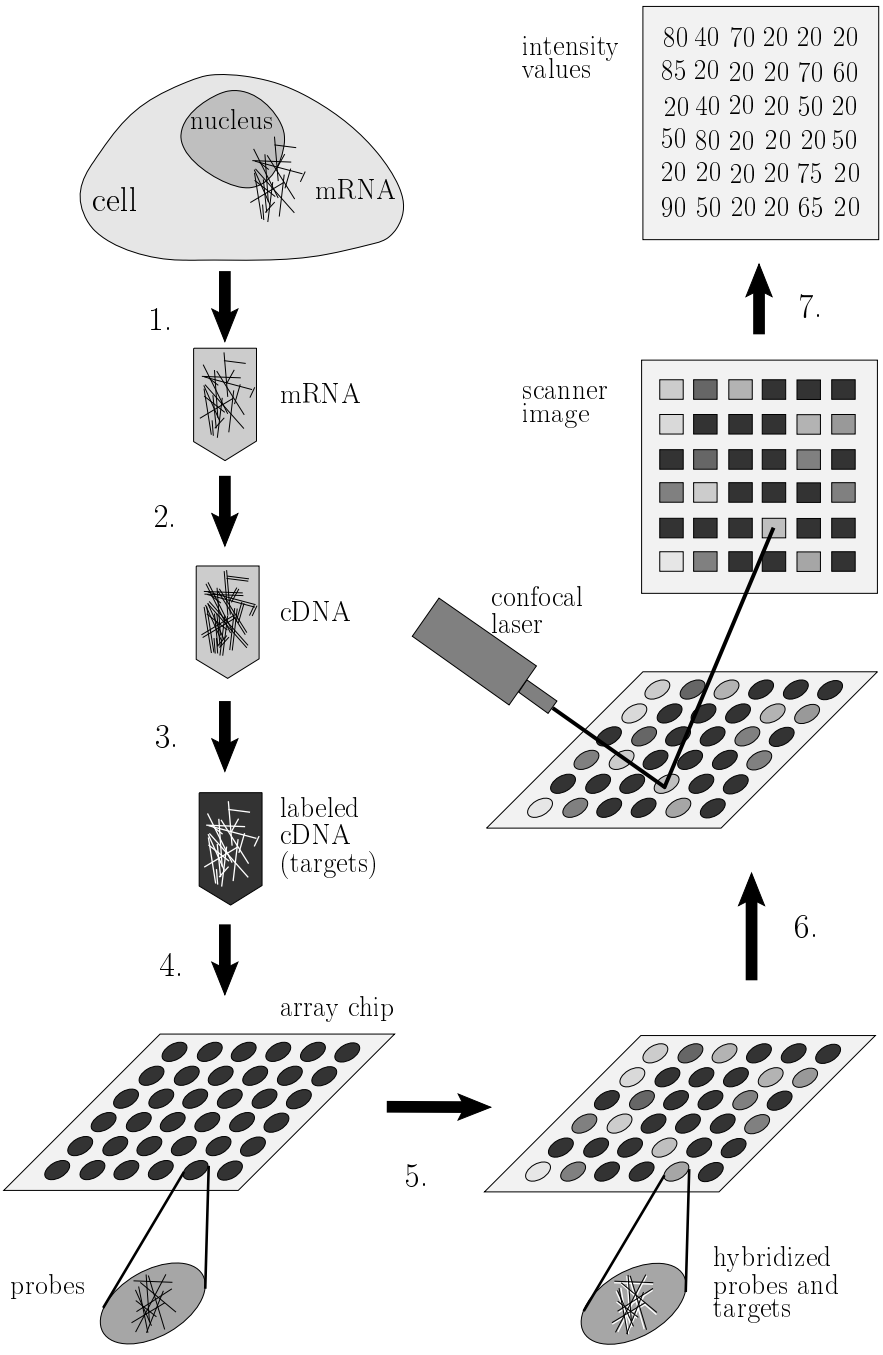


Figure 6.1: The microarray technique (see text for explanation).

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].

6.2 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.

6.2.1 Filter Methods

Filter methods extract features whose values show dependencies with class labels without explicitly relying on a regression or classification method.

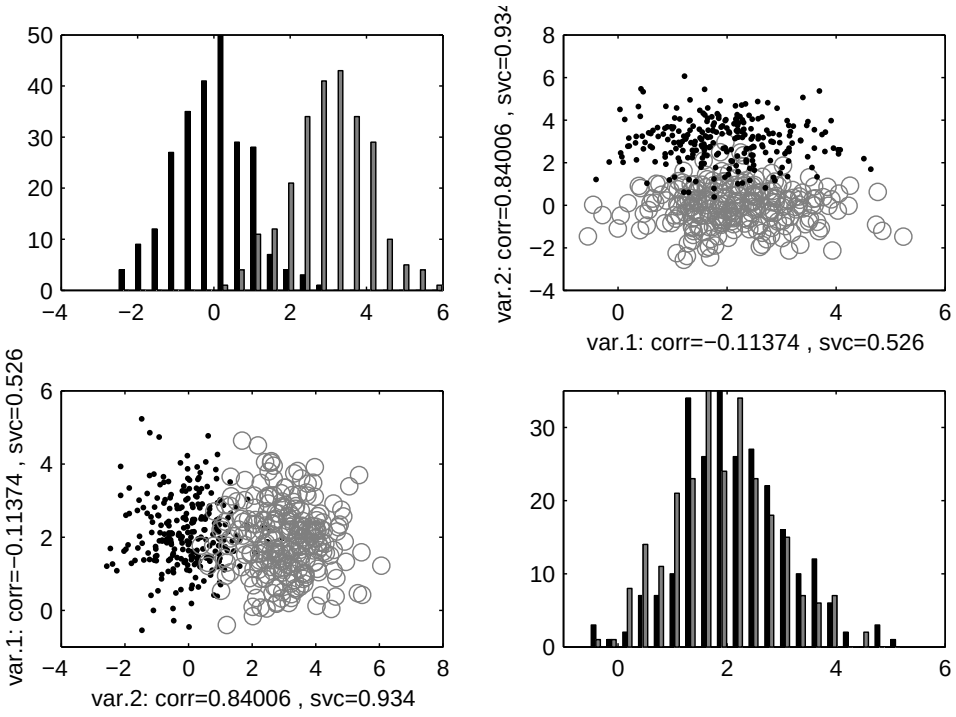


Figure 6.2: Simple two feature classification problem, where feature 1 (var. 1) is noise and feature 2 (var. 2) is correlated to the classes. In the upper right figure and lower left figure only the axis are exchanged. The upper left figure gives the class histogram along feature 2 whereas the lower 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. 2.9 (below Fig. 6.2). 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

$$\bar{y} \;=\; \frac{1}{l}\sum_{i=1}^{l} y^i \tag{6.1}$$

$$\bar{x}_j \;=\; \frac{1}{l}\sum_{i=1}^{l} x_j^i \tag{6.2}$$

$$\sigma_y^2 \;=\; \frac{1}{l}\sum_{i=1}^{l} \left(y^i \;-\; \bar{y}\right)^2 \tag{6.3}$$

$$\sigma_j^2 \;=\; \frac{1}{l}\sum_{i=1}^{l} \left(x_j^i \;-\; \bar{x}_j\right)^2 \;. \tag{6.4}$$

For classification we have $y \in \{-1, 1\}$ and assume we have l^+ class $+1$ and l^- class -1 examples then we define

$$\bar{x}_j^+ \;=\; \frac{1}{l^+}\sum_{i=1;y^i=1}^{l} x_j^i \tag{6.5}$$

$$\bar{x}_j^- \;=\; \frac{1}{l^-}\sum_{i=1;y^i=-1}^{l} x_j^i \tag{6.6}$$

$$\left(\sigma_j^+\right)^2 \;=\; \frac{1}{l^+}\sum_{i=1;y^i=1}^{l} \left(x_j^i \;-\; \bar{x}_j^+\right)^2 \tag{6.7}$$

$$\left(\sigma_j^-\right)^2 \;=\; \frac{1}{l^-}\sum_{i=1;y^i=1}^{l} \left(x_j^i \;-\; \bar{x}_j^-\right)^2 \;. \tag{6.8}$$

Pearson's correlation coefficient is

$$\text{corr} \;=\; \frac{\sum_{i=1}^{l}\left(x_j^i \;-\; \bar{x}_j\right)\;\left(y^i \;-\; \bar{y}\right)}{\sigma_y\;\sigma_j} \tag{6.9}$$

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

$$\frac{\bar{x}_j^+ \;-\; \bar{x}_j^-}{\sigma_j^+ \;+\; \sigma_j^-} \tag{6.10}$$

which is also known as Golub's criterion.

The Fisher criterion is

$$\frac{\left(\bar{x}_j^+ \;-\; \bar{x}_j^-\right)^2}{\left(\sigma_j^+\right)^2 \;+\; \left(\sigma_j^-\right)^2} \;. \tag{6.11}$$

The two-sample t-test uses

$$\frac{\bar{x}_j^+ - \bar{x}_j^-}{\sqrt{\left(\sigma_j^+\right)^2 / l^+ + \left(\sigma_j^-\right)^2 / l^+}} . \tag{6.12}$$

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 and often can not be achieved by statistical methods. Remember Fig. 2.11 which is below given as Fig. 6.3 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. 2.12 which is below shown again as Fig. 6.4. 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. 6.1) where features which have no correlation with the target should be selected. The table shows also an example where the feature with the largest correlation with the target should not be selected. Reason is that correlation between features can 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. 6.1, 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. 6.1 depicts another situation, where $t = f_2 + f_3$. f_1, the

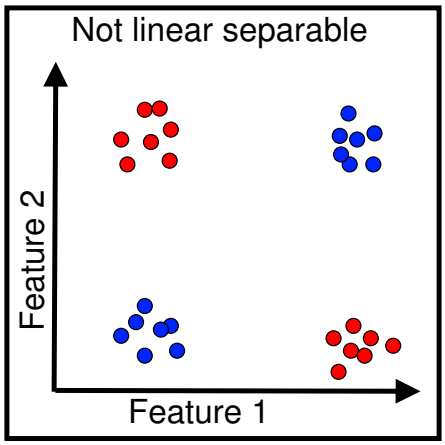


Figure 6.3: An XOR problem of two features, where each single feature is neither correlated to the problem nor helpful for classification. Only both features together help.

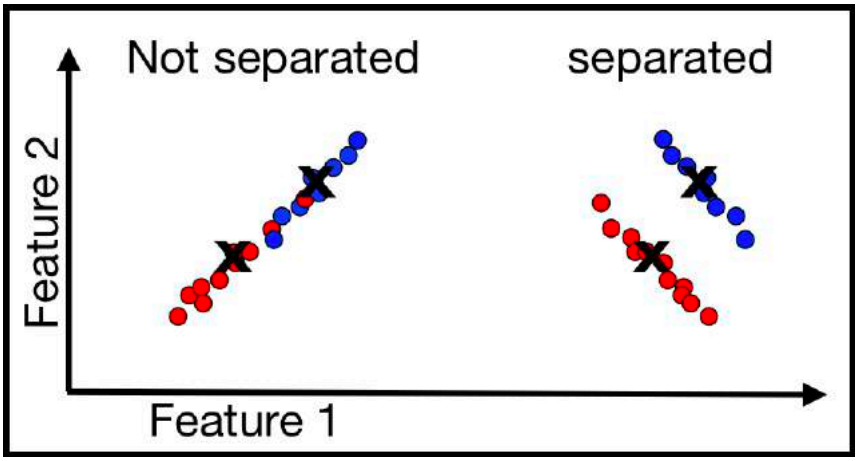


Figure 6.4: The left and right subfigure each show two classes where the features mean value and variance for each class is equal. However, the direction of the variance differs in the subfigures leading to different performance in classification.

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 6.1: 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.

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.

6.2.2 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].

6.2.3 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 techniques, 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.

6.2.3.1 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 5.4.5.3 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.

6.2.3.2 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].

6.2.3.3 P-SVM Feature Selection

The Potential Support Vector Machine (P-SVM) from Section 3.12 is suitable for feature selection.

The P-SVM selects the optimal features in Tab. 6.1 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 α 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

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \; & \frac{1}{2} \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^\top \boldsymbol{F}^\top \boldsymbol{F} \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right) \\ & - \boldsymbol{y}^\top \boldsymbol{F} \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right) + \epsilon \, \mathbf{1}^\top \left(\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-\right) \\ \text{subject to } & \mathbf{0} \leq \boldsymbol{\alpha}^+ , \; \mathbf{0} \leq \boldsymbol{\alpha}^- \; . \end{aligned} \tag{6.13}$$

- 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

6.2.4 Automatic Relevance Determination

Automatic Relevance Determination (ARD) is a special case of weight decay from Section 5.4.5.4, where weight decay is applied to the input weights of hidden units and the maximum posterior is determined [Neal, 1996, MacKay, 1993].

Here a Gaussian prior is used. However the Gaussian 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

$$\beta_j = \frac{1 - \beta_j \, H_{jj}^{-1}}{(w_i)_{\mathrm{MAP}}^2} \; . \tag{6.14}$$

Using this update of β, the evidence framework 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.

6.3 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.

6.3.1 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.

6.3.2 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.

6.3.3 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 7

Time Series Prediction

In this chapter we consider time series. A time series is a sequence of observations, measured typically at successive time points often in equidistant intervals. Examples are stock markets, daily temperature, hourly traffic volume in an area, etc. Data analysis for time series can focus on finding structures in the time series or on forecasting, that is, predicting the future.

7.1 Discrete Fast Fourier Transformation

7.1.1 The Method

First we want to investigate whether a time series has a regularity in terms of periodically occurring events. The *discrete Fourier transform* (DFT) maps a time series (equally spaced) to coefficients of complex sinusoids ordered by their frequencies. The frequency is one over the period. Thus, DFT maps a time series into an equivalent representation in the frequency domain. DFT is typically used to find structures in time series, e.g. some periodically occurring events. A typical application of DFT is speech recognition where the frequencies characterize the spoken word. Therefore the power of each frequency is computed and used as a feature to characterize the sound.

For a time series of complex numbers $x_0, \ldots, x_{n-1}$, the DFT is

$$X_k = \sum_{l=0}^{n-1} x_l e^{-i2\pi k \frac{l}{n}}, \qquad k = 0, \ldots, n-1\,. \tag{7.1}$$

To compute these values requires $O(n^2)$ operations: for each of the n values of k, the X_k is computed as a sum over n values. This kind of computing the DFT is inefficient. A *fast Fourier transform* (FFT) computes all the values X_k in $O(n \log n)$ operations.

The Fourier transform is an invertible, linear transformation of a time series because the weights $e^{-i2\pi k \frac{l}{n}}$ do not depend on x_l. Therefore each time series can completely be represented in the frequency domain by the X_k. The frequency components X_k give the strength of the according frequency in the time series, the strength is the amplitude of the according sine or cosine function. The frequency components X_k also give the shift of the according sine or cosine functions, that is, where they start in the time series. The DFT allows to describe the time series by sine and cosine functions where for each possible frequency the amplitude and the shift is given. The possible frequencies are determined by the length of the time series and the distance between the observations. The sine and cosine function form an orthonormal system in the space of functions,

therefore all functions can be represented by sine and cosine functions. Thus, DFT detects all periodic structures in the time series. How prominent these structures are, is given by the amplitude of the according sine and cosine functions.

7.1.2 Examples

7.1.2.1 Toy Example: Sine

For $\sin(a * x)$, the frequency is $f = a/(2 * pi)$ and the period $T = 2 * pi/a = 1/f$. Amplitude $\mathrm{fft}(0)$ corresponds to the constant, i.e. is the constant offset. $\mathrm{fft}(n)$ corresponds to frequency $f = (n-1)/N$, where N is the length of the time series. Fig. 7.1 shows the sine function which is now analyzed by FFT:

```
  [1] -0.1271710+ 0.0000000i -0.1285587+ 0.0267120i -0.1327889+ 0.0540163i
  [4] -0.1400710+ 0.0825534i -0.1507837+ 0.1130696i -0.1655263+ 0.1464929i
  [7] -0.1852095+ 0.1840486i -0.2112135+ 0.2274437i -0.2456697+ 0.2791878i
 [10] -0.2919921+ 0.3431868i -0.3559498+ 0.4259353i -0.4480520+ 0.5391614i
 [13] -0.5895920+ 0.7065214i -0.8311289+ 0.9840678i -1.3288687+ 1.5448285i
 [16] -2.9216331+ 3.3176941i 33.2577080-36.7354101i  2.7220874- 2.9144441i
 [19]  1.4868101- 1.5384705i  1.0541469- 1.0515208i  0.8345114- 0.8007089i
```

The FFT component $n = 17$ has the largest contribution. This component corresponds to frequency $(n-1)/N = 16/100 = 0.16$ and a period of 6.25. The sine function has frequency $1/(2 * \pi) = 0.1591549$ and a period of $2 * \pi = 6.28$. For demonstrating the principle, we were able to detect the frequency of the sine by FFT.

Fig. 7.2 shows a sine function $\sin(ax)$ with $a = 0.3$. We want to see if we also detect this frequency by FFT:

```
  [1]   2.3039705+ 0.0000000i   2.3828756+ 0.8779728i   2.6699646+ 2.0341885i
  [4]   3.4258895+ 4.1502170i   6.3525463+11.2064203i -17.2169565-43.0979803i
  [7]  -2.3859796- 8.5970499i  -0.9125592- 5.0318487i  -0.3678168- 3.6387286i
 [10]  -0.0899008- 2.8812670i   0.0758478- 2.3979055i   0.1844013- 2.0586730i
 [13]   0.2601095- 1.8051119i   0.3153613- 1.6069145i   0.3570957- 1.4467319i
 [16]   0.3894845- 1.3138789i   0.4151763- 1.2013920i   0.4359275- 1.1045255i
 [19]   0.4529444- 1.0199251i   0.4670804- 0.9451468i   0.4789545- 0.8783642i
```

The FFT component $n = 6$ has the largest contribution. This component corresponds to frequency $(n-1)/N = 5/100 = 0.05$ and a period of 20. The sine function with $a = 0.3$ has frequency $0.3/(2 * \pi) = 0.04774648$ and period of $2 * \pi/0.3 = 20.94$. We also detected this frequency.

7.1.2.2 Lung Related Deaths

Three time series giving the monthly deaths from bronchitis, emphysema, and asthma in the UK from 1974 to 1979. The time series is a count of the death of both sexes and shown in Fig. 7.3.

The frequency is 0.0833 and the period is 12 months. This means that we observe an annual period of increased deaths. The peaks are in winter time at the months December, January, and February.

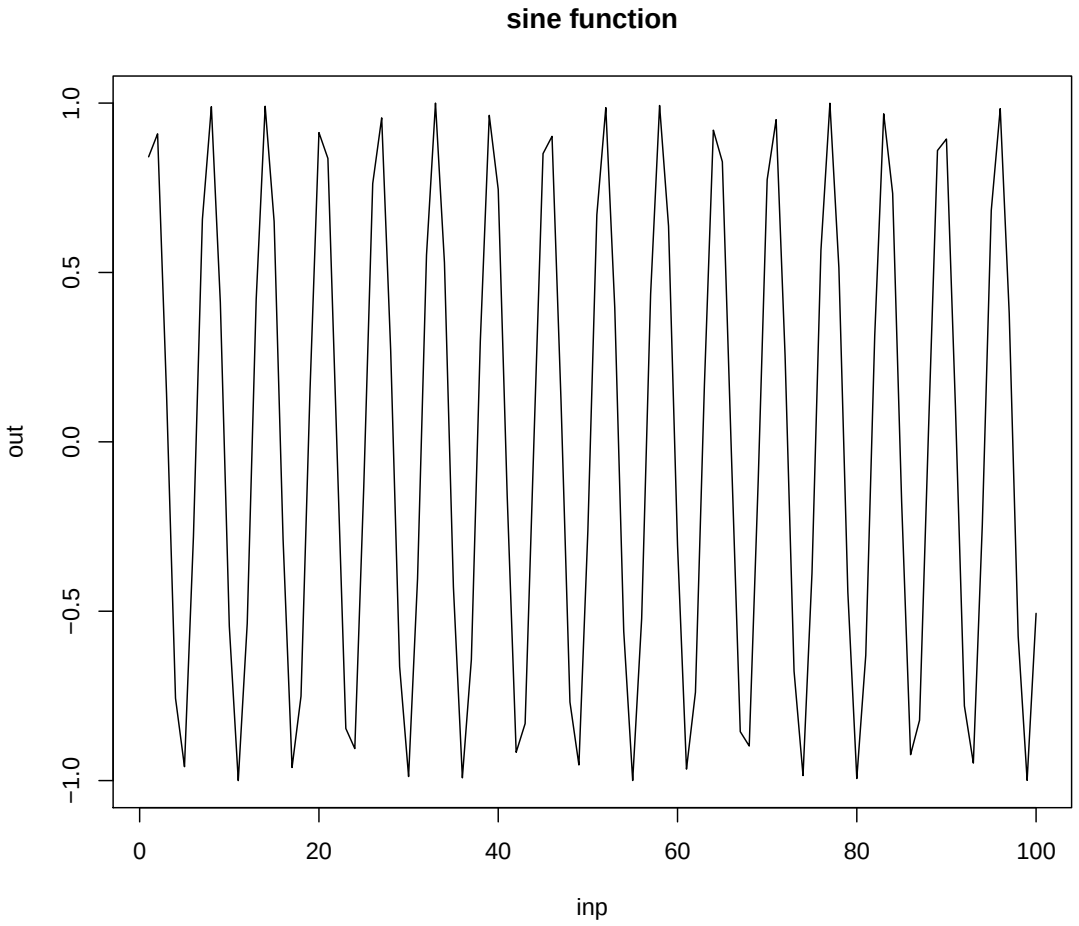


Figure 7.1: The sine function.

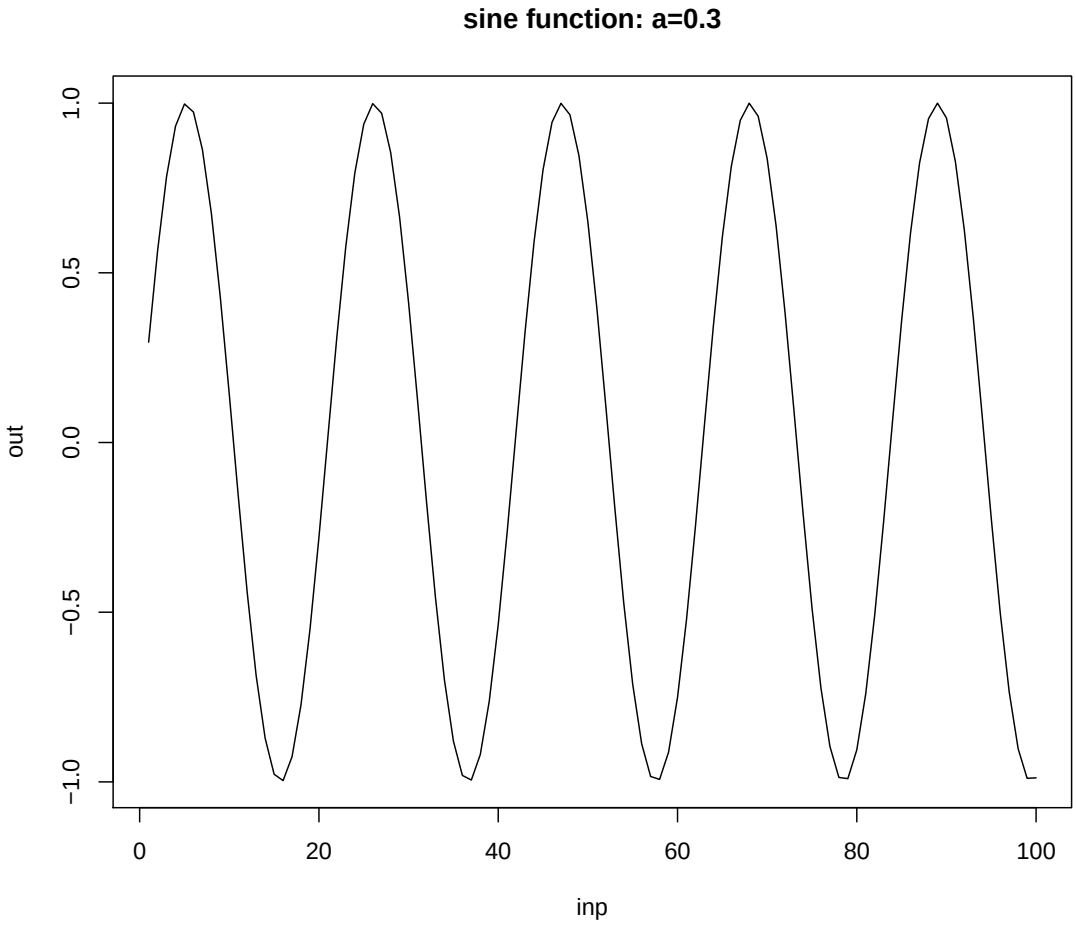


Figure 7.2: The sine function with $a = 0.3$.

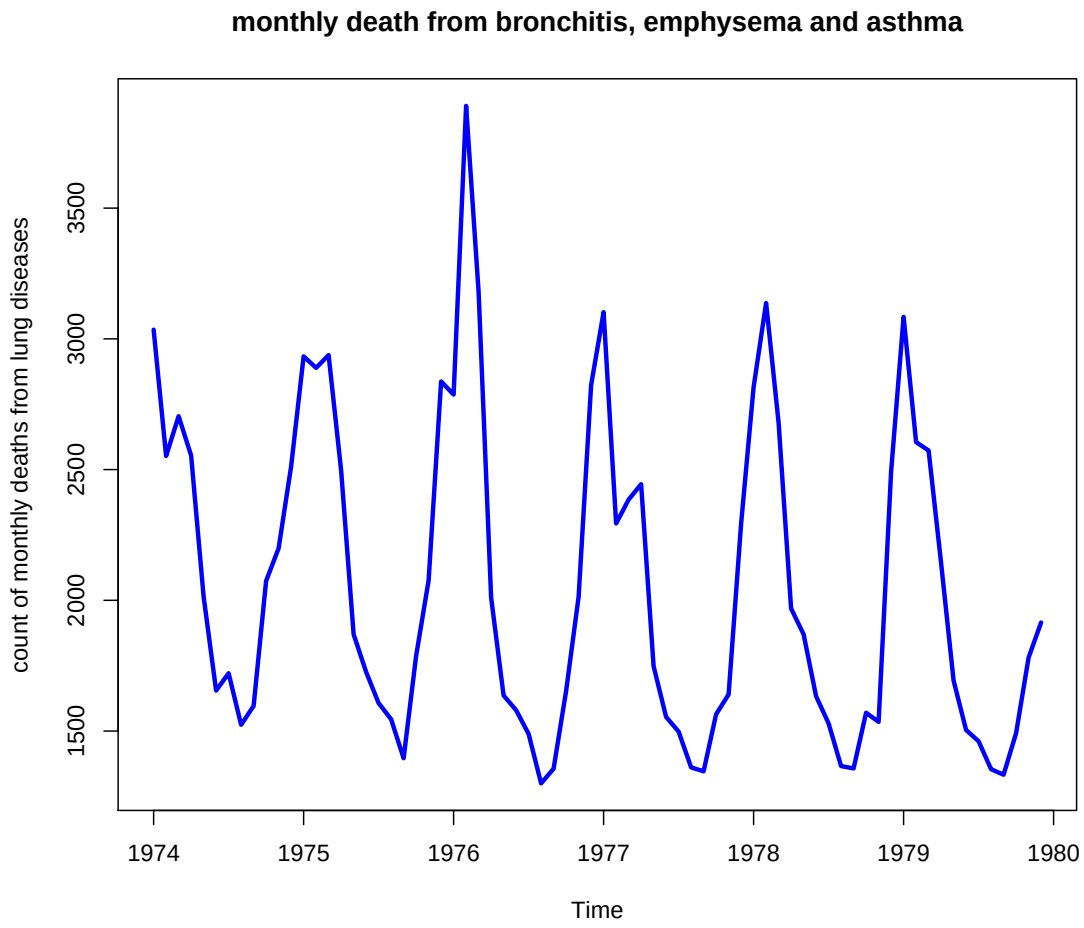


Figure 7.3: Time series giving the monthly deaths from bronchitis, emphysema, and asthma in the UK from 1974 to 1979.

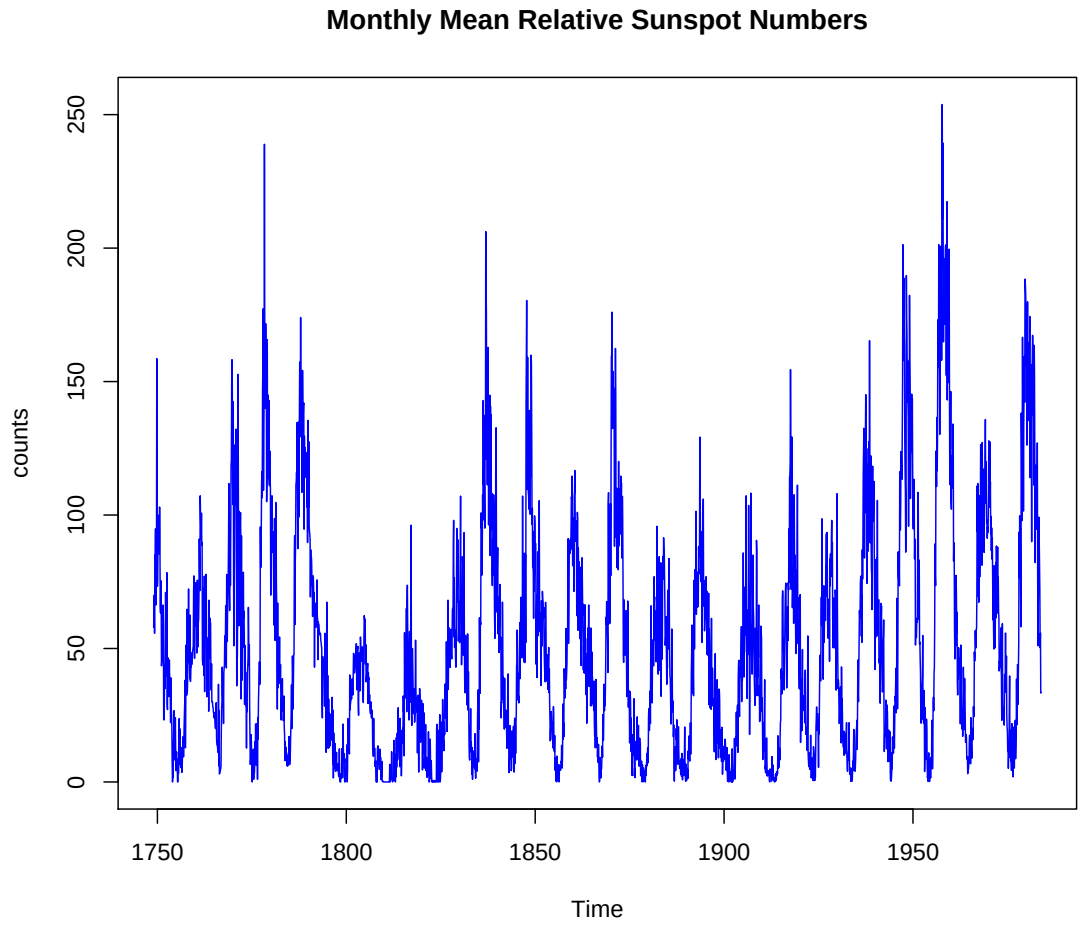


Figure 7.4: Monthly mean relative sunspot numbers from 1749 to 1983.

7.1.2.3 Sunspots

Monthly mean relative sunspot numbers were recorded from 1749 to 1983. Collected at Swiss Federal Observatory, Zürich until 1960, then at Tokyo Astronomical Observatory. Fig. 7.4 shows the time series.

We compute the power spectrum of the sunspot data. Fig. 7.5 shows the power spectrum of the sunspot time series.

Next we analyze the frequency and the periods that are prominent. The most prominent frequency is 0.00744 and the according period is 134.28 months, which is 11.19 years. As a result we found that the number of sunspots on average increases all 11.19 years.

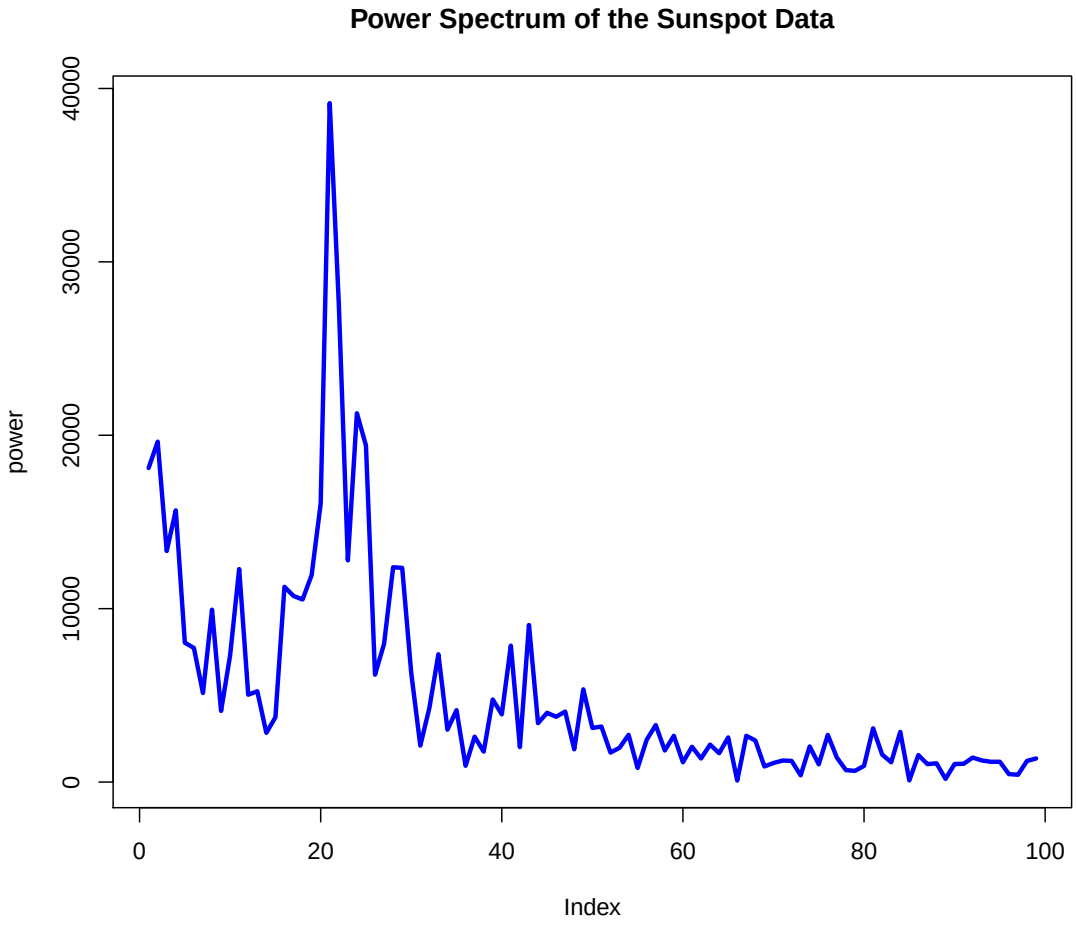


Figure 7.5: The power spectrum of the sunspot data.

7.2 ARMA and ARIMA Models

7.2.1 The Method

The second approach is based on the theory of linear models and can be used for forecasting. We can re-use our knowledge from linear models. *Autoregressive-moving-average* (ARMA) models of a time series $x_0, \ldots, x_n$ are linear models of the form:

$$x_t \;=\; c \;+\; \sum_{i=1}^{p} \beta_i \, x_{t-i} \;+\; \sum_{i=1}^{q} \alpha_i \, \epsilon_{t-i} \;+\; \epsilon_t \,, \tag{7.2}$$

where ϵ_t are iid from a zero-mean normal distribution. The part

$$\sum_{i=1}^{p} \beta_i \, x_{t-i} \tag{7.3}$$

is called the *autoregressive* part and the part

$$\sum_{i=1}^{q} \alpha_i \, \epsilon_{t-i} \tag{7.4}$$

is called the *moving-average* part. The autoregressive part is a linear model which predicts the next data point of the time series by means of previous data points of the time series. The moving average part is a linear model based on the past errors to predict the next data point. Therefore, local tendencies to underestimate or to overestimate the next data point are captured. Thus, local trends may be found and included into the model.

The parameters p and q are hyperparameters and determine the model complexity. The coefficients $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ can be found by least squares regression.

Autoregressive-integrated-moving-average (ARIMA) models are ARMA models based on the differences of time series. Instead of the time series x_t, the time series of differences $x_t - x_{t-1}$ is used as data, where for equidistant time points these differences can be considered as derivatives. Consequently, the result must be "integrated" in order to recover the original time series.

The assumptions to justify ARMA models are

- the stochastic process x_t is stationary (probability distribution is the same when shifted in time) and ergodic (time average is the population average);
- the regressors $\boldsymbol{x}_t$ (moving average and autoregressive combined) are predetermined: $\mathrm{E}(\boldsymbol{x}_t \epsilon_t) = 0$ for all $t = 0, \ldots, n$;
- the covariance matrix of the regressors is of full rank;
- the sequence $\{(x_t, \epsilon_t)\}$ is a martingale difference sequence (zero mean given the past) with existing second moments $\mathrm{E}(\epsilon_t^2 \boldsymbol{x}_t \boldsymbol{x}_t^T)$.

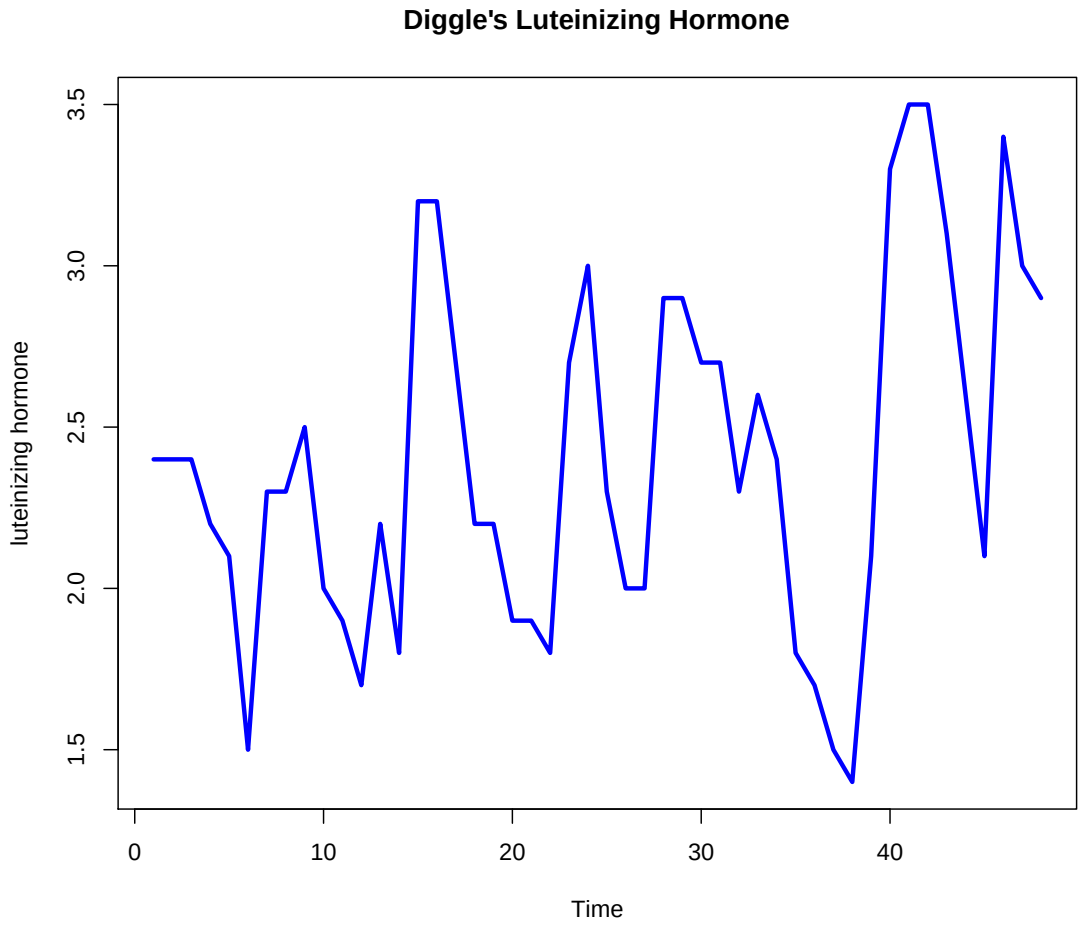


Figure 7.6: Diggle's luteinizing hormone: time series at 10 min intervals from a human female.

7.2.2 Examples

7.2.2.1 Luteinizing Hormone

The first example is a time series giving the luteinizing hormone in blood samples at 10 mins intervals from a human female. 48 time points are available. The data is from P. J. Diggle (1990), "Time Series: A Biostatistical Introduction", Table A.1, series 3. The data are:

```
2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8 3.2 3.2 2.7 2.2 2.2
1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4
2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9
```

The time series is shown in Fig. 7.6.

First we fit an ARMA model with one autoregressive component:

```
Coefficients:
```

```
         ar1  intercept
      0.5739     2.4133
s.e.  0.1161     0.1466

sigma^2 estimated as 0.1975:  log likelihood = -29.38,  aic = 64.76
```

Then we fit an ARMA model with three autoregressive components:

```
Coefficients:
         ar1      ar2      ar3  intercept
      0.6448  -0.0634  -0.2198     2.3931
s.e.  0.1394   0.1668   0.1421     0.0963

sigma^2 estimated as 0.1787:  log likelihood = -27.09,  aic = 64.18
```

Then we fit an ARMA model with one autoregressive and moving average component:

```
Coefficients:
         ar1     ma1  intercept
      0.4522  0.1982     2.4101
s.e.  0.1769  0.1705     0.1358

sigma^2 estimated as 0.1923:  log likelihood = -28.76,  aic = 65.52
```

Akaike's information criterion prefers the last model. Fig. 7.8 shows the different ARMA models for the luteinizing hormone time series. The models follow the original data with one time step delay.

Next we do 12 steps prediction into the future. Fig. 7.8 shows the prediction into the future for these models.

7.2.2.2 US Accidental Deaths

This is a time series of monthly total accidental deaths in the USA in the period from 1973 to 1978. The data stems from P. J. Brockwell and R. A. Davis (1991) "Time Series: Theory and Methods".

The data including the first six month of 1979 is

```
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1973  9007  8106  8928  9137 10017 10826 11317 10744  9713  9938  9161  8927
1974  7750  6981  8038  8422  8714  9512 10120  9823  8743  9129  8710  8680
1975  8162  7306  8124  7870  9387  9556 10093  9620  8285  8466  8160  8034
1976  7717  7461  7767  7925  8623  8945 10078  9179  8037  8488  7874  8647
1977  7792  6957  7726  8106  8890  9299 10625  9302  8314  8850  8265  8796
1978  7836  6892  7791  8192  9115  9434 10484  9827  9110  9070  8633  9240
1979  7798  7406  8363  8460  9217  9316
```

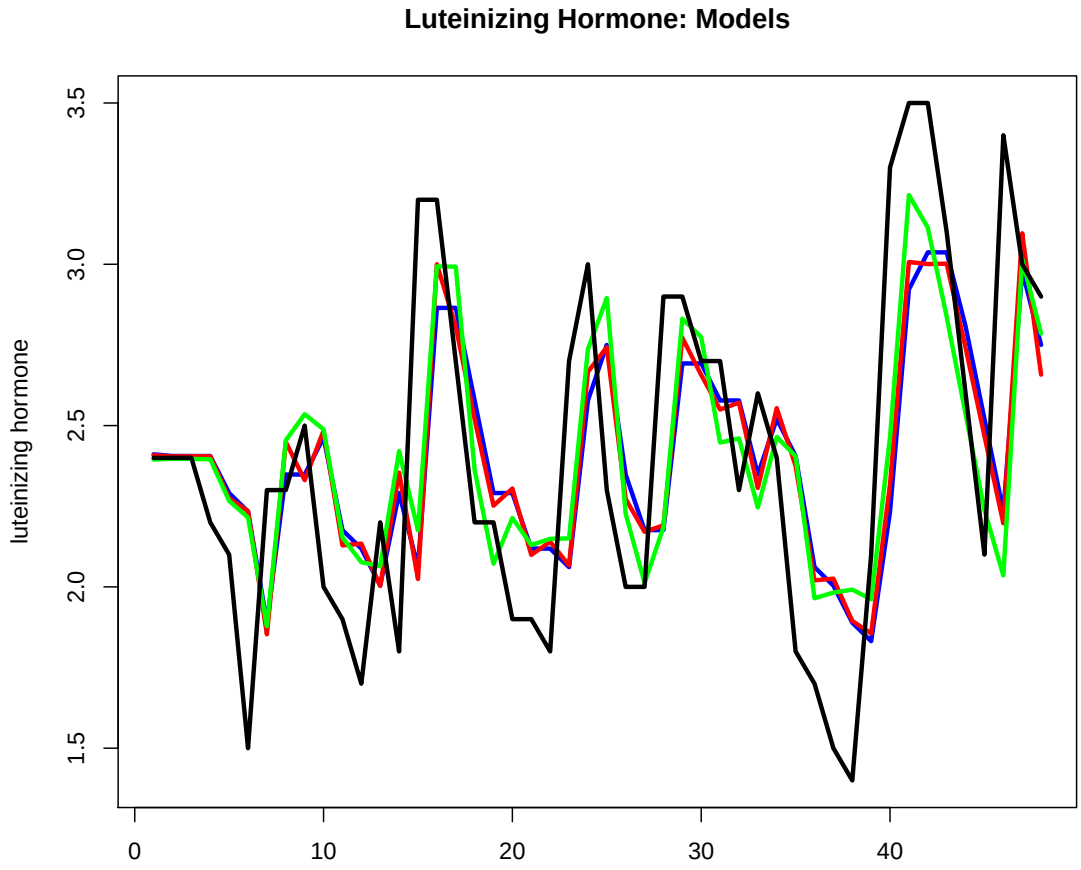


Figure 7.7: ARMA models for the luteinizing hormone time series. The original data is shown as black line while the models are (ar=1,ma=0)=blue, (ar=1,ma=1)=red, and (ar=3,ma=0)=green. The models follow the original data with one time step delay.

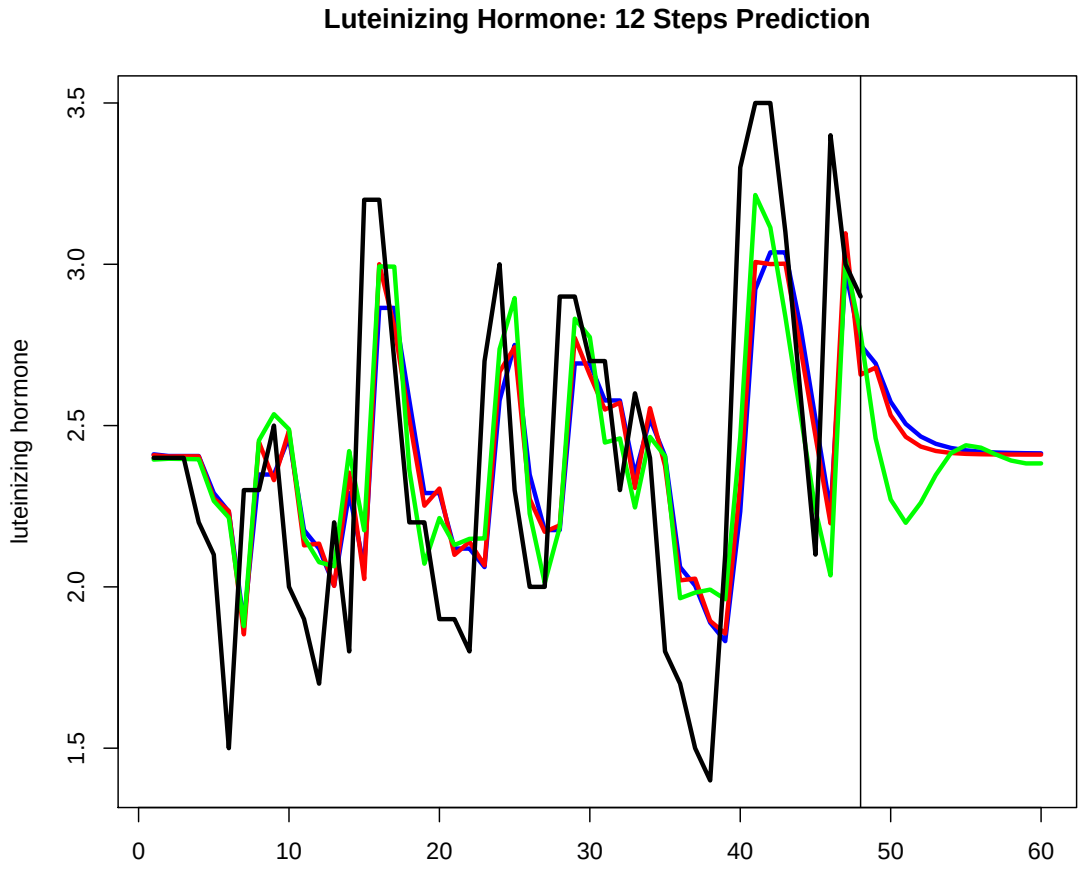


Figure 7.8: Prediction of 12 step into the future for the luteinizing hormone time series. The vertical bar shows when the prediction starts (at time point 49). The models are (ar=1,ma=0)=blue, (ar=1,ma=1)=red, and (ar=3,ma=0)=green.

We fit 5 ARIMA models to this data with (AR, I, MA) components: fit1=(0,1,1), fit2=(1,1,1), fit3=(3,1,1), fit4=(1,1,3), and fit5=(3,1,3). Since it is an integrative model, the differences must be predicted and just following the original time series is not possible.

```
fit1:
Coefficients:
          ma1     sma1
      -0.4303  -0.5528
s.e.   0.1228   0.1784

sigma^2 estimated as 99347:  log likelihood = -425.44,  aic = 856.88


########
fit2:
Coefficients:
         ar1      ma1    sar1     sma1
      0.0762  -0.4867  0.3004  -0.9940
s.e.  0.2967   0.2601  0.1776   1.2268

sigma^2 estimated as 81497:  log likelihood = -424.99,  aic = 859.98


########
fit3:
Coefficients:
          ar1      ar2     ar3      ma1     sar1     sar2     sar3     sma1
      -0.1606  -0.1376  0.0195  -0.2214  -0.2675  -0.2339  -0.2536  -0.2665
s.e.      NaN   0.0019  0.0003   0.1452   0.0018   0.0007      NaN   0.1665

sigma^2 estimated as 91788:  log likelihood = -423.97,  aic = 865.95


########
fit4:
Coefficients:
          ar1     ma1      ma2     ma3     sar1    sma1     sma2     sma3
      -0.8721  0.4861  -0.4505  0.0308  -0.7942  0.1136  -0.7357  -0.3171
s.e.   0.1550  0.3193   0.1773  0.1523      NaN     NaN      NaN      NaN

sigma^2 estimated as 75558:  log likelihood = -424.06,  aic = 866.12


########
fit5:
Coefficients:
          ar1      ar2     ar3     ma1     ma2      ma3     sar1     sar2
```

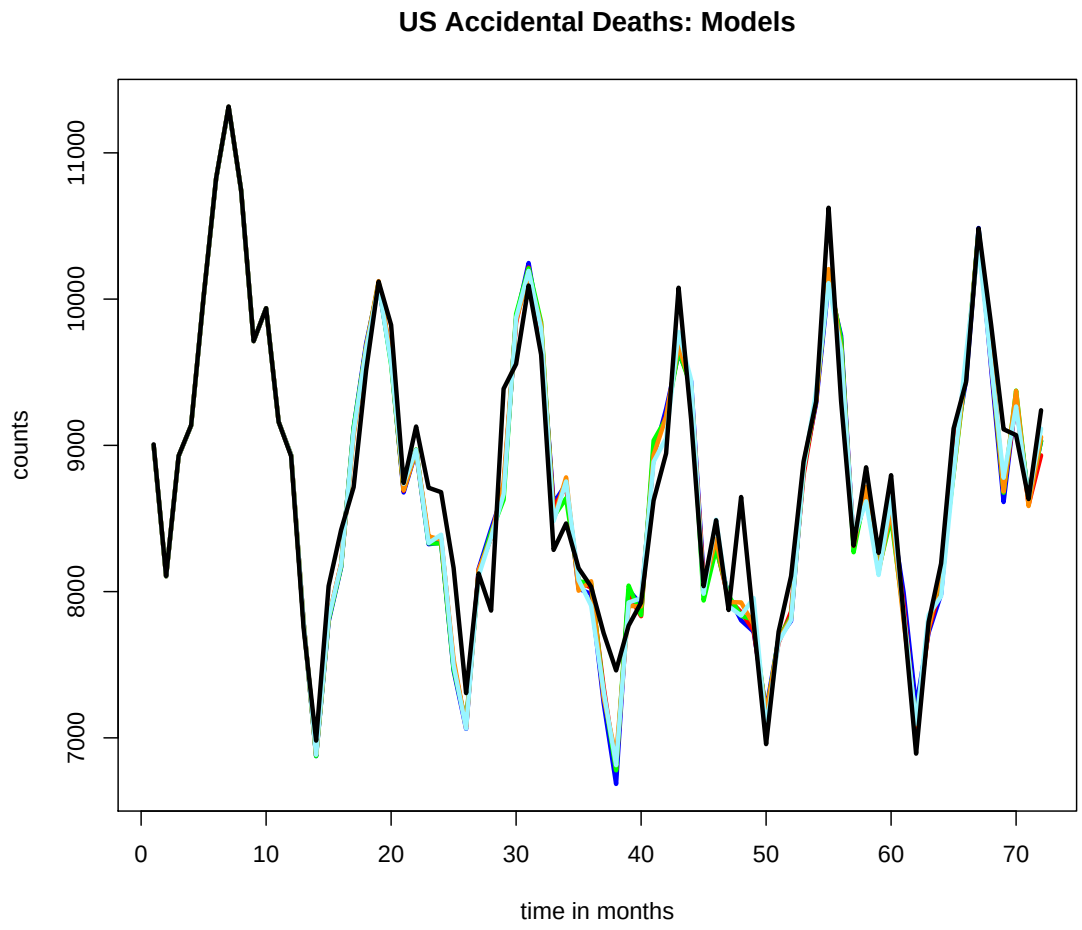


Figure 7.9: ARMA models for the US accidental deaths time series. The original data is shown as black line while the models are (ar=0,ma=1)=blue, (ar=1,ma=1)=red, (ar=3,ma=1)=green, (ar=1,ma=3)=orange, and (ar=3,ma=3)=lightblue (cadetblue). The models are very similar to each other and a clear difference cannot be observed.

```
      -0.8547  -0.6228  0.3131  0.5153  0.2923  -0.6215  -0.3115  -0.6228
s.e.   0.3840   0.4497  0.3823  0.3286  0.3601   0.3153   1.1681   0.5155
         sar3     sma1    sma2     sma3
      -0.3086  -0.2422  0.4987  -0.1312
s.e.   0.4378   1.2299     NaN     NaN

sigma^2 estimated as 80785:  log likelihood = -422.22,  aic = 870.45
```

In this case the AIC prefers the last ARIMA model (3,1,3) with both autoregressive and moving average having 3 components. Fig. 7.9 shows the models and the original data. The models are very similar to each other and a clear difference cannot be observed.

For the prediction we have the first six months of 1979 available to validate the quality of the predictions. Fig. 7.10 shows the predictions of the different models together with the true values.

True observations from the first six months of 1979 were available to check the quality of the prediction. Again a clear difference between the models cannot be seen.

7.2.2.3 Approval Ratings of US Presidents

This time series is the quarterly approval ratings of US presidents from the first quarter of 1945 to the last quarter of 1974 (120 values). The data are actually a fudged version of the approval ratings stemming from the Gallup Organization and analyzed in D. R. McNeil (1977) "Interactive Data Analysis".

The data contains missing values:

```
     Qtr1 Qtr2 Qtr3 Qtr4
1945   NA   87   82   75
1946   63   50   43   32
1947   35   60   54   55
1948   36   39   NA   NA
1949   69   57   57   51
1950   45   37   46   39
1951   36   24   32   23
1952   25   32   NA   32
1953   59   74   75   60
1954   71   61   71   57
1955   71   68   79   73
1956   76   71   67   75
1957   79   62   63   57
1958   60   49   48   52
1959   57   62   61   66
1960   71   62   61   57
1961   72   83   71   78
1962   79   71   62   74
1963   76   64   62   57
1964   80   73   69   69
1965   71   64   69   62
1966   63   46   56   44
1967   44   52   38   46
1968   36   49   35   44
1969   59   65   65   56
1970   66   53   61   52
1971   51   48   54   49
1972   49   61   NA   NA
1973   68   44   40   27
1974   28   25   24   24
```

We fit 4 ARIMA models to this data with (AR, I, MA) components: fit1=(1,0,0), fit2=(1,0,1), fit3=(3,0,0), and fit4=(3,1,0).

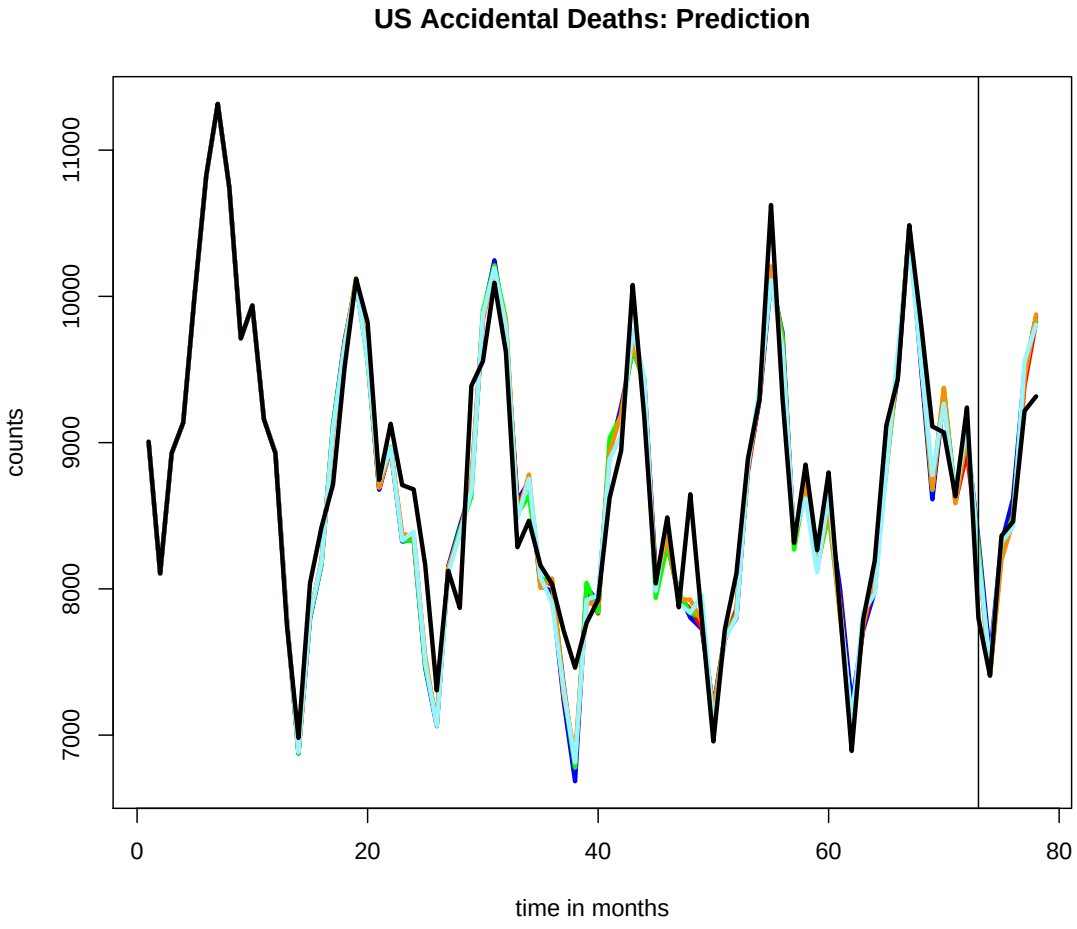


Figure 7.10: Predictions of ARIMA models for the US accidental deaths time series. The original data is shown as black line while the models are (ar=0,ma=1)=blue, (ar=1,ma=1)=red, (ar=3,ma=1)=green, (ar=1,ma=3)=orange, and (ar=3,ma=3)=lightblue (cadetblue). The black vertical line indicates where the prediction starts. True observations from the first six months of 1979 were available to check the quality of the prediction. Again a clear difference between the models cannot be seen.

```
#######
fit1:
Coefficients:
         ar1  intercept
      0.8242    56.1505
s.e.  0.0555     4.6434

sigma^2 estimated as 85.47:  log likelihood = -416.89,  aic = 839.78


#######
fit2:
Coefficients:
         ar1      ma1  intercept
      0.8629  -0.1092    56.0745
s.e.  0.0597   0.1018     5.2207

sigma^2 estimated as 84.72:  log likelihood = -416.32,  aic = 840.63


#######
fit3:
Coefficients:
         ar1     ar2      ar3  intercept
      0.7496  0.2523  -0.1890    56.2223
s.e.  0.0936  0.1140   0.0946     4.2845

sigma^2 estimated as 81.12:  log likelihood = -414.08,  aic = 838.16


#######
fit4:
Coefficients:
          ar1     ar2      ar3
      -0.1646  0.0675  -0.1731
s.e.   0.0969  0.0933   0.0949

sigma^2 estimated as 84.66:  log likelihood = -412.55,  aic = 833.1
```

The AIC of the model (1,0,1) is largest closely followed by the model (1,0,0). As the moving average component is very small, the last data point was most important for predicting the next data point. Therefore we expect that the time series is approximated by delaying it for one time point (what else can only the past data point predict?). Fig. 7.11 shows the models and the fitted values. indeed, the models approximate the time series by using the original data with one time step delay. Also the ARIMA model has one time step delay.

The quality of fitting can be checked via the residuals and their auto-correlation. The auto-

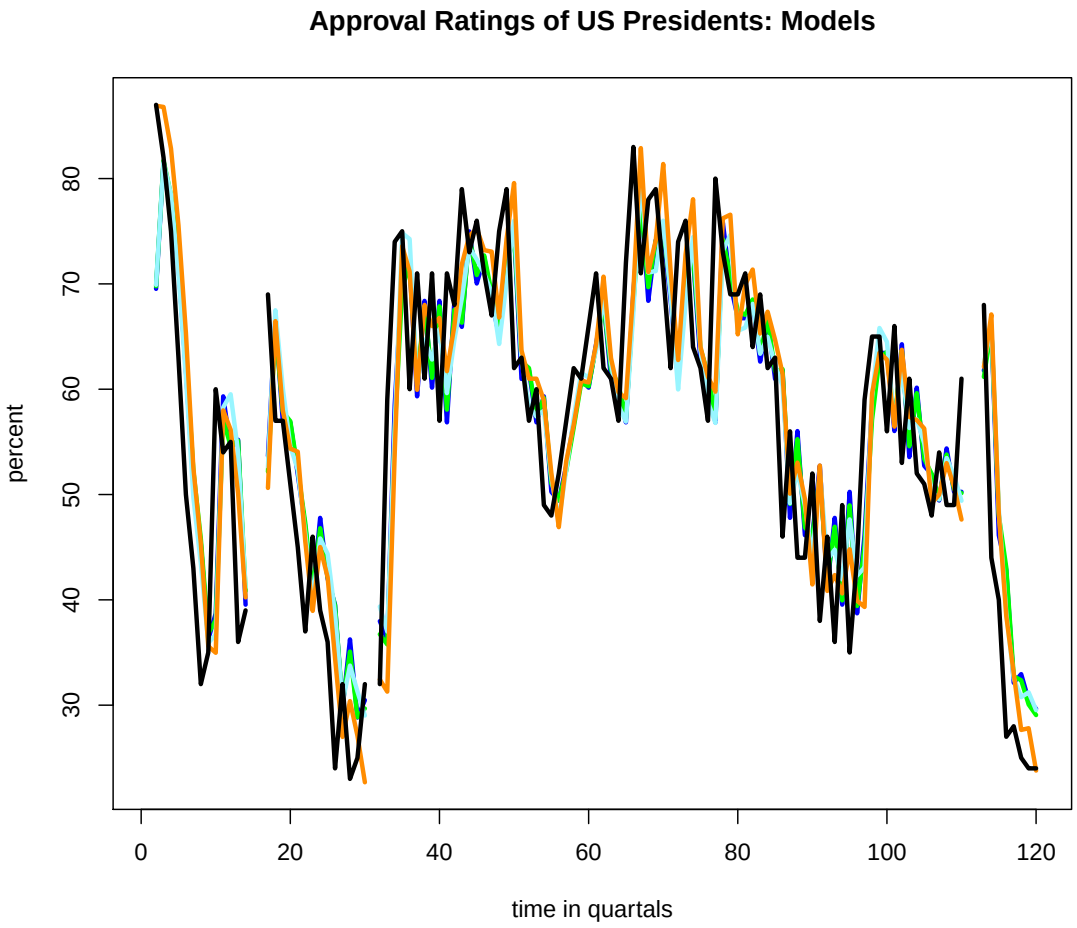


Figure 7.11: ARIMA models for the approval ratings of US presidents. The original data is shown as black line while the models are (ar=1,ma=0)=blue, (ar=1,ma=1)=green, (ar=3,ma=0)=lightblue (cadetblue), and the ARIMA with integration (ar=3,ma=0)=orange. The models approximate the time series by using the original data with one time step delay. Also the ARIMA model has one time step delay.

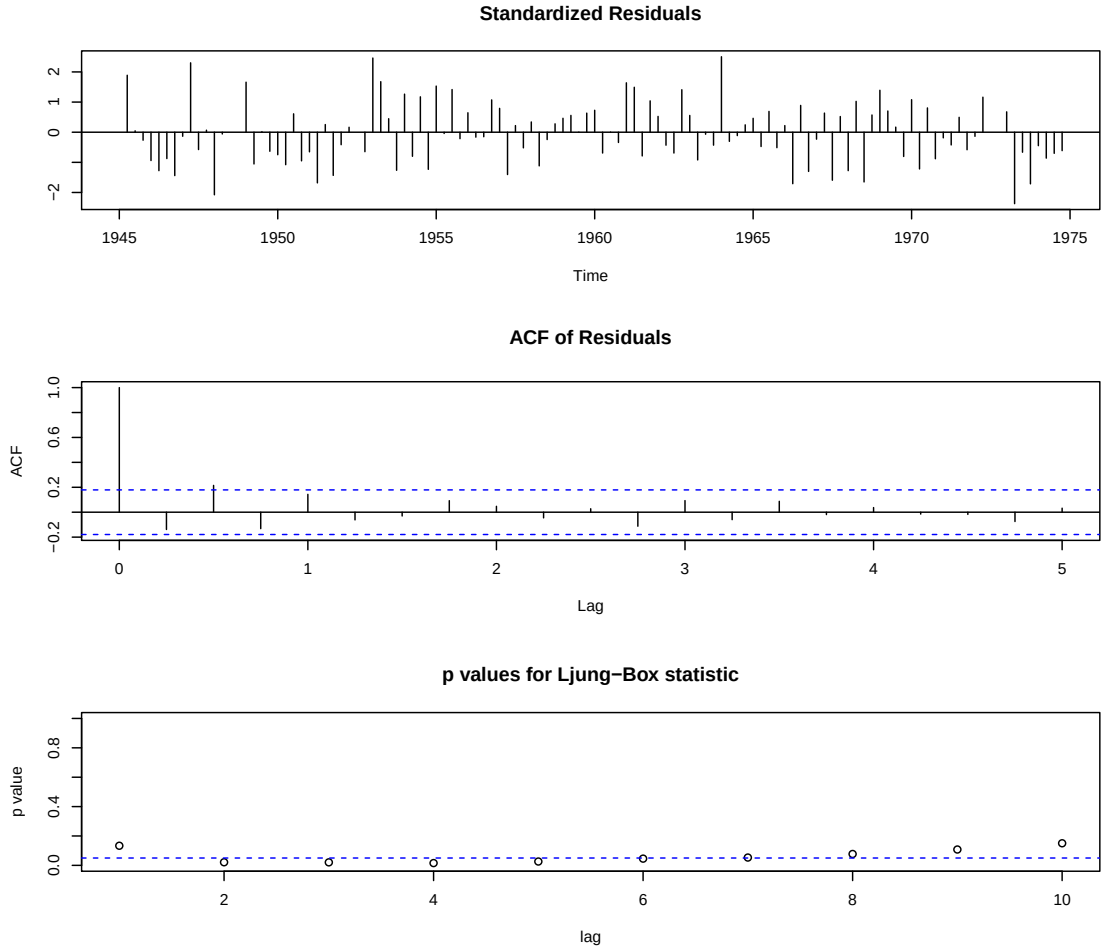


Figure 7.12: Analysis of the (1,0,0) ARIMA model where the standardized residuals, ACF of residuals, p-values for the Ljung-Box statistic are given.

correlation (ACF) plot of the residuals shows whether there is information that has not been accounted for in the model. The Ljung-Box test is a test of whether auto-correlations of a time series are different from zero. For ARIMA models this test can be used to test the null hypothesis that the residuals contain no auto-correlation. These tests are provided as a plot by the R function `tsdiag()`. The model analyses are shown in Fig. 7.12 for the (1,0,0) ARIMA model, in Fig. 7.13 for the (1,0,1) ARIMA model, in Fig. 7.14 for the (3,0,0) ARIMA model, and in Fig. 7.15 for the (3,1,0) ARIMA model. These diagnostic plots show that the later two models contain less auto-correlation in the residuals and, therefore, capture more structure in the time series.

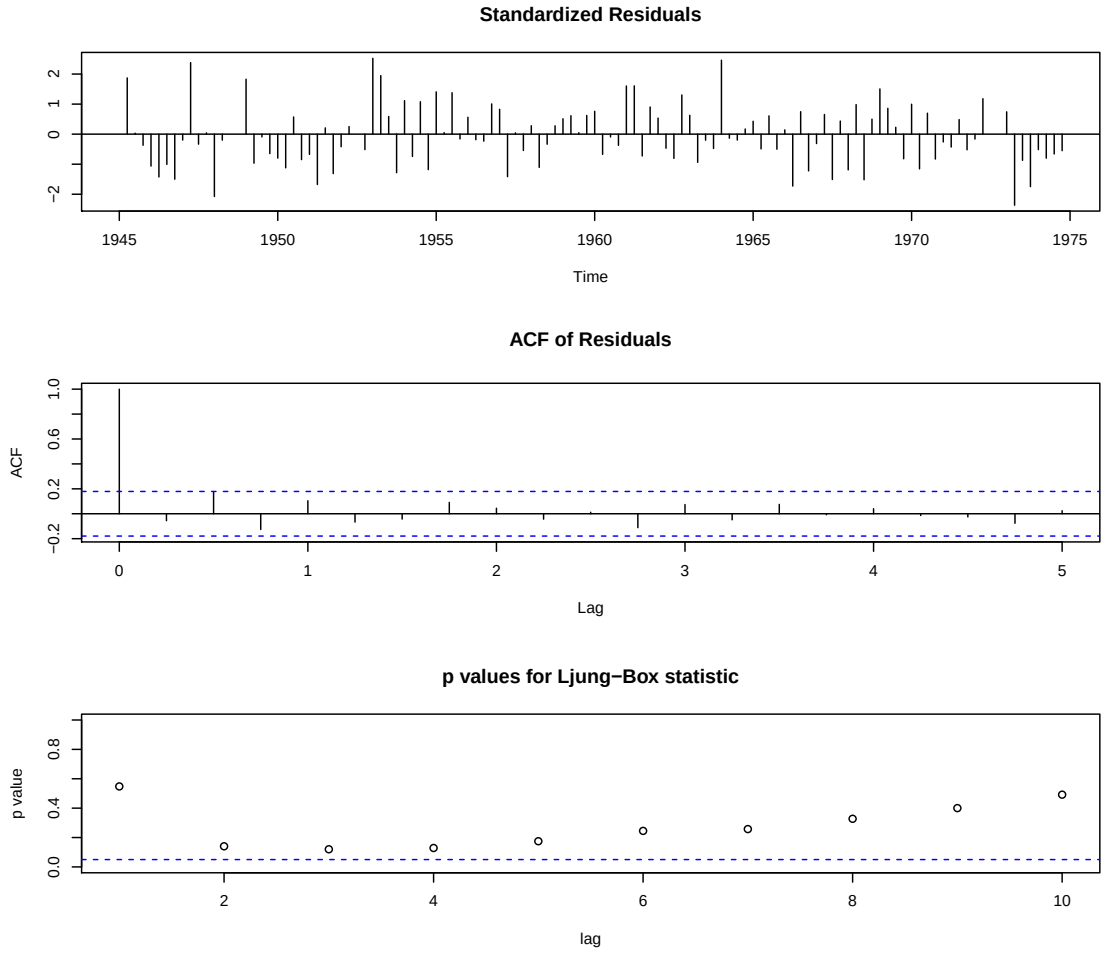


Figure 7.13: Analysis of the (1,0,1) ARIMA model where the standardized residuals, ACF of residuals, p-values for the Ljung-Box statistic are given.

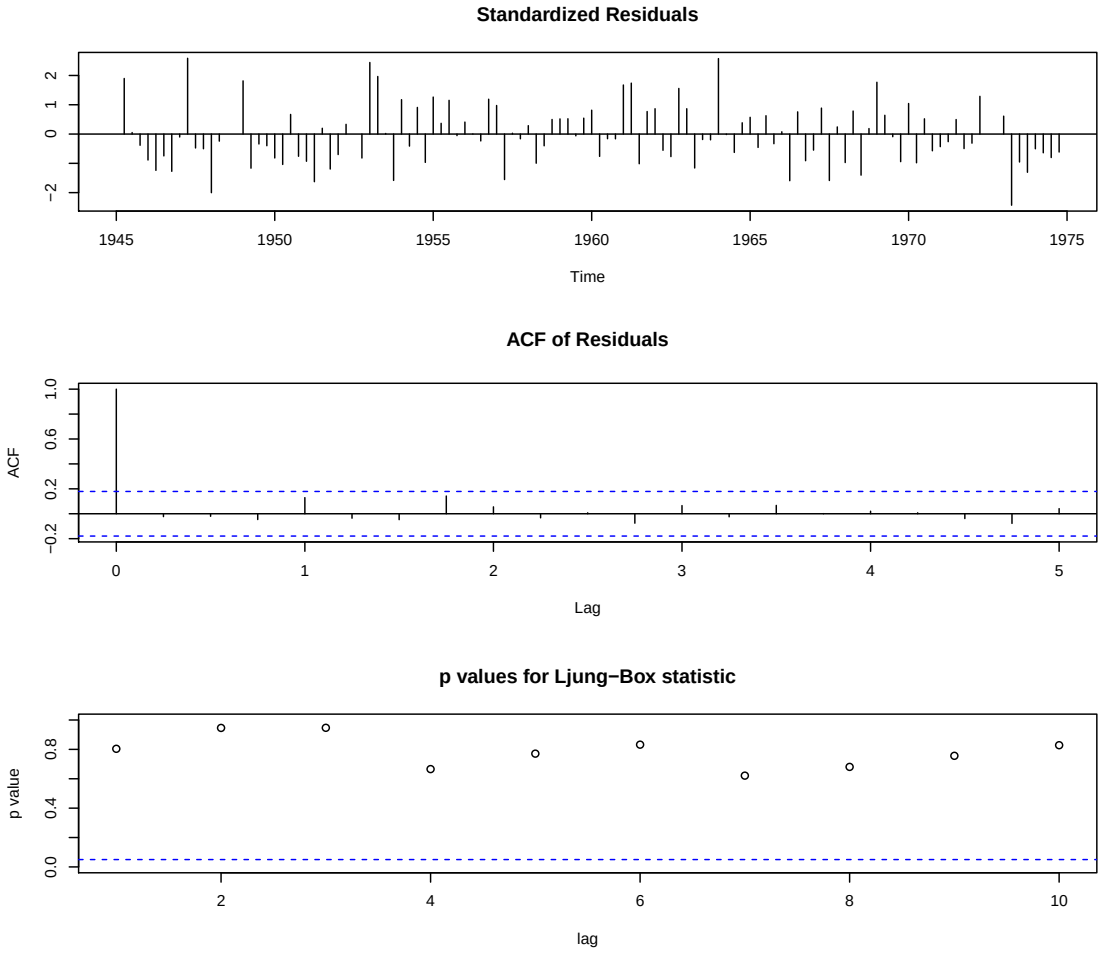


Figure 7.14: Analysis of the (3,0,0) ARIMA model where the standardized residuals, ACF of residuals, p-values for the Ljung-Box statistic are given.

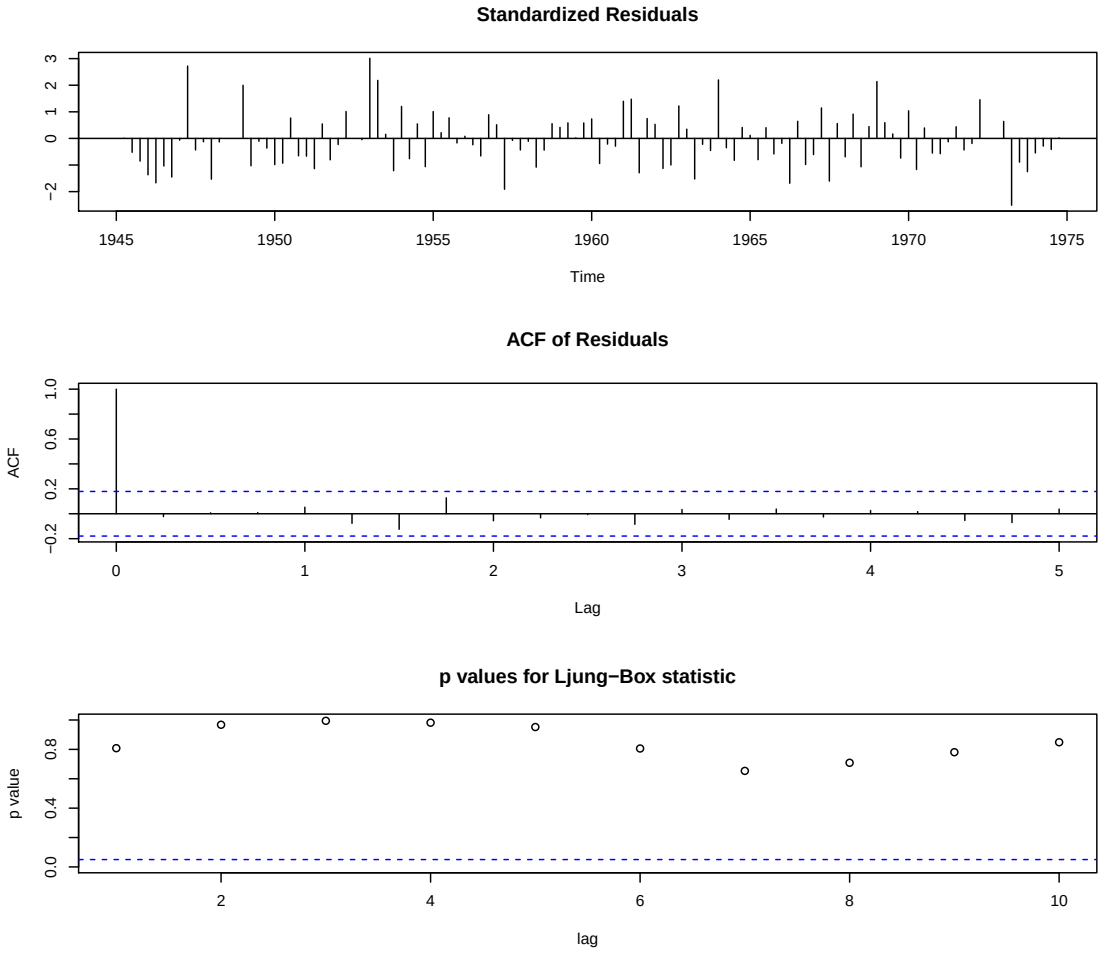


Figure 7.15: Analysis of the (3,1,0) ARIMA model where the standardized residuals, ACF of residuals, p-values for the Ljung-Box statistic are given.

7.2.2.4 Level of Lake Huron

This time series reports the annual measurements of the level, in feet, of Lake Huron from 1875 to 1972. The time series is over 98 years, therefore has 98 data points. The data is from P. J. Brockwell and R. A. Davis (1996) "Introduction to Time Series and Forecasting". The data are:

```
580.38 581.86 580.97 580.80 579.79 580.39 580.42 580.82 581.40 581.32
581.44 581.68 581.17 580.53 580.01 579.91 579.14 579.16 579.55 579.67
578.44 578.24 579.10 579.09 579.35 578.82 579.32 579.01 579.00 579.80
579.83 579.72 579.89 580.01 579.37 578.69 578.19 578.67 579.55 578.92
578.09 579.37 580.13 580.14 579.51 579.24 578.66 578.86 578.05 577.79
576.75 576.75 577.82 578.64 580.58 579.48 577.38 576.90 576.94 576.24
576.84 576.85 576.90 577.79 578.18 577.51 577.23 578.42 579.61 579.05
579.26 579.22 579.38 579.10 577.95 578.12 579.75 580.85 580.41 579.96
579.61 578.76 578.18 577.21 577.13 579.10 578.25 577.91 576.89 575.96
576.80 577.68 578.38 578.52 579.74 579.31 579.89 579.96
```

We fit 4 ARIMA models to this data with (AR, I, MA) components: fit1=(2,0,0), fit2=(2,0,2), fit3=(4,0,4), and fit4=(4,1,0).

```
#######
fit1:
Coefficients:
         ar1      ar2  intercept  time(LakeHuron) - 1920
      1.0048  -0.2913   579.0993                -0.0216
s.e.  0.0976   0.1004     0.2370                 0.0081

sigma^2 estimated as 0.4566:  log likelihood = -101.2,  aic = 212.4


#######
fit2:
Coefficients:
         ar1     ar2     ma1     ma2  intercept  time(LakeHuron) - 1920
      0.5614  0.0127  0.4612  0.1134   579.1059                -0.0213
s.e.  1.0890  0.7267  1.0832  0.4043     0.2497                 0.0085

sigma^2 estimated as 0.4547:  log likelihood = -101,  aic = 216


#######
fit3:
Coefficients:
         ar1      ar2     ar3     ar4  intercept  time(LakeHuron) - 1920
      1.0229  -0.3495  0.0461  0.0189   579.1100                -0.0211
s.e.  0.1025   0.1508  0.1511  0.1061     0.2587                 0.0088
```

```
sigma^2 estimated as 0.4546:  log likelihood = -100.99,  aic = 215.98


#######
fit4:
Coefficients:
        ar1      ar2      ar3      ar4  time(LakeHuron) - 1920
      0.127  -0.2150  -0.1343  -0.0943                 -0.0063
s.e.  0.103   0.1055   0.1044   0.1065                  0.0552

sigma^2 estimated as 0.5027:  log likelihood = -104.39,  aic = 220.78
```

The last, integrative model gives the best AIC value. The other models are dominated by ar1 or ma1, therefore they only follow the time series with a delay. Fig. 7.16 shows the different models fitted to the original data. Again all models follow with one time step delay the original time series. The ARIMA model based on differences seems to be closer to the time series at some time points.

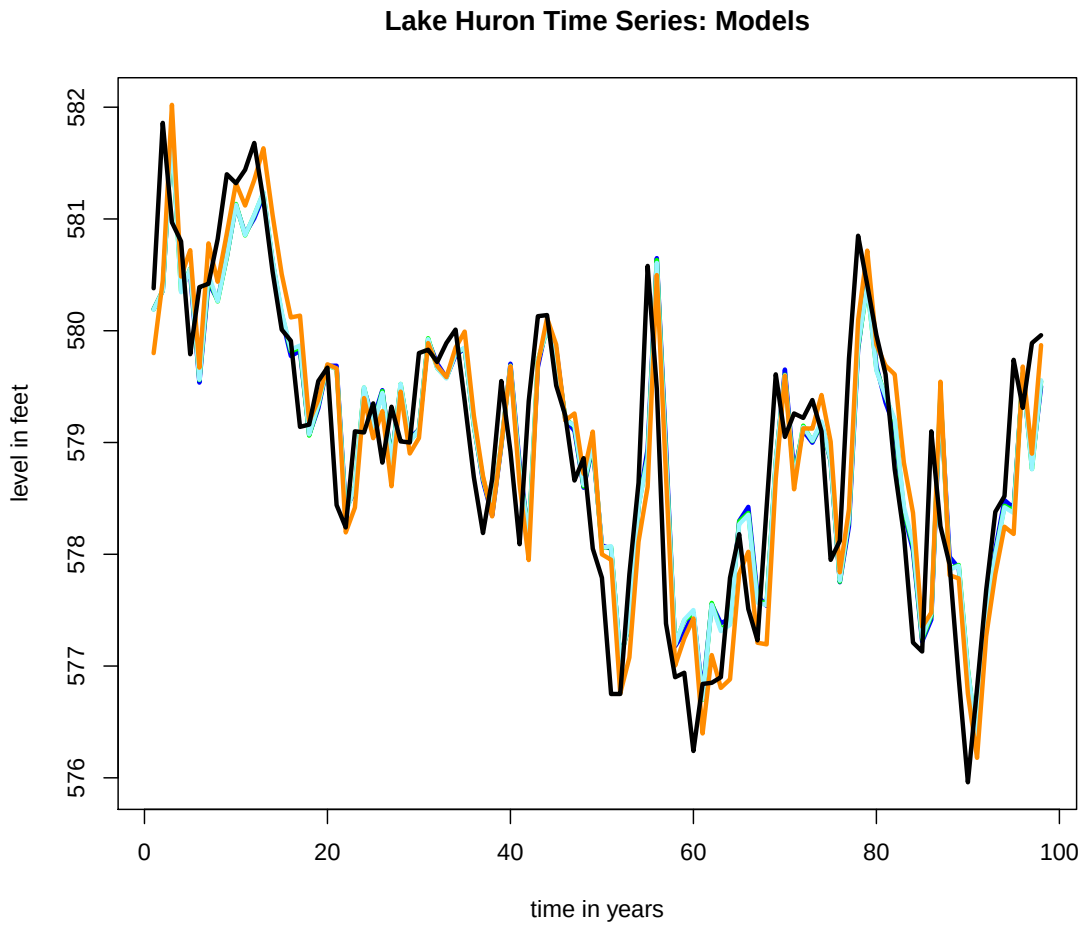


Figure 7.16: ARIMA models for the measurements of the level, in feet, of Lake Huron. The original data is shown as black line while the models are (ar=2,ma=0)=blue, (ar=2,ma=2)=green, (ar=4,ma=0)=lightblue (cadetblue), and the ARIMA with integration (ar=4,ma=0)=orange. Again all models follow with one time step delay the original time series. The ARIMA model based on differences seems to be closer to the time series at some time points.

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.

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.

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. 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. 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.

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.

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.

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.

S. Degroeve, B. De Baets, Y. Van de Peer, and P. Rouz. Feature subset selection for splice site prediction. *Bioinformatics*, 18:S75–S83, 2002.

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.

S.R. Eddy and R. Durbin. Maximum discrimination hidden Markov models of sequence consensus. *J. Comp. Biol.*, 2:9–23, 1995.

J. L. Elman. Finding structure in time. Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.

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.

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. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

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.

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.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the Thirtyth International Conference Machine Learning (ICML'2013)*, 2013.

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 M. Bittner, Y. Chen, A. Dorsel, and E. Dougherty, editors, *Proc. SPIE Int. Symp. on Biomedical Optics*, volume 4266 of *Microarrays: Optical Technologies and Informatics*, pages 132–140, 2001.

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, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, New York, 2001.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdino. Improving neural networks by preventing co-adaptation of feature detectors. Technical report arXiv:1207.0580, arXiv, 2012.

S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

S. Hochreiter. Long short-term memory. Advances in Neural Information Processing Systems 9 — Workshop "Dynamic Recurrent Nets" — Talk with discussion, 1997a.

S. Hochreiter. Recurrent neural net learning and vanishing gradient. In C. Freksa, editor, *Proceedings in Artificial Intelligence — Fuzzy-Neuro-Systeme 97*, pages 130–137. INFIX, Sankt Augustin, Germany, 1997b.

S. Hochreiter and M. C. Mozer. An electric field approach to independent component analysis. In P. Pajunen and J. Karhunen, editors, *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, Helsinki, Finland*, pages 45–50. Otamedia, Espoo, Finland, ISBN: 951-22-5017-9, 2000.

S. Hochreiter and M. C. Mozer. Beyond maximum likelihood and density estimation: A sample-based criterion for unsupervised learning of complex models. In T. K. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, 2001a.

S. Hochreiter and M. C. Mozer. Coulomb classifiers: Reinterpreting SVMs as electrostatic systems. Technical report, University of Colorado, Boulder, Department of Computer Science, 2001b.

S. Hochreiter and M. C. Mozer. Coulomb classifiers: Reinterpreting SVMs as electrostatic systems. Technical Report CU-CS-921-01, Department of Computer Science, University of Colorado, Boulder, 2001c.

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, 2001d.

S. Hochreiter, M. C. Mozer, and K. Obermayer. Coulomb classifiers: Generalizing support vector machines via an analogy to electrostatic systems. In S. Beckers, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 545–552. MIT Press, Cambridge, MA, 2003.

S. Hochreiter and K. Obermayer. Classification of pairwise proximity data with support vectors. In Y. LeCun and Y. Bengio, editors, *The Learning Workshop*. Computational & Biological Learning Society, Snowbird, Utah, 2002.

S. Hochreiter and K. Obermayer. Feature selection and classification on matrix data: From large margins to small covering numbers. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 889–896. MIT Press, Cambridge, MA, 2003.

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, 2004a.

S. Hochreiter and K. Obermayer. Sphered support vector machine. Technical report, Technische Universität Berlin, Fakultät für Elektrotechnik und Informatik, 2004b.

S. Hochreiter and K. Obermayer. Nonlinear feature selection with the potential support vector machine. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature extraction, Foundations and Applications*. Springer, 2005.

S. Hochreiter and J. Schmidhuber. Flat minimum search finds simple nets. Technical Report FKI-200-94, Fakultät für Informatik, Technische Universität München, 1994.

S. Hochreiter and J. Schmidhuber. Bridging long time lags by weight guessing and "Long Short-Term Memory". In F. L. Silva, J. C. Principe, and L. B. Almeida, editors, *Spatiotemporal models in biological and artificial systems*, pages 65–72. IOS Press, Amsterdam, Netherlands, 1996. Serie: Frontiers in Artificial Intelligence and Applications, volume 37.

S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997a.

S. Hochreiter and J. Schmidhuber. LOCOCODE. Technical Report FKI-222-97, Fakultät für Informatik, Technische Universität München, 1997b.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997c.

S. Hochreiter and J. Schmidhuber. Low-complexity coding and decoding. In K. M. Wong, I. King, and D. Yeung, editors, *Theoretical Aspects of Neural Computation (TANC 97), Hong Kong*, pages 297–306. Springer, 1997d.

S. Hochreiter and J. Schmidhuber. Low-complexity coding and decoding. In K. M. Wong, I. King, and D. Yeung, editors, *Theoretical Aspects of Neural Computation (TANC 97), Hong Kong*, pages 297–306. Springer, 1997e.

S. Hochreiter and J. Schmidhuber. Low-complexity coding and decoding. In K. M. Wong, I. King, and D. Yeung, editors, *Theoretical Aspects of Neural Computation (TANC 97), Hong Kong*, pages 297–306. Springer, 1997f.

S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 473–479. MIT Press, Cambridge MA, 1997g.

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, 1997h.

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, 1997i.

S. Hochreiter and J. Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In J.-F. Cardoso, C. Jutten, and P. Loubaton, editors, *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation, Aussois, France*, pages 149–154, 1999a.

S. Hochreiter and J. Schmidhuber. Nonlinear ICA through low-complexity autoencoders. In *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems (ISCAS'99)*, volume 5, pages 53–56. IEEE, 1999b.

S. Hochreiter and J. Schmidhuber. Source separation as a by-product of regularization. In M. S. Kearns, S. A. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 459–465. MIT Press, Cambridge, MA, 1999c.

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.

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. 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.

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.

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.

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. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 24*, 2012.

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, 1994.

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.

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.

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.

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 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. Møller. 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.

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.

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.

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.

E. E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. Technical Report AIM No. 1602, CBCL no. 144, MIT, 1997.

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.

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.

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, 1992.

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.

E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Decomposing gene expression into cellular processes. In *Pac. Symp. Biocomputing*, pages 89–100, 2003.

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.

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.

P. Vincent. A connection between score matching and denoising autoencoders. *Neural Computatio*, 23(7):1661–1674, 2011.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the Twentyfifth International Conference Machine Learning (ICML'2013)*, 2008.

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.

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.

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.