Support Vector Machines in Bioinformatics

Florian Markowetz

Diplomarbeit
im Studiengang Mathematik
- revised version -

Betreuer: Prof. Dr. E. Mammen

Mathematische Fakultät der
Ruprecht-Karls Universität Heidelberg

Heidelberg, January 24, 2002
[Abgabetermin 26.04.2001]

Acknowledgements

This work was written as my diploma thesis in mathematics at the University of Heidelberg under the supervision of Prof. Dr. Enno Mammen. I wish to thank him for his help and advice.

The research was carried out at the Department of Theoretical Bioinformatics (TBI) at the German Cancer Research Center (DKFZ) Heidelberg. I am very much indebted to Dr. Martin Vingron (now a director at the MPI Molecular Genetics Berlin/Dahlem) for this opportunity. r I want to thank both Prof. Dr. Enno Mammen and Dr. Martin Vingron for the cooperation between the Institute for Applied Mathematics and the TBI.

In Tobias Müller I found a very competent advisor. I owe him much for his assistance in all aspects of my work. Thank you very much!

Dr. Lutz Edler, head of the Biostatistics Unit at the DKFZ, kindly allowed me to use his data and showed great interest in the progress of my work. Parts of this work will be presented by Dr. Lutz Edler at the conference on *Statistics in Functional Genomics*, June 2001 at Mount Holyoke College. The article (Markowetz et al., 2001) contains material from this thesis.

I have very much enjoyed working at the TBI and can hardly imagine a better working environment. I have learned a lot from the group meetings and the discussions with my colleagues. Thanks, everyone, for the good cooperation and willingness to help!

Jede Statistik, jede rein deskriptive oder informative Arbeit beruht auf der großartigen und vielleicht unsinnigen Hoffnung, in der weitläufigen Zukunft könnten Menschen wie wir, nur hellsichtiger, mit Hilfe der von uns hinterlassenen Daten zu einer glücklichen Schlußfolgerung oder einer bemerkenswerten Generalisierung gelangen.

Jorge Luis Borges (*1899 Buenos Aires, †1986 Genf),
aus "Ein Nachmittag mit Ramón Bonavena".

Contents

Notations

x^t	vector x transposed
$\langle \cdot, \cdot \rangle$	euclidean inner product
I	identity matrix
$\mathbb{I}_{[A]}(\cdot)$	indicator function of A
$sign(x)$	equals $+1$ if $x \geq 0$ and -1 otherwise
x_i	datapoints
y_i	class of the datapoint x_i
$\mathcal{X}$	training set $\{(x_i, y_i) : \ i = 1, \dots, l\}$
$R(\cdot)$	risk functional
$R_{emp}(\cdot)$	empirical risk
(w, b)	define the hyperplane $H = \{x : \langle w, x \rangle + b = 0\}$
γ_i	margin of a single point x_i
$\gamma_{\mathcal{S}}$	margin of a whole set of points $\mathcal{S}$
L_P, L_D	Lagrangian: primal and dual
α_i	Lagrange multipliers
ξ_i	slack variables (for linear-nonseparable datasets)
C	error weight
h	Vapnik-Chervonenkis dimension
$k(\cdot, \cdot)$	kernel function
K	kernel matrix $(k(x_i, x_j))$
$\mathcal{L}$	(low-dimensional) input space
$\mathcal{H}$	(high-dimensional) feature space
SVM	Support Vector Machines
SV	support vectors
rbf	radial basis functions
rbf-SVM	Support Vector Machine with radial basis function kernel
poly$k$-SVM	Support Vector Machine with polynomial kernel of degree k
ERM	Empirical Risk Minimization
SRM	Structural Risk Minimization
APE	apparent prediction error (error on the training set)
TPE	test prediction error (error on the test set)
CV(k)	k-fold cross validation error

Chapter 0

Introduction

The initial goal of the Human Genome Project to produce a draft sequence of the human genome has been achieved in summer 2000 and an advanced assembly is reported in february 2001. But knowledge of the genomic sequence is just the first step towards an understanding of how an organism develops and functions. The next key landmark will be an overview of the characteristics and activities of the proteins encoded in the genes. Since not all genes are expressed at the same time, a further question is which genes are active under which circumstances.

To answer any of these questions, advanced mathematical and statistical methods are needed. Analysis of gene expression is done by use of DNA microarrays. The data thus obtained makes great demands on statistical models and interpretation. The function of a protein strongly depends on its structure. However, there is a large gap between the number of known protein sequences and the number of identified structures. Methods of statistical classification can help to bridge this gap.

In this work, we describe Support Vector Machines, a method of statistical learning theory recently introduced to bioinformatics, and compare its performance in the prediction of protein fold classes to the results obtained by various other methods of statistical classification.

Support Vector Machines (SVM) for Classification are a rather new concept in Learning Theory. It's origins reach back to the early 60's (Vapnik and Lerner, 1963; Vapnik and Chervonenkis, 1964), but it stirred up attention only in 1995 with Vladimir Vapnik's book *The Nature of Statistical Learning Theory* (Vapnik, 1995). In the last few years Support Vector Machines proofed excellent performance in many real-world applications such as text categorisation, hand-written character recognition, image classification or biological sequence analysis.

Because many aspects in the theory of Support Vector Machines are still under intensive research, the number of introductory literature is limited. The two books by Vladimir Vapnik (*The Nature of Statistical Learning Theory* (Vapnik, 1995) and *Sta-*

tistical Learning Theory (VAPNIK, 1998)) present a general high-level introduction to statistical inference including SVM. The first tutorial purely on Support Vector Machines was written by C. Burges in 1998 (BURGES, 1998). Last year CRISTIANINI and SHAWE-TAYLOR (2000) published *An Introduction to Support Vector Machines*.

These books and tutorials provide a very good view over the theory of Support Vector Machines, but they don't give a straightforward introduction to application. Either they come to the point very quickly, but lack mathematical details in optimization theory and kernel techniques (like (BURGES, 1998)), or the reader has to work through hundred pages or more before the idea of SVM is explained to him ((VAPNIK, 1998; CRISTIANINI and SHAWE-TAYLOR, 2000)). In applications, the extension of binary classification to n-class problems becomes important, and you often have to deal with very imbalanced datasets, where examples of one class are much more numerous than examples of the other class. These topics are not treated in detail by the books above. Against this background, we decided to give a thorough introduction to classification by SVM, ranging from the basic terminology and mathematical tools to advanced topics in generalization theory and application. We will show that SVM outperform classical methods of classification on a dataset of protein sequences.

Part I. The work is divided into three parts. The first one introduces into Supervised Learning Theory. Chapter 1 gives a general model for learning from examples. It presents two concepts of risk minimization (*Empirical Risk Minimization* and *Structural Risk Minimization*) and shortly describes alternative models of learning.

Chapter 2 explains the fundamental terminology used in Machine Learning. Terms like *generalization ability*, *capacity* and *curse of dimensionality* are widely used without proper introduction. We study in detail the VC dimension as a measure of capacity and compare it to the number of parameters (the dimensionality).

The basic tools for attacking the learning problem are provided by optimization theory, so we state the fundamental theorems of constrained optimization in chapter 3. We emphasize the dual description of optimization problems and the so called Kuhn-Tucker conditions, which play a central role in the theory of Support Vector Machines.

Part II describes the theory of Support Vector Machines as a combination of two main concepts: *Maximal Margin Hyperplanes* (also called *Optimal Separating Hyperplanes*) and *kernel functions*. This account is focussed on SVM for classification, so Support Vector Regression or the connection to other hyperplane classifiers like the perceptron are not treated.

In chapter 4, the simplest form of SVM is introduced as a hyperplane maximizing the margin of separation on linear separable data sets (*Hard Margin SVM*). This concept is then generalized to allow errors in the training phase (*Soft Margin SVM*). We present two ways to implement training errors, which result in different optimization problems (*1-Norm Soft Margin* and *2-Norm Soft Margin*).

By combining Maximum Margin Hyperplanes with implicit mappings to high dimensional feature spaces we gain nonlinear classifiers in chapter 5. These mappings are called *kernel functions* and allow to do a linear separation of the data, but to still come up with a non-linear classifier. We start with an example, where we are able to give the kernel mapping explicitly. Then we study what general conditions must be satisfied by kernel functions. This leads to Mercer's Theorem.

Until now the data was divided into only two different classes. In chapter 6 we discuss the problem of multiclass classification. The optimization task for the 2-class case can directly be generalized to more than two classes, but we will also show two methods of performing a multi-classification using 2-class SVM. These heuristic methods are mostly used in applications.

The last chapter of part II is on generalization theory. It deals with statistical properties of separating hyperplanes and gives bounds on the probability of generalization errors. We give a bound on the actual risk of SVM in terms of the empirical risk and the VC dimension orginally obtained by V. Vapnik. Because the VC dimension of SVM will turn out to be very large or even infinite, this bound is disappointing. We will give an outlook on how this shortcoming is battled in current research and adress the question, how this affects the implementation of Structural Risk Minimization by Support Vector Machines.

Part III. The third part of this work discusses the application of support vector theory to problems in bioinformatics. Chapter 8 offers a brief summary of the biological background. The main focus lies on protein structure and its identification.

In Chapter 9, we evaluate the performance of SVM for protein fold class prediction in comparison to results obtained by other statistical classification methods, like Neural Networks or Linear Discriminant Analysis. In the first part of this chapter, we compare these methods with respect to the error rate on the training set, on the test set and in 10-fold cross validation. We lay strong emphasis on discussing our choice of parameters and on interpreting the observations made about the performance of a classifier even before any test results are obtained. We will argue, that Support Vector Machines are no black box method. They can be interpreted in terms of the Lagrange multipliers calculated by quadratic programming and the support vectors determining the separating hyperplane.

In the second part of chapter 9, we discuss the influence of different embeddings on Support Vector Machines. We embed the protein sequences by the frequencies of single, pairs and triplets of amino acids yielding input spaces of dimension 20, 400 and 8000. We will show that SVM are successfull even in this very high dimensional spaces and thus provide a way to break the curse of dimensionality. This ability lets us expect good performance of Support Vector Machines even on very difficult datasets like gene expression data.

Appendices. We give a complete list of the used dataset (Appendix A) and shortly describe the competing methods (Appendix B). In the last section we give a proof of one of the main theorems by Vladimir Vapnik (Appendix C).

Part I

Introduction to Learning Theory

Chapter 1

Supervised Learning Theory

Can a machine only do what we know how to order it to do? Do we always have to describe the connection of the inputs to the desired outputs explicitly? In many cases we want our machine to perform tasks that cannot be described by an algorithm, i.e. a sequence of explicit instructions the computer has to follow. It is not possible to solve these task by classical programming techniques, since no mathematical model of the problem is available or the computation of the solution may be very expensive. As examples consider the problem to perform hand-written digit recognition (a classical problem of machine learning), the modeling of complex chemical reactions, where the precise interactions of the reactants are unknown, or the prediction of protein folding structure based on the amino acid sequence.

What we need is an alternative approach to the solution of these problems. Maybe we can teach the machines in the same way that we teach children, i.e., not giving them abstract definitions and theories but pointing out examples of the input-output functionality. How do children learn reading the letters of the alphabet? The teacher does not give them precise definitions of each single letter, but he shows them examples. The pupils infer general properties of the letters by carefully examining these examples. At the end, they will be able to read words in script style even if they were taught only on types.

The most obvious purpose in learning a classifier is to be able to correctly classify future observations. For this even black box methods suffice. But we could also be interested in the form of the classifier *per se*, i.e. in what combinations of the features discriminate between the classes. Here we need methods with a clear interpretation of the influence of the features on the resulting classifier.

How can *learning from examples* be formalized in a mathematical setting? The solution of this problem promises great rewards because of the broad range of applications and the avoidance of laborious design and programming.

1.1 Modeling the Learning Problem

Learning from examples can be described in a general model which contains three elements: the *generator* of the data x, the *supervisor* that assigns labels y to the data and the *learning machine* that returns a value $\hat{y}$ close to the supervisor's response.

The goal is to *imitate* the supervisor: Try to construct an operator which achieves the best prediction of the supervisor's outputs based on the data provided by the generator. This differs from trying to *identify* the supervisor's operator. Identifying requires to construct an operator which is close to the supervisor's one in a given metric, good results in prediction are not enough. The problem of identification is more difficult and can only be developed in an asymptotic theory. We will concentrate on the problem of imitation. For a more detailed discussion see (Vapnik, 1998).

The generator determines the environment in which the supervisor and the learning machines act. It generates the vectors $x \in \mathbb{R}^n$ independently and identically distributed according to some unknown probability distribution $P(x)$.

The supervisor assigns the 'true' values according to a conditional distribution function $P(y|x)$. This assumption includes the case $y = f(x)$ in which the supervisor associates a fixed y with every x. In this work we will be occupied with *binary classification problems*, so the output consists in one of the values $+1$ or -1 assigning x to the positive or the negative class.

The learning machine is defined by a set of possible mappings $x \mapsto f(x, \alpha)$ where α is element of a parameter space Λ. An example of a learning machine is defined by oriented hyperplanes [1] $\{x : \langle w, x\rangle + b = 0\}$ where $\alpha \equiv (w, b) \in \mathbb{R}^{n+1}$ determines the position of the hyperplanes in $\mathbb{R}^n$. This results in a learning machine

$$LM = \{f(x,(w,b)) = sign(\langle w,x\rangle + b) : (w,b) \in \mathbb{R}^{n+1}\}.$$

The functions $f : \mathbb{R}^n \mapsto \{-1, +1\}$, mapping x to the positive or negative class, are called decision functions.

Training. A particular choice of the parameter α based on l observations

$$\mathcal{X} = \{(x_1, y_1), \ldots, (x_l, y_l) : x_i \in \mathbb{R}^n , y_i \in \{-1, +1\}\}.$$

is called *training of the machine*. The training set $\mathcal{X}$ is drawn according to $P(x, y)$. If all the data is given to the learner at the start of training, the learning process is called *batch learning*. If the learner receives one example at a time and gives an estimate of the output before receiving the correct value, it is called *on-line learning*. In this work we consider the learning methodology from batch training data.

Test. Once we have chosen a function, we want to see how well it works. The usual procedure is to split the data with known outputs in two sets. The first one is taken as

[1] From now on we will describe a hyperplane by the defining equation $\langle w, x\rangle + b = 0$ and not as the set of points $\{x : \langle w, x\rangle + b = 0\}$.

a training set and the chosen $f(x, \alpha)$ is then applied to the second one, which is called the *test set*. The number of classification errors on this set is an estimate of the overall error rate of the machine.

1.2 Risk Minimization

How do we choose a decision function from LM? A measure of quality is the expected classification error for a trained machine:

$$R(\alpha) = \int \frac{1}{2}|y - f(x, \alpha)| \, dP(x, y) \tag{1.1}$$

The function $\frac{1}{2}|y - f(x, \alpha)|$ is called the *loss*. For the case of binary classification it can only take the values 0 and 1. It measures the discrepancy between the response y of the supervisor to a given input x and the response $f(x, \alpha)$ provided by the learning machine. $R(\alpha)$ is referred to as the risk functional, or just the *risk*. The goal is to find the parameter α^* such that $f(x, \alpha^*)$ minimizes the risk over the class of functions $f(x, \alpha)$, $\alpha \in \Lambda$. Since $P(x, y)$ is unknown, we cannot compute the value of the risk for a given parameter α directly. The only available information is contained in the training set $\mathcal{X}$.

1.2.1 Empirical Risk Minimization (ERM)

The *empirical risk* is defined as

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(x_i, \alpha)|. \tag{1.2}$$

No probability distribution appears in this formula, $R_{emp}(\alpha)$ is a fixed number for a particular choice of α and for a training set $\mathcal{X} = \{(x_i, y_i)\}_{i=1,\dots,l}$. The function minimizing the empirical risk on $\mathcal{X}$ will be called $f(x, \alpha_{\mathcal{X}})$. This leads to the principle of empirical risk minimization (ERM).

Definition 1.1 (Empirical Risk Minimization) Replace the risk functional $R(\alpha)$ by the empirical risk $R_{emp}(\alpha)$ constructed on the basis of the training set $\mathcal{X}$. Approximate $f(x, \alpha^*)$ by $f(x, \alpha_{\mathcal{X}})$.

The ERM principle is very general. With different loss functions it is realized in the least-squares method for regression estimation or in the maximum-likelihood method for density estimation. The learning process is called *consistent* if both $R(\alpha_{\mathcal{X}})$ and $R_{emp}(\alpha_{\mathcal{X}})$ converge to the minimal possible risk $\inf_{\alpha \in \Lambda} R(\alpha)$ for $l \to \infty$.

1.2.2 Structural Risk Minimization (SRM)

Does a low number of errors on the training data imply that a machine is good at classifying new inputs that did not belong to the training set? For a trained machine $f(x, \alpha^*)$ with $R_{emp}(\alpha^*) = 0$ consider the classifier

$$f'(x, \alpha^*) = \begin{cases} f(x, \alpha^*) & \text{for} \quad x \in \mathcal{X} \\ -f(x, \alpha^*) & \text{for} \quad x \notin \mathcal{X} \end{cases}$$

f' displays the same performance on the training set as f, but they behave contrary on new data. As it seems, we have somehow to limit the choice of functions in the learning machine.

A similiar problem is usually described as *overfitting*. If the learning machine is a rich class of highly flexible functions it is said to have a high *capacity*. Machines with too much capacity are able to learn any set of training data without error, but they may perform poorly if faced with new inputs. These machines are too much adapted on the details of the training set to recognize common features in new data points. Hence, the best generalization performance is a trade-off between the accuracy attained on the training set and the capacity of the machine.

This idea can be formalized by bounds on the risk which we will study in greater detail in chapter 7 *Generalization Theory*: With probability $1 - \eta$ $(0 \leq \eta \leq 1)$ the following bound holds for all α:

$$R(\alpha) \;\leq\; R_{emp}(\alpha) + \sqrt{\frac{1}{l}\left(h\left(1 + \ln\frac{2l}{h}\right) + \ln\frac{4}{\eta}\right) + \frac{1}{l}}, \tag{1.3}$$

where h is a measure for the capacity called Vapnik-Chervonenkis (VC) dimension. It will be defined in the next chapter.

It is usually not possible to compute the left-hand side of this equation, but if we know the VC dimension we can easily compute the right-hand side to gain an upper bound on the risk. The second summand on the right-hand side increases monotonic in h and is usually called *VC confidence*.

The bound shows that low risk depends both on the chosen class of functions (the learning machine) and on the the particular function chosen by the learning algorithm. The bound decreases if we achieve a good separation on the training set by a learning machine with low VC dimension. This is the essential idea of structural risk minimization.

Definition 1.2 (Structural Risk Minimization) Let the entire class of functions $S = \{f(x, \alpha) : \; \alpha \in \Lambda\}$, be divided into nested subsets of functions $S_k = \{f(x, \alpha) : \; \alpha \in \Lambda_k\}$, such that

$$S_1 \subset S_2 \subset \ldots \subset S_n \ldots,$$

For each subset S_i, we must be able either to compute h_i or to get a bound on h_i itself, then

$$h_1 \leq h_2 \leq \ldots \leq h_n \leq \ldots$$

For a given training set the SRM principle chooses the subset S_k which minimizes the bound on the risk functional (1.1).

The SRM principle chooses the subset S_k for which minimizing the empirical risk yields the best bound on the actual risk. This can be done by training one machine in each of the subsets with the goal of minimizing the empirical risk. From this sequence of trained machines the one is taken whose sum of empirical risk and VC confidence is minimal.

In this approach we keep the confidence interval fixed (by choosing a particular h_k) and minimize the empirical risk. This is implemented in Neural Networks by first choosing an appropriate architecture and then eliminating classification errors. The architecture determines the flexibility of the Neural Network and thus the VC dimension. Hence, choosing an architecture results in fixing the confidence term in (1.3). A second approach is to keep the empirical risk fixed (say equal to zero) and minimize the confidence interval. We will discuss the implementation of this idea by Support Vector Machines in chapter 7.

1.3 Other Learning Models

In supervised learning theory we are provided with 'true' values by a trusted source. What happens if we do not have examples of output values? This is the realm of *unsupervised learning*. The learning task is to gain insight in the structure of the data and to understand the process that generated it. The methods used in this field include density estimation and clustering.

In the model we will study, the learner has no influence on the training data given to him. One can consider more complex interactions between the learner and the environment, e.g. allow the learner to query to environment about the output associated to a given input or even supply the learner with a range of actions to influence the environment. The first model is called *query learning* and the second *reinforcement learning*.

Chapter 2

Learning Terminology

In the field of machine learning there is a broad range of notions and concepts which are part of the basic terminology but usually lack an explicit introduction. We spend this chapter on giving a description of the ones most important for the further theory.

First we will describe what is understood by the term *generalization ability*, then we explain what is meant by *capacity* and give a first example of a measure for the capacity of a function class, the *VC dimension*. In chapter 7 we will introduce two further concepts, the *entropy* and the *growth function*. We end this chapter with a discussion of the *curse of dimensionality* and three examples which show that the number of parameters (the dimensionality) and the VC dimension (the ability to separate points) can be independent.

2.1 Generalization Ability

As we have seen in the last chapter, our task is to infer general features from a set of labeled examples. We have to generalize from the training examples to the whole range of possible observations. Our success is measured by our ability to correctly classify new data that did not belong to the training set. This is called the *Generalization ability*.

Mathematically this is expressed by the risk functional, the expected difference of the outputs of the learning machine to the true values. Since the risk cannot be computed directly, it is assessed by bounds which we will state in chapter 7 *Generalization Theory*.

What can be computed is the empirical risk on the training set. To describe the generalization ability we have to answer the questions, what actual risk is provided by the function minimizing the empirical risk and how close is this risk to the minimal possible one for the given set of functions (Vapnik, 1995, p72).

A direct way to describe generalization ability is by testing the trained machine on a set of known samples which is different from the training set and sum up all differences. This gives an estimate of the overall generalization performance of the learning machine.

2.2 Capacity

With the term 'capacity' we describe the ability of a machine to learn any training set without error. It is a measure of the richness or flexibility of the function class.

A machine with too much capacity tends to overfitting, whereas low capacity leads to errors on the training set. Burges illustrates that with an impressive example: "A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that if it's green, it's a tree. Neither can generalize well" (Burges, 1998, p2).

The most popular concept to describe the richness of a function class in machine learning is the Vapnik-Chervonenkis dimension. It is related to the entropy of a set of functions (Vapnik, 1998, p145).

2.3 VC Dimension

We introduce the Vapnik-Chervonenkis dimension in the case of indicator functions, i.e. functions with only two possible outcomes.

Definition 2.1 (Shattering) Given a set of indicator functions $F = \{f(\alpha)\}$, $\alpha \in \Lambda$. A set of l points is shattered by F if the set can be labeled in all possible 2^l ways, and for each labeling, a member of F can be found which correctly assigns those labels.

Definition 2.2 (VC Dimension) The Vapnik-Chervonenkis Dimension (VC dimension) of a set of indicator functions is defined as the maximum number of training points that can be shattered by it. The VC dimension is infinite, if l points can be shattered by the set of functions no matter how large l is.

If the VC dimension is h, then there exists at least one set of h points that can be shattered. The definition does not imply that this holds for *every* set of h points.

Example: We consider *oriented hyperplanes*, i.e. the pair (H, w), where H is the set of points which lie in the hyperplane and w is a particular choice of the unit vector. So (H, w) and $(H, -w)$ are different oriented hyperplanes. The VC dimension of oriented hyperplanes in $\mathbb{R}^2$ is equal to $h = 3$, since it is possible to find 3 points that can be shattered, but not 4. With 4 points one can always assign two points to class I and the other points to class II such that they cannot be separated by a line in the plane. The figures 2.1 and 2.2 illustrate that.

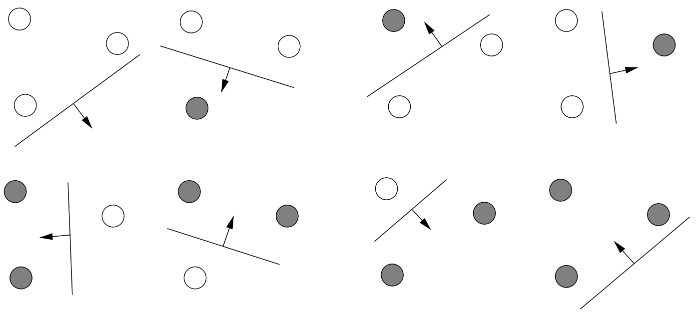

Figure 2.1: Three points not lying in one line can be shattered by oriented hyperplanes in $\mathbb{R}^2$. The arrow points in the direction of the positive examples (grey).

Figure 2.2: No four points can be found in $\mathbb{R}^2$ which can be shattered, because in each possible position the above labeling prevents separation.

Theorem 2.3 *Consider some set of m points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the m points can be shattered by oriented hyperplanes iff the position vectors of the remaining points are linearly independent.*

Proof: (Burges, 1998, p37) □

From this theorem follows, that the VC dimension of oriented hyperplanes in $\mathbb{R}^n$ is equal to $h = n+1$: For all $n+1$ points we can choose one as origin and the remaining n points are linearly independent, while we can never choose $n+2$ such points, because no $n+1$ points in $\mathbb{R}^n$ can be linearly independent.

2.4 Curse of Dimensionality

An extremely high dimension of the space we do classification in can make the computational and generalization performance degrade. This is unfortunate, since the more features we use to describe our data the more likely that the target function can be represented using the learning machine. The situation seems to be cursed: performance decreases with increasing information content.

In chapter 7 we will see that the VC dimension and not the number of parameters is responsible for the generalization ability of learning machines. So we can hope for a good generalization on basis of functions containing a huge number of parameters if the VC dimension of the set is small.

Intuitively, one could think that an increasing number of parameters leads to an higher VC dimension, while learning machines with few parameters have low VC dimension. But examples can be shown that the VC dimension is not controlled by the number of parameters: (Vapnik, 1995, pp 76–78) and (Vapnik, 1998, pp 155–160).

Example 1. (VCdim $\equiv$ #parameters) From Theorem 2.3 follows that the VC dimension of the set of *linear indicator functions* $f(x, (\alpha, \alpha_0)) = sign(\langle \alpha, x \rangle + \alpha_0)$ in n-dimensional space is equal to $h = n + 1$. In this case VC dimension indeed coincides with the number of free parameters $\alpha_0, \alpha_1, \dots, \alpha_n$. In general this is not true, as can be seen by the following striking example. It shows, that a learning machine with just one parameter is able to shatter any number of given points.

Example 2. (VCdim $>$ #parameters) Consider the set of indicator functions $f(x, \alpha) = sign(\sin(\alpha x))$, $x, \alpha \in \mathbb{R}$. For any given number l the points $x_i = 10^{-i}$, $i = 1, \dots, l$ with the sequence of labels $y_1, \dots, y_l$, $y_i \in \{-1, +1\}$ can be shattered by choosing the value of the parameter α to be

$$\alpha = \pi \left(1 + \sum_{i=1}^{l} \frac{(1 - y_i) 10^i}{2} \right).$$

Thus the VC dimension of this machine is infinite, and it is shown, that VC dimension can exceed the number of parameters.

Example 3. (VCdim $<$ #parameters) Consider a set of one-dimensional nondecreasing indicator functions

$$f(x, \alpha) = sign\left(\sum_{i=1}^{n} |\alpha_i x^i| \ sign(x) \ + \alpha_0 \right),$$

where $x \in \mathbb{R}^1$, $\alpha \equiv (\alpha_0, \dots, \alpha_n) \in \mathbb{R}^{n+1}$. Using this set on the line, one can shatter only one point. Hence, the VC dimension is independent of the number of parameters $n + 1$.

The VC dimension of Support Vector Machines will be discussed in chapter 7. Before we introduce SVM in the next part of this study we give an overview of the Lagrange theory of constrained optimization which will become the basic tool in support vector theory.

Chapter 3

Optimization Theory

As we have seen in the first chapter, the learning task may be formulated as an optimization problem. The hypothesis function should be chosen to minimize a certain functional, the risk function. Typically the optimization will be subject to some constraints.

In support vector theory we are only concerned with the case in which the cost function is a convex quadratic function, while the constraints are linear. The methods to solve these problems are called *convex quadratic programming.*

In this chapter we will consider Lagrangian theory, that will provide us with necessary and sufficient conditions for a given function to be a solution. Furthermore, we will introduce the concept of duality, which will play a major role in the construction of support vector machines. Lagrangian theory was developed in 1797 and initially only dealed with equality constraints. In 1951 the theory was extended by Kuhn and Tucker to the case of inequality constraints. This generalization is now referred to as Kuhn-Tucker theory.

3.1 Problem Formulation

The general optimization problem can be stated as a minimization problem, since reversing the sign of the function that is being optimized converts the problem to a maximization one.

Definition 3.1 (Primal Optimization Problem) Given functions f, g_i, $i = 1, \ldots, k$, and h_j, $j = 1, \ldots, m$, defined on a domain $\Omega \subseteq \mathbb{R}^n$,

$$\begin{array}{lll} \textit{minimize} & f(w) & w \in \Omega \\ \textit{subject to} & g_i(w) \leq 0 & i = 1, \ldots, k \\ & h_j(w) = 0 & j = 1, \ldots, m \end{array} \tag{3.1}$$

where $f(w)$ is called the objective function, g_i the inequality and h_i the equality constraints. To simplify our notation we will write $g(w) \leq 0$ and $h(w) = 0$ to indicate, that the constraints hold for all i. The optimal value of f is called the *value of the optimization problem.*

We will call $F = \{w \in \Omega \ : \ g(w) \leq 0\,,\, h(w) = 0\}$ the *feasible region.* It is the region of the domain, where f is defined and all constraints are satisfied. A solution $w^* \in F$ of the optimization problem is called a *global minimum* of f in F iff no $w \in F$ exists such that $f(w) < f(w^*)$. A point w^* is called a *local minimum* of $f(w)$ if there exists $\epsilon > 0$ such that $f(w) \geq f(w^*)$ for all $w \in \Omega$ with $\|w - w^*\| < \epsilon$.

An optimization problem is called a *linear programme* if the objective function and all the constraints are linear, and a *quadratic programme* if the objective function is quadratic while the constraints are all linear.

An important concept is that of an *active constraint.* A constraint g_i is called active at w, if $g_i(w) = 0$, so that any constraint is active at w if w is at the boundary of its feasible region. In this sense, equality constraints are always active. In particular the constraints active at the optimal solution w^* are of interest. If $\mathcal{A}(w^*) = \{i : \ g_i(w^*) = 0\}$ is known, then the remaining constraints can be ignored locally.

We will restrict our study to the case of convex quadratic programmes, because in support vector theory only this class of problems will be needed. First we define convexity of functions and of sets.

Definition 3.2 (Convexity) A set $\Omega \subseteq \mathbb{R}^n$ is called convex if $\forall w, u \in \Omega$, and for any $\theta \in (0,1)$ the point $(\theta w + (1-\theta)u) \in \Omega$. A real-valued function $f(w)$ is called convex for $w \in \mathbb{R}^n$ if, $\forall w, u \in \mathbb{R}^n$, and for any $\theta \in (0,1)$ the relation holds $f(\theta w + (1-\theta)u) \leq \theta f(w) + (1-\theta) f(u)$. If strict inequality holds, the function is said to be strictly convex.

Essentially a convex function is one for which the epigraph is a convex set. For twice differentiable functions a criterion for convexity can be given by the positive semi-definiteness of the Hessian matrix.

A function that can be expressed in the form $f(w) = Aw + b$ for some matrix A and vector b is called *affin.* Affin functions are all convex, because their Hessian equals zero.

The problem of minimizing a convex function on a convex set is said to be a *convex programming problem.* What makes these kind of problems attractive is that every local solution to a convex problem is a global solution and that any global solution is unique, if $f(w)$ is also strictly convex. This is proofed in every good textbook, for example (Fletcher, 1987, p217).

To develop support vector theory we can restrict ourselves to the case of linear constraints, a convex and quadratic objective function and $\Omega \subset \mathbb{R}^n$. Hence, the next sections will be restricted to the case of convex quadratic programmes.

3.2 Lagrangian Theory

We begin with the most simple case. In absence of constraints the solution can sufficiently be characterized by the stationarity of the objective function.

Theorem 3.3 (Fermat) *A necessary condition for w^* to be a minimum of $f(w)$, $f \in C^1$, is $\frac{\partial}{\partial w} f(w^*) = 0$. This also is a sufficient condition for convex functions.*

To use this in constrained problems, one defines a function, known as the Lagrangian, that unites information about both the objective function and the constraints, and whose stationarity can be used to detect solutions.

Definition 3.4 (Lagrangian) The Lagrangian function is defined as the objective function $f(w)$ plus a linear combination of equality constraints $h_i(w),\ i = 1, \dots, m$:

$$L(w, \alpha) = f(w) + \sum_{i=1}^{m} \alpha_i h_i(w)$$

where the coefficients α_i are called the *Lagrange multipliers*.

Theorem 3.5 (Lagrange) *Given an optimization problem with an objective function $f(w)$ and equality constraints $h_i(w),\ i = 1, \dots, m$ and $f, h_i \in C^1$. Then a necessary condition for $w^* \in \mathbb{R}^n$ to be a solution is*

$$\begin{aligned} \frac{\partial}{\partial w} L(w^*, \alpha^*) &= 0 \quad \textit{and} \\ \frac{\partial}{\partial \alpha} L(w^*, \alpha^*) &= 0 \end{aligned}$$

These conditions are also sufficient in the case that $L(w^*, \alpha^*)$ is a convex function of w.

The conditions give a linear system of $n+m$ equations, where the last m are the equality constraints. By solving this system one obtains the solution. Note that at the optimal point the constraints equal zero, so the value of the Lagrangian is equal to the value of the objective function:

$$L(w^*, \alpha^*) = f(w^*)$$

We now include inequality constraints to the problem. This compels to give a definition of the generalized Lagrangian:

Definition 3.6 (Generalized Lagrangian Function) Consider the general optimization problem

$$\begin{array}{ll} \textit{minimize} & f(w) \\ \textit{subject to} & g_i(w) \leq 0, \quad i = 1, \dots, k, \\ & h_j(w) = 0, \quad j = 1, \dots, m. \end{array}$$

We define the generalized Lagrangian as

$$\begin{aligned} L(w, \alpha, \beta) &= f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{j=1}^{m} \beta_i h_j(w) \\ &= f(w) + \alpha^t g(w) + \beta^t h(w). \end{aligned}$$

We will now define the Lagrangian dual problem. Duality will be a central element of support vector theory.

Definition 3.7 (Lagrangian Dual Problem) Given the primal problem of definition 3.1, we define the dual problem to be

$$\begin{array}{ll} \textit{maximize} & \theta(\alpha, \beta) = \inf_{w} L(w, \alpha, \beta) \\ \textit{subject to} & \alpha > 0\ . \end{array}$$

The maximal value of the objective function will be called the value of the problem.

Theorem 3.8 (Weak Duality Theorem) *For a feasible solution w of the primal problem and a feasible solution (α, β) of the dual problem, the relation holds*

$$f(w) \geq \theta(\alpha, \beta)$$

Proof:

$$\begin{aligned} \theta(\alpha, \beta) &= \inf_{v} L(v, \alpha, \beta) \\ &\leq L(w, \alpha, \beta) \\ &= f(w) + \alpha^t g(w) + \beta^t h(w) \\ &\leq f(w)\ . \end{aligned}$$

The last inequality follows directly from feasibility of w and (α, β), since $g(w) \leq 0$, $h(w) = 0$ and $\alpha \geq 0$. $\square$

Duality gap. As a direct consequence of this theorem we get, that the value of the dual is upper bounded by the value of the primal. The difference of the two values is known as the *duality gap*. If $f(w^*) = \theta(\alpha^*, \beta^*)$ while the constraints hold, then w^* and

(α^*, β^*) are solutions of the primal and dual problem respectively. Since the values are equal, the sequence of inequalities in the proof of the last theorem become equalities and it follows that in this case $\alpha_i^* g_i(w^*) = 0 \ \ \forall i$. So, if we compare the value of the primal to the value of the dual problem we are able to test for optimality. If the duality gap reduces to zero, we have found an optimal solution.

Saddle point. Without monitoring both values at once we can detect the absence of the duality gap by the presence of a *saddle point*. A saddle point is defined as a triple (w^*, α^*, β^*) with $w^* \in \Omega$, $\alpha^* \geq 0$ satisfying the property

$$L(w^*, \alpha, \beta) \ \leq \ L(w^*, \alpha^*, \beta^*) \ \leq \ L(w, \alpha^*, \beta^*),$$

for all $w \in \Omega$, $\alpha \geq 0$. The saddle point is a minimum with respect to w and a maximum with respect to (α, β).

Theorem 3.9 *If the triple (w^*, α^*, β^*) is a saddle point of the primal Lagrangian function, then its components are optimal solutions of the primal and dual problem and there is no duality gap: $f(w^*) = \theta(\alpha^*, \beta^*)$*

Proof: (Walsh, 1975, p20) □

The next theorem guarantees that the dual and primal problems have the same value for the optimisation problems we will have to solve in support vector theory.

Theorem 3.10 (Strong Duality Theorem) *Given the primal optimization problem 3.1, where the domain Ω is convex and the constraints g_i and h_i are affine functions. Then the duality gap is zero.*

Proof: (Fletcher, 1987, p219) □

3.3 Kuhn-Tucker Theory

Theorem 3.11 (Kuhn-Tucker) *Given the primal optimization problem (3.1) with $f \in C^1$ convex and g_i, h_i affine. Necessary and sufficient conditions for w^* to be an optimum are the existence of α^*, β^* such that*

$$\begin{aligned}
\frac{\partial}{\partial w} L(w^*, \alpha^*, \beta^*) &= 0; \\
\frac{\partial}{\partial \beta} L(w^*, \alpha^*, \beta^*) &= 0; \\
\alpha_i^* g_i(w^*) &= 0 \quad i = 1, \ldots, k; \\
g_i(w^*) &\leq 0 \quad i = 1, \ldots, k; \\
\alpha_i^* &\geq 0 \quad i = 1, \ldots, k.
\end{aligned}$$

Proof: (FLETCHER, 1987, p218), where we have to use that affine functions are in C^1 and that the following regularity assumption holds: The intersection of the set of feasible directions with the set of descent directions coincides with the intersection of the set of feasible directions *for linearized constraints* with the set of descent directions. This holds for all support vector machines, since the constraints are always linear. $\square$

The relation $\alpha_i^* g_i(w^*) = 0$ is referred to as the Kuhn-Tucker *complementarity condition.* It states that both α_i^* and $g_i(w^*)$ cannot be non-zero, or equivalently that inactive constraints have zero multiplier. It is easy to see, why this condition holds. To solve the optimization problem, we have to minimize the primal Lagrangian, where $\alpha_i g_i$ occurs as a positive summand. If $g_i > 0$, the value of the whole sum is reduced by setting the multiplier α_i to zero. On the other hand, if $g_i = 0$ the constraint remains $\alpha_i \geq 0$. If there is no i such that $\alpha_i^* = g_i(w^*) = 0$ then *strict complementarity* is said to hold.

The complementarity condition allows for the following interpretation: A solution point can be in one of two positions with respect to an inequality constraint, either in the interior of the feasible region with the constraint inactive, or on the boundary defined by the constraint with the constraint active.

For active constraints the Lagrange multiplier represents the sensitivity of the optimal value to the constraint. This can be seen by replacing the i-th constraint by $h_i(w) = b_i$ and considering the value of the objective function at the optimal solution $f^* = f(w^*)$ as a function of b_i. It follows, that

$$\left[\frac{\partial f^*}{\partial b_i}\right]_{b_i=0} = \alpha_i^*$$

Perturbing inactive constraints has no effect on the solution. The case $\alpha_i^* = g_i(w^*) = 0$ is an intermediate state between a strongly active constraint and one being inactive.

3.4 Duality

The introduction of dual variables is a powerful tool, because the alternative dual description of an optimization problem often turns out to be easier to solve than the primal problem. The dual function does not depend on the primal variables and must be maximised under simpler constraints. The dual variables are considered to be the fundamental unknowns of the problem, so dual methods give new insights in the structure of the optimization task.

How do we gain the dual description? The transformation is performed in two steps: First, set to zero the derivatives of the primal Lagrangian with respect to the primal variables, and then substitute these relations back into the Lagrangian. This removes the dependence on the primal variables and corresponds to explicitly computing $\theta(\alpha, \beta) = \inf_w L(w, \alpha, \beta)$.

This strategy is a standard technique in the theory of Support Vector Machines. Many advantages can be derived from it. It provides us with algorithmic methods derived from optimization theory. The dual representation will allow to work in high dimensional spaces by using kernel methods without falling prey to the curse of dimensionality. The KT complementarity conditions lead to an reduction of the amount of data involved in the learning task. Only active constraints will have non-zero dual variables, so the number of variables involved may be significantly fewer than the full training set size. In the next chapter we will see how the training examples for which the dual variables are non-zero determine the classifying hyperplane. This leads to the term *support vectors* after which the whole theory is named.

Part II

Support Vector Machines

Chapter 4

Linear Classifiers

As a first step in the construction of Support Vector Machines we study classification by simple linear hyperplanes. Consider the class of hyperplanes $\langle w, x\rangle + b = 0, w \in \mathbb{R}^n, b \in \mathbb{R}$, corresponding to decision functions

$$f(x) = sign(\langle w, x\rangle + b) \tag{4.1}$$

We will first consider the case of linear separable data. There are many possible ways in which a hyperplane can separate the two classes. We will need a criterion to choose 'the best one', the 'optimal' separating hyperplane.

The idea of learning from examples is to gain an impression of what the elements of a class look like by examining the training points that belong to that class. All new data points are thought to lie somewhere near the known training data. Hence, the hyperplane should be chosen such, that small shifts of the data do not result in changing predictions. If the distance between the separating hyperplane and the training points becomes too small, even test examples very close to the training samples may be classified incorrectly. The figure 4.1 illustrates that.

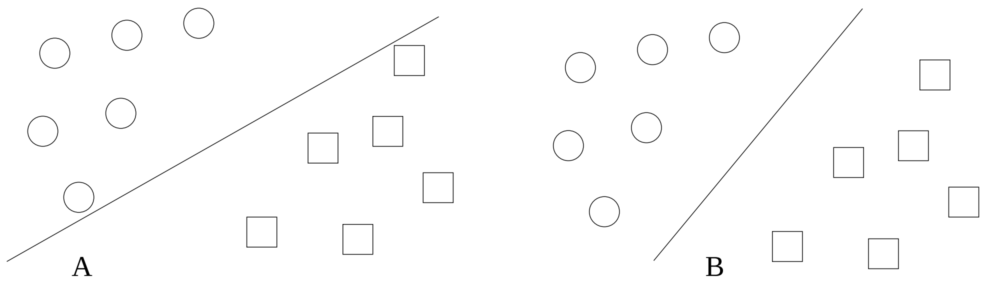

Figure 4.1: Which separation is the better? Almost zero margin of separation in (A) compared to large margin in (B).

It seems that correctly classifying unseen data will be more successfull in setting B than in setting A. This observation leads to the concept of maximal margin hyperplanes.

The approach of VAPNIK and LERNER (1963) and VAPNIK and CHERVONENKIS (1964) stems from the intuition that the generalisation ability depends on the distance of the hyperplane to the training points. They propose a learning algorithm for separable problems, called the *Generalized Portrait*, that constructs a hyperplane yielding the maximum margin of separation between the classes:

$$\max_{w,b} \quad \min\{\|x - x_i\| : x \in \mathbb{R}^n, \langle w, x\rangle + b = 0, i = 1, \ldots, l\} \tag{4.2}$$

In this chapter we will describe an effective way to construct this hyperplane. We begin with separating hyperplanes constructed on linear separable datasets and will generalize the results to allow for errors on the training set.

4.1 The Optimal Separating Hyperplane (OSH)

Definition 4.1 (Separability) The training set $\mathcal{X} = \{(x_1, y_1), \ldots, (x_l, y_l) : x_i \in \mathbb{R}^n, y_i \in \{-1, +1\}\}$ is called separable by the hyperplane $\langle w, x\rangle + b = 0$ if there exist both a unit vector w $(\|w\| = 1)$ and a constant b such that the equalities hold true:

$$\langle w, x_i\rangle + b > 0 \qquad \text{for} \quad y_i = +1 \tag{4.3}$$
$$\langle w, x_i\rangle + b < 0 \qquad \text{for} \quad y_i = -1 \tag{4.4}$$

The hyperplane defined by w and b is called a separating hyperplane.

Definition 4.2 (Margin) Consider the separating hyperplane H defined by $\langle w, x\rangle + b = 0$.

- The margin $\gamma_i(w, b)$ of a training point x_i is defined as the distance between H and x_i:
$$\gamma_i(w, b) = y_i(\langle w, x_i\rangle + b)$$
- The margin $\gamma_{\mathcal{S}}(w, b)$ of a set of vectors $\mathcal{S} = \{x_1, \ldots, x_n\}$ is defined as the minimum distance from H to the vectors in $\mathcal{S}$:
$$\gamma_{\mathcal{S}}(w, b) = \min_{x_i \in \mathcal{S}} \gamma_i(w, b)$$

Now consider the unit vector w^* and the constant b^* which maximise the margin of the training set $\gamma(w, b)$ under the condition that inequalities (4.3) and (4.4) are satisfied.

The pair (w^*, b^*) determines the hyperplane that separates the positive from the negative examples and has maximal margin. We call this hyperplane the *Maximal Margin Hyperplane* or the *Optimal Separating Hyperplane*.

Definition 4.3 (Optimal Separating Hyperplane) The Optimal Separating Hyperplane (OSH) of a training set $\mathcal{X}$ is defined by

$$(w^*, b^*) \quad = \quad \operatorname*{argmax}_{w,b} \ \gamma_{\mathcal{X}}(w, b)$$

Theorem 4.4 (Uniqueness of OSH) *The Optimal Separating Hyperplane is unique.*

Proof: This is a new formulation of Vapnik's proof in (Vapnik, 1998, p402). By definition we get:

$$\begin{aligned} (w^*, b^*) &= \operatorname*{argmax}_{w,b} \ \gamma_{\mathcal{X}}(w, b) \\ &= \operatorname*{argmax}_{w,b} \ \min_{x_i \in \mathcal{X}} \gamma_i(w, b) \\ &= \operatorname*{argmax}_{w,b} \ \min_{x_i \in \mathcal{X}} y_i(\langle w, x_i \rangle + b) \end{aligned}$$

We first consider the maximum point w^* of the continuous function

$$\gamma(w) = \min_{x_i \in \mathcal{X}} \ y_i \langle w, x_i \rangle$$

defined in the area $\|w\| \leq 1$.

(i) *Existence* of the maximum follows from the continuity of γ in the bounded region $\|w\| \leq 1$.

(ii) Suppose that the maximum is achieved at some *interior point* w', $\|w'\| < 1$. Then the vector $w'' = w'/\|w'\|$ would define a larger margin

$$\gamma(w'') \ = \ \frac{\gamma(w')}{\|w'\|} \ > \ \gamma(w'),$$

which is a contradiction. This shows that the maximum can only be achieved on the boundary $\|w\| = 1$.

(iii) Since $\gamma(w)$ is a convex function, the maximum can not be achieved on *two* boundary points. Otherwise, it would also be achieved on the line that connects these two points. But a maximum at an inner point is impossible by the preceeding argument.

With a unique w^* we can describe b^* uniquely by

$$b^* = \frac{1}{2} \left(\min_{i \in I_+} \langle w^*, x_i \rangle - \max_{i \in I_-} \langle w^*, x_i \rangle \right),$$

where $I_+ = \{i \mid y_i = +1\}$ and $I_- = \{i \mid y_i = -1\}$. $\square$

4.1.1 The Construction of the OSH

The OSH is the solution of the optimization problem

$$\begin{aligned} \textit{maximize} \quad & \gamma_{\mathcal{X}}(w,b) \\ \textit{subject to} \quad & \gamma_{\mathcal{X}}(w,b) > 0 \\ & \|w\|^2 = 1 \end{aligned} \tag{4.5}$$

This poses several difficulties. The objective function is neither linear nor quadratic and the constraints are nonlinear. Hence, from an algorithmical point of view this is a difficult optimization problem. But we can find an effective method for constructing the OSH by considering an equivalent statement of the problem:

$$\begin{aligned} \textit{minimize} \quad & \frac{1}{2}\|w\|^2 & & \\ \textit{subject to} \quad & \langle w, x_i\rangle + b \geq +1 & \quad \textit{for} \quad & y_i = +1 \\ & \langle w, x_i\rangle + b \leq -1 & \quad \textit{for} \quad & y_i = -1 \end{aligned} \tag{4.6}$$

Theorem 4.5 *The optimization problems (4.5) and (4.6) are equivalent: the vector* w_0 *that solves problem (4.6) is related to the vector* w^* *solving problem (4.5) by the equality*

$$w^* = \frac{w_0}{\|w_0\|} \; .$$

Proof: This is a new formulation of Vapnik's proof in (Vapnik, 1998, p403). We will show: under constraints of problem (4.6), the size of the margin $\gamma(w^*)$ equals $1/\|w_0\|$. Hence, maximizing the margin $\gamma(w)$ is equivalent to minimizing $\|w\|$, which in turn is equivalent to minimizing $\frac{1}{2}\|w\|^2$.

The minimum point w_0 of (4.6) is unique (quadratic objective function under linear constraints). Then define the unitvector $w^* = \frac{w_0}{\|w_0\|}$. Since the constraints in (4.6) are valid, the equations hold

$$\begin{aligned} \frac{1}{\|w_0\|}\langle w_0, x_i\rangle + \frac{b}{\|w_0\|} \quad & \geq \quad \frac{1}{\|w_0\|} \qquad \forall i \in I_+ \\ -\frac{1}{\|w_0\|}\langle w_0, x_j\rangle - \frac{b}{\|w_0\|} \quad & \geq \quad \frac{1}{\|w_0\|} \qquad \forall j \in I_- \; . \end{aligned}$$

This leads to

$$\langle \frac{w_0}{\|w_0\|}, x_i\rangle - \langle \frac{w_0}{\|w_0\|}, x_j\rangle \;\geq\; \frac{2}{\|w_0\|} \; .$$

Because this holds for all $i \in I_+$ and $j \in I_-$, it holds for the minimum, too.

$$\min_{\substack{i \in I_+ \\ j \in I_-}} \left[\langle \frac{w_0}{\|w_0\|}, x_i\rangle - \langle \frac{w_0}{\|w_0\|}, x_j\rangle \right] \;\geq\; \frac{2}{\|w_0\|}$$

By definition (4.2), we get

$$\gamma\left(\frac{w_0}{\|w_0\|}\right) \;\geq\; \frac{1}{\|w_0\|}\;.$$

To prove the theorem it is sufficient to show that the relation '>' is impossible. This will be done by contradiction.

Suppose '>' holds true. We define the unit vector $w' := \frac{w_0}{\|w_0\|}$, then $\gamma(w') > 1/\|w_0\|$. Consider the vector $w'' = w'/\gamma(w')$. By assumption it has norm smaller than $\|w_0\|$:

$$\|w''\| \;=\; \frac{1}{\gamma(w')}\|w'\| \;<\; \|w_0\|\;.$$

We will now show, that w'' satisfies the constraints in (4.6). For $y_i = +1$ this follows from the chain of equations

$$\begin{aligned}\langle w'', x_i\rangle + b \;&=\; \frac{1}{\gamma(w')}\langle w', x_i\rangle + b\\ &=\; \frac{1}{\gamma(w')}\frac{1}{\|w_0\|}\langle w_0, x_i\rangle + b\\ &>\; \langle w_0, x_i\rangle + b \;\;>\; 1\end{aligned}$$

(Analogously for $y_i = -1$.) This contradicts the assertion that w_0 is the smallest vector satisfying these constraints. Thus we have shown that $\gamma(\frac{w_0}{\|w_0\|}) = \frac{1}{\|w_0\|}$, and the theorem is proofed. □

Theorem 4.5 implies: to construct the OSH we have to solve problem (4.6), which is a quadratic optimization problem with linear constraints. To simplify our notation let us rewrite the constraints of (4.6) in the equivalent form

$$y_i(\langle w, x_i\rangle + b) - 1 \;\geq\; 0 \qquad i = 1, \ldots, l. \tag{4.7}$$

We will solve the optimization problem by the Lagrange method, i.e. we try to find the saddle point of the Lagrangian

$$L_P(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l}\alpha_i[y_i(\langle w, x_i\rangle + b) - 1], \tag{4.8}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. The Lagrangian $L_P(w, b, \alpha)$ has to be minimized with respect to the primal variables w and b and maximized with respect to the dual variables α_i.

At the saddle point the derivatives of L_P with respect to the primal variables must vanish,

$$\frac{\partial}{\partial w}L_P(w, b, \alpha) = 0 \quad \text{and} \quad \frac{\partial}{\partial b}L_P(w, b, \alpha) = 0\;.$$

This leads to

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \ . \tag{4.9}$$

By substituting (4.9) back into L_P one arrives at the Wolfe dual of the optimization problem:

$$\begin{aligned} L_D(w,b,\alpha) &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i [y_i(\langle w, x_i\rangle + b) - 1] \\ &= \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - \sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle + \sum_{i=1}^{l}\alpha_i \\ &= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle. \end{aligned} \tag{4.10}$$

The dual problem can be formulated: Find multipliers α_i which solve

$$\begin{aligned} &\textit{Maximize} \quad && L_D(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle, \\ &\textit{subject to} && \alpha_i \geq 0 \qquad i = 1,\ldots,l, \\ &&& \sum_{i=1}^{l}\alpha_i y_i = 0. \end{aligned} \tag{4.11}$$

To construct the OSH we have to find the coefficients α_i^* that solve the dual problem. As a solution we obtain

$$w^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i \ . \tag{4.12}$$

Note that we have given the Langrangian different labels to emphasize the difference between the two formulations. 'P' stands for 'primal' and 'D' for 'dual'. L_P and L_D arise from the same objective function, but with different constraints. The solution is found by minimizing the primal L_P or by maximizing the dual L_D.

Lemma 4.6 *Let the vector w^* define the optimal hyperplane. Then the maximum of the functional $L_D(\alpha)$ is equal to*

$$L_D(\alpha^*) \ = \ \frac{1}{2}\|w^*\|^2 \ = \ \frac{1}{2}\sum_{i=1}^{l}\alpha_i^* \ .$$

Proof: The optimal solution must satisfy the Kuhn-Tucker conditions of theorem 3.11 at page 26. This implies:

$$\alpha_i^*[y_i(\langle w^*, x_i\rangle + b^*) - 1] \ = \ 0 \quad \forall i$$

Summing over all α_i we get

$$\sum_{i=1}^{l} \alpha_i^* y_i \langle w^*, x_i \rangle + b^* \sum_{i=1}^{l} \alpha_i^* y_i - \sum_{i=1}^{l} \alpha_i^* \; = \; 0 \; .$$

This leads to

$$\sum_{i=1}^{l} \alpha_i^* \; = \; \sum_{i=1}^{l} \alpha_i^* y_i \langle w^*, x_i \rangle \; = \; \|w^*\|^2, \tag{4.13}$$

because $\sum \alpha_i^* y_i = 0$ and $w^* = \sum \alpha_i^* y_i x_i$ by (4.9). This proofs the second equality of the lemma. Also by (4.9) we can write

$$\|w^*\|^2 \; = \; \sum_{i,j} \alpha_i^* \alpha_j^* y_i y_j \langle x_i, x_j \rangle \; . \tag{4.14}$$

Substituting (4.13) and (4.14) into (4.11) we get

$$\begin{aligned} L_D(\alpha^*) \; &= \; \sum_{i=1}^{l} \alpha_i^* - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i^* \alpha_j^* y_i y_j \langle x_i, x_j \rangle \\ &= \; \|w^*\|^2 - \frac{1}{2}\|w^*\|^2 \; = \; \frac{1}{2}\|w^*\|^2 \; . \end{aligned}$$

□

Lemma 4.7 *The margin $\gamma(w^*)$ of the OSH can be computed from the solution coefficients α_i^* as*

$$\gamma(w^*)^2 \; = \; \left(\sum_{i=1}^{l} \alpha_i^* \right)^{-1} \; .$$

Proof: Use lemma 4.6 in connection with $\gamma(w^*) = 1/\|w^*\|$ which is shown in the proof of theorem 4.5. □

Our result in this section is: the construction of the Optimal Separating Hyperplane amounts to maximizing L_D with respect to the α_i, subject to (4.7) and positivity of the α_i, with solution given by (4.12). An important observation is that the training points enter the algorithm only as entries in an inner product. So the influence of the training set can be brought together in the so called Gram matrix $G = (\langle x_i, x_j \rangle)$. This will play a central role in the next chapter, where we will generalize the concept of separating hyperplanes to nonlinear decision surfaces.

4.1.2 Support Vectors

For the primal problem above, the KT conditions may be stated:

$$
\begin{aligned}
\frac{\partial}{\partial w_\nu} L_P = w_\nu - \sum_i \alpha_i y_i x_{i\nu} &= 0 \qquad \nu = 1, \ldots, n \\
\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i &= 0 \\
y_i(\langle w, x_i \rangle + b) - 1 &\geq 0 \qquad i = 1, \ldots, l \\
\alpha_i &\geq 0 \qquad \forall i \\
\alpha_i [y_i(\langle w, x_i \rangle + b) - 1] &= 0 \qquad \forall i
\end{aligned} \tag{4.15}
$$

The regularity assumption of Kuhn-Tucker theory holds for all Support Vector Machines, since the constraints are always linear. The problem for SVMs is convex (a convex function, with constraints giving a convex feasible region). For convex problems, the KT conditions are *necessary and sufficient* for w, b, α to be a solution. Thus solving the SVM problem is equivalent to finding a solution to the KT conditions.

Let's have a closer look at equation (4.15). It is called the KT *complementarity condition.* In our chapter on optimization theory we tried to build up an intuition on how it works. Now, what is its meaning in the SVM problem? It states, that for a given training point x_i either the corresponding Lagrange multiplier α_i equals zero, or x_i lies on one of the hyperplanes H_1 or H_2:

$$H_1: \qquad \langle w, x_i \rangle + b = +1 \tag{4.16}$$

$$H_2: \qquad \langle w, x_i \rangle + b = -1 \tag{4.17}$$

These hyperplanes will be called the *margin hyperplanes.* They contain the training points with the minimal distance to the OSH, they form the boundary of the margin. The vectors that lie on H_1 or H_2 with $\alpha_i > 0$ are called *Support Vectors* (SV).

Definition 4.8 (Support Vectors) A training point x_i is called a support vector iff the corresponding Lagrange multiplier α_i satisfies $\alpha_i > 0$.

The support vectors are the training points closest to the separating hyperplane. All other training points have $\alpha_i = 0$ and lie either on one of the margin hyperplanes (such that equality in (4.7) holds) or on the side of H_1 or H_2 such that the strict inequality in (4.7) holds. Note that the complementarity condition only states that all support vectors lie on the margin hyperplanes, but not that the SV are the only training points on these hyperplanes. There may be the case, that both α_i and $y_i(\langle w, x_i \rangle + b) - 1$ equal zero. Then, the point x_i lies on one of the margin hyperplanes without being a support vector.

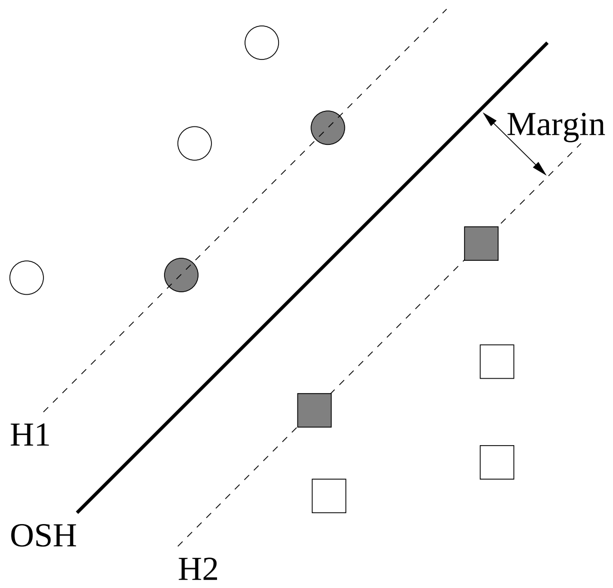

Figure 4.2: The optimal separating hyperplane (OSH) is determined by the support vectors (grey) lying on the margin hyperplanes (H_1, H_2).

The support vectors are the critical elements of the training set. If one repeats the training with all other points removed or moved around without crossing H_1 or H_2, the same separating hyperplane would be found. This results from the expansion of the normal vector w as a weighted sum of the training points in (4.9). The support vectors are the only x_i with nonzero Lagrange multiplier α_i, so we can rewrite Equation (4.9) to

$$w^* = \sum_{i=1}^{\#SV} \alpha_i y_i x_i^{SV} \quad . \tag{4.18}$$

Another application of the KT complementarity condition is the computation of the threshold b. While w is explicitly determined by the training procedure, the constant b is not. But we can easily find b by using inequality (4.7): just choose any i for which $\alpha_i \neq 0$, then (4.7) becomes an equality and we can compute b. Of course, it is numerically safer to compute b for all i and take the mean value of the results.

4.1.3 Test Phase

How do we use the separating hyperplane, once we have trained it on the training set? The hyperplane divides the $\mathbb{R}^n$ into two regions: one where $\langle w^*, x\rangle + b^* > 0$ and one where $\langle w^*, x\rangle + b^* < 0$. To use the maximal margin classifier, we determine on which side the test pattern lies and assign the corresponding class label. Hence, the predicted class of a test point x is the output of

$$\begin{aligned} f(x) &= sign(\langle w^*, x\rangle + b^*) \\ &= sign\left(\sum_{i=1}^{\#SV} \alpha_i y_i \langle x_i^{SV}, x\rangle + b^*\right). \end{aligned} \tag{4.19}$$

The optimal separating hyperplane we get by solving the margin optimization problem is a very simple special case of a Support Vector Machine, because the computation is done directly on the input data. Before we generalize the support vector concept to nonlinear classifiers and introduce the kernel mapping, we will demonstrate how a separating hyperplane can be adapted to the case of linear non-separable datasets.

4.2 OSH for Linear Non-Separable Datasets

The algorithm we worked out above cannot be used in many real-world problems. In general, noisy data will render linear separation impossible. No feasible solution to the margin maximisation problem can be found, due to the objective function (i.e. the dual Langrangian) growing arbitrarily large.

How can we extend our approach to handle non-separable data? The main drawback of the OSH is, that it allows for no classification errors. Either we get a solution without any training errors or no solution at all. To generalize the concept of the maximal margin hyperplane we relax the separability constraints (4.3) and (4.4). But each exceeding of the constraints will be punished by a misclassification penalty (i.e. an increase in the primal objective function).

This can be done by introducing positive slack variables ξ_i $(i = 1, \ldots, l)$ in the constraints, which then become:

$$\begin{aligned} \langle w, x_i\rangle + b &\geq +1 - \xi_i \quad \text{for} \quad y_i = +1 \\ \langle w, x_i\rangle + b &\leq -1 + \xi_i \quad \text{for} \quad y_i = -1 \\ \xi_i &\geq 0 \quad \forall i. \end{aligned}$$

As in equation (4.7) the first two constraints can be put into one:

$$y_i(\langle w, x_i\rangle + b) - 1 + \xi_i \geq 0 \qquad i = 1, \ldots, l. \tag{4.20}$$

We are interested in the smallest slack variable for which (4.20) is satisfied, i.e.

$$\xi_i = max\ \{0, 1 - y_i(\langle w, x_i\rangle + b)\}.$$

It measures how much a point fails to have a margin of $1/\|w\|$. The value of ξ_i indicates, where x_i lies compared to the separating hyperplane.

$$
\begin{aligned}
\xi_i \geq 1 \quad &\Longleftrightarrow \quad y_i(\langle w, x_i \rangle + b) < 0 \text{ , i.e. misclassification;} \\
0 < \xi_i < 1 \quad &\Longleftrightarrow \quad x_i \text{ is classified correctly,} \\
&\qquad\quad \textit{but} \text{ lies inside the margin;} \\
\xi_i = 0 \quad &\Longleftrightarrow \quad x_i \text{ is classified correctly,} \\
&\qquad\quad \textit{and} \text{ lies outside the margin} \\
&\qquad\quad \text{or on the margin boundary.}
\end{aligned}
$$

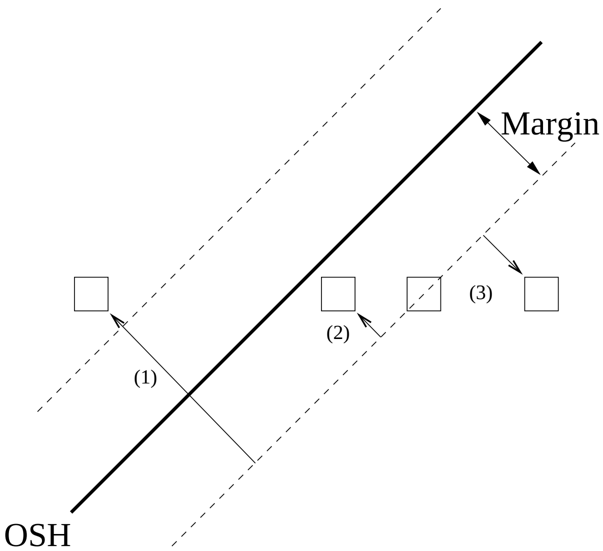

Figure 4.3: Value of slack variables: (1) misclassification if ξ_i is larger than the margin of separation, (2) correct classification of x_i in margin if $0 < \xi_i < 1$, (3) correct classification of x_i on the margin boundary or outside the margin if $\xi_i = 0$.

A classification error will be evidenced by the corresponding ξ_i exceeding unity, so $\sum_{i=1}^{l} \xi_i$ is an upper bound on the number of training errors. We want to simultaneously maximize the margin and minimize the number of misclassifications. This can be achieved by changing the objective function from $\frac{1}{2}\|w\|^2$ to $\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i^k$:

$$
\begin{aligned}
&\textit{Minimize} && \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i^k \\
&\textit{subject to} && y_i(\langle w, x_i \rangle + b) - 1 + \xi_i \;\geq\; 0 \qquad i = 1, \ldots, l \\
& && \xi_i \geq 0 \qquad i = 1, \ldots, l
\end{aligned}
\tag{4.21}
$$

The *error weight* C is a parameter to be chosen by the user, it measures the size of the penalties assigned to errors. In practice, C is varied through a wide range of values and the optimal performance is usually assessed by cross-validation. The optimization problem is convex for any positive integer k, for $k = 2$ and $k = 1$ it is also a quadratic programming problem. This approach is called the *Soft Margin Generalisation* of the OSH, while the original concept with no errors allowed is called *Hard Margin*. We will consider the soft margin case for $k = 2$ and $k = 1$.

4.2.1 1-Norm Soft Margin

For $k = 1$, the primal Lagrangian for this problem becomes

$$L_P(w, b, \xi, \alpha, \beta) =$$
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i(y_i(\langle w, x_i\rangle + b) - 1 + \xi_i) - \sum_{i=1}^{l}\beta_i\xi_i$$

with $\alpha_i \geq 0$ and $\beta_i \geq 0$. The β_i are the Lagrange multipliers introduced to enforce $\xi_i \geq 0$. Differentiation with respect to w, ξ and β yields

$$\begin{aligned}
\frac{\partial L_P}{\partial w} &= w - \sum_{i=1}^{l}\alpha_i y_i x_i &= 0 \\
\frac{\partial L_P}{\partial \xi_i} &= C - \alpha_i - \beta_i &= 0 \\
\frac{\partial L_P}{\partial b} &= \sum_{i=1}^{l}\alpha_i y_i &= 0
\end{aligned}$$

By resubstitution of these relations into the primal problem we obtain the following dual formulation:

$$\begin{aligned}
\mathit{Maximize} \quad & L_D(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j\langle x_i, x_j\rangle \qquad (4.22)\\
\mathit{subject\ to} \quad & 0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \\
& \sum_{i=1}^{l}\alpha_i y_i = 0
\end{aligned}$$

Comparing this to the hard margin optimization problem (4.11) at page 36 we find it almost identical! The only difference is that $C - \alpha_i - \beta_i = 0$ together with $\beta_i \geq 0$ enforces $\alpha_i \leq C$. So in the soft margin case, the langrange multipliers have an upper bound of C. Because of $\beta_i = \alpha_i - C$ the Kuhn-Tucker complementarity conditions derived from the primal problem are:

$$\begin{aligned}
\alpha_i[y_i(\langle w, x_i\rangle + b) - 1 + \xi_i] &= 0 \qquad \forall i \\
\xi_i(\alpha_i - C) &= 0 \qquad \forall i.
\end{aligned}$$

The KT conditions implie that non-zero slack variables can only occur for $\alpha_i = C$. The corresponding x_i has distance from the hyperplane less than $1/\|w\|$, as can be seen from the first of the two conditions above. Points for which $0 < \alpha_i < C$ lie on one of the two margin hyperplanes.

This also shows that the hard margin hyperplane can be attained in the soft margin setting by choosing the bound C to become infinite. The fact that the Lagrange multipliers are upper bounded gives rise to the name *box constraint* for this formulation, as the vector α is constrained to lie inside the box with side length C in the positive orthant.

4.2.2 2-Norm Soft Margin

We now consider the case $k = 2$. Before we state the primal Lagrangian for this problem note that for $\xi_i < 0$ the first constraint of (4.21) will still hold, while the value of the objective function is reduced by this change. Hence, we can remove the positivity constraint on ξ_i and still attain the optimal solution of the optimization problem. This leads to the primal Lagrangian

$$
\begin{aligned}
& L_P(w, b, \xi, \alpha, \beta) = \\
= \;& \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{l} \xi_i^2 - \sum_{i=1}^{l} \alpha_i (y_i(\langle w, x_i\rangle + b) - 1 + \xi_i) \\
= \;& \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i \langle w, x_i\rangle + \frac{C}{2}\sum_{i=1}^{l} \xi_i^2 - \sum_{i=1}^{l} \alpha_i \xi_i - b\sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i
\end{aligned}
$$

As before, the corresponding dual is found by differentiating with respect to w, ξ and β, imposing stationarity,

$$
\frac{\partial L_P}{\partial w} = w - \sum_{i=1}^{l} \alpha_i y_i x_i = 0 \;\Rightarrow\; w = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{4.23}
$$

$$
\frac{\partial L_P}{\partial \xi_i} = C\xi - \alpha = 0 \;\Rightarrow\; \xi = \frac{1}{C}\alpha \tag{4.24}
$$

$$
\frac{\partial L_P}{\partial b} = \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{4.25}
$$

and resubstituting into the primal:

$$
\begin{aligned}
L_D(\alpha) &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle + \frac{1}{2C}\langle \alpha, \alpha\rangle - \frac{1}{C}\langle \alpha, \alpha\rangle \\
&= \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle - \frac{1}{2C}\langle \alpha, \alpha\rangle.
\end{aligned}
$$

Using the equation

$$\langle \alpha, \alpha \rangle = \sum_{i=1}^{l} \alpha_i^2 = \sum_{i,j=1}^{l} \alpha_i \alpha_j \delta_{ij} = \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \delta_{ij}, \tag{4.26}$$

where δ_{ij} is the Kronecker delta defined to be 1 if $i = j$ and 0 otherwise, we obtain: Solving the 2-Norm Soft Margin optimization problem is equivalent to maximizing

$$L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\langle x_i, x_j \rangle + \frac{1}{C} \delta_{ij}),$$

subject to

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0.$$

The Kuhn-Tucker complementarity conditions derived from the primal formulation are

$$\alpha_i [y_i (\langle w, x_i \rangle + b) - 1 + \xi_i] = 0 \qquad \forall i. \tag{4.27}$$

This problem can be solved with the same methods used for the maximal margin hyperplane. The only difference is the addition of $1/C$ to the diagonal of the Gram matrix $G = (\langle x_i, x_j \rangle)$.

4.3 Adaption to Imbalanced Datasets

In the previous sections we supposed that the error penalty for each of the two classes should be the same. But what happens in the case of very imbalanced datasets, where points of one class are much more numerous than points of the other class? Here, the cost of misclassifying a point from the small class should be heavier than the cost for errors on the large class. We want to preserve the points of the small class and avoid them being counted just as misclassified examples of the larger class.

The basic idea is to introduce different error weigths C^+ and C^- for the positive and the negative class, which result in a bias for larger multipliers α_i of the 'critical' class. This induces a decision boundary which is much more distant from the smaller class than from the other. See (Veropoulos et al., 1999; Karakoulas and Shawe-Taylor, 1999) for further details.

We can readily generalize the soft margin approach of the previous section, so that the primal Lagrangian has two loss functions for the two types of errors: Let $I_+ = \{i \mid y_i =$

$+1\}$ and $I_- = \{i \mid y_i = -1\}$, then the primal optimization problem is:

$$\begin{aligned} \textit{Minimize} \quad & \frac{1}{2}\|w\|^2 + C^+ \sum_{i \in I_+} \xi_i^k + C^- \sum_{i \in I_-} \xi_i^k \\ \textit{subject to} \quad & y_i(\langle w, x_i \rangle + b) - 1 + \xi_i \ \geq \ 0 \qquad i = 1, \dots, l \\ & \xi_i \geq 0 \qquad i = 1, \dots, l \end{aligned} \tag{4.28}$$

1-Norm Soft Margin. For $k = 1$ the primal Lagrangian becomes:

$$L_P(w, b, \xi, \alpha, \beta) =$$
$$\frac{1}{2}\|w\|^2 + C^+ \sum_{i \in I_+} \xi_i + C^- \sum_{i \in I_-} \xi_i - \sum_{i=1}^{l} \alpha_i (y_i(\langle w, x_i \rangle + b) - 1 + \xi_i) - \sum_{i=1}^{l} \beta_i \xi_i$$

The same straightforward computations as above show that the dual formulation is the same as in (4.22) with the exception that

$$\begin{aligned} 0 \leq \alpha_i \leq C^+ \quad & \text{for} \quad y_i = +1 \quad \text{and} \\ 0 \leq \alpha_i \leq C^- \quad & \text{for} \quad y_i = -1 \ . \end{aligned}$$

2-Norm Soft Margin. For $k = 2$ we get:

$$L_P(w, b, \xi, \alpha, \beta) =$$
$$\frac{1}{2}\|w\|^2 + \frac{C^+}{2} \sum_{i \in I_+} \xi_i^2 + \frac{C^-}{2} \sum_{i \in I_-} \xi_i^2 - \sum_{i=1}^{l} \alpha_i (y_i(\langle w, x_i \rangle + b) - 1 + \xi_i)$$

This results in a dual formulation

$$L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\langle x_i, x_j \rangle + \mathbb{I}_{[i \in I_+]} \frac{1}{C^+} \delta_{ij} + \mathbb{I}_{[i \in I_-]} \frac{1}{C^-} \delta_{ij}),$$

where $\mathbb{I}_{[\cdot]}$ is the indicator function. This can be viewed as a change in the Gram matrix G. Add $1/C^+$ to the elements of the diagonal of G which correspond to examples of the positive class and $1/C^-$ to those corresponding to examples of the negative class:

$$G'_{ii} = \begin{cases} \langle x_i, x_i \rangle + \frac{1}{C^+} & \text{for } y_i = +1 \\ \langle x_i, x_i \rangle + \frac{1}{C^-} & \text{for } y_i = -1 \end{cases}$$

Viewing adaptions of the Hard Margin algorithm as changes in the Gram matrix gives the key to a simple kernel formulation of Soft Margin type Support Vector Machines in section 5.3 *Kernel Based Classification*.

4.4 Duality of Linear Machines

In this section we will stress a fact that was remarked several times before in this chapter. The linear learning machines we constructed can be given in a dual description. This representation will turn out to be a crucial concept in the construction of Support Vector Machines.

What do we mean by the term 'duality of classifiers'? The normal vector w allows a representation as a linear combination of the training points

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i, \tag{4.29}$$

where $\mathcal{X} = \{(x_i, y_i) : \ i = 1, \ldots, l\}$ is the given training set of already classified data. The α_i were introduced in the Lagrange solution of the margin maximisation problem. They are called the dual variables and are considered to be the fundamental unknowns of the problem. The expansion of w results in the Lagrangian

$$L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{4.30}$$

and a reformulation of the decision function

$$\begin{aligned} f(x) &= sign(\langle w, x \rangle + b) \\ &= sign\left(\sum_{i=1}^{l} \alpha_i y_i \langle x_i, x \rangle + b \right). \end{aligned} \tag{4.31}$$

The important observation in these formulas is that training and test points never act through their individual attributes. In the training step we only need the training points as entries in the Gram matrix and in the test phase the new points only appear in an inner product with the training data.

This property will enable us to generalize the concept of the Optimal Separating Hyperplane to nonlinear classifiers.

Chapter 5

Nonlinear Classifiers

In the last chapter we have seen how classfiers linear in input space can easily be computed by standard optimization techniques. Although linear classifiers are easy to handle, they pose severe restrictions on the learning task. The target concept may be too complex to be expressed as a linear combination of the given attributes. This problem can be overcome by an approach called *kernel technique*. It dates back to the early 60's and was introduced as the *method of potential functions* by AIZERMAN ET AL. (1964).

The general idea is to map the input data to a high dimensional space and separate it there by a linear classifier. This will result in a classifier nonlinear in input space.

5.1 Explicit Mappings

Feature mapping. We will change the representation of the training examples by mapping the data to a (possibly infinite dimensional) Hilbert space $\mathcal{H}$. Usually $\mathcal{H}$ will have a much higher dimension than the space $\mathcal{L}$ in which the data live. ($\mathcal{L}$ and $\mathcal{H}$ are usually used as a mnemonic for 'low dimensional' and 'high dimensional'.) The mapping $\Phi : \mathcal{L} \to \mathcal{H}$ is applied to each example before training and the optimal separating hyperplane is constructed in $\mathcal{H}$. We shall call the i-th component of $\Phi(x)$ the i-th *feature* under the mapping Φ, $\mathcal{H}$ will be called the *feature space*. The task of choosing the most suitable representation of the data is known as *feature selection*.

The solution of the optimization problem will be a vector $w \in \mathcal{H}$, that can be written as a linear combination of the training inputs. From the dual formulation of the learning task we know that the algorithm only depends on inner products between training examples and test points. So the Gram matrix $G = (\langle \Phi(x_i), \Phi(x_j) \rangle)$ and the vector $(\langle \Phi(x_i), \Phi(x) \rangle)$, where x is a new test point, suffice for the learning task.

The decision function in $\mathcal{H}$ comes out as

$$f(x) = sign\left(\sum_{i=1}^{l} \alpha_i y_i \langle \Phi(x_i), \Phi(x) \rangle + b\right). \tag{5.1}$$

What do we win by using a mapping Φ? Suppose we are given a data set that is linear non-separable in the input space. One often observes that applying a mapping to a high-dimensional space will make the data separable in the feature space. So, using Φ renders our problem more suitable for an maximal margin approach. We will illustrate this by an easy example.

Example. Consider a training set $\mathcal{X} = (x_i, y_i)$ of points in $\mathbb{R}$ with class labels +1 or -1: $\mathcal{X} = \{(-1, +1), (0, -1), (+1, +1)\}$. Trivially these three points are not separable by a hyperplane (i.e. one point) in $\mathbb{R}$. We now map the training examples to $\mathbb{R}^3$ by applying

$$\Phi : \mathbb{R} \longrightarrow \mathbb{R}^3,\ x \mapsto (x^2, \sqrt{2}x, 1)^t.$$

This results in a training set consisting of vectors $\{(1, -\sqrt{2}, 1)^t, (0, 0, 1)^t, (1, \sqrt{2}, 1)^t\}$ with labels $(+1, -1, +1)$. The maximal margin hyperplane will be determined by $w = (1, 0, 0)^t$ and $b = 0.5$, as can easily be seen by the position of the data in the $\Phi_1\Phi_2$-plane. So, the learning task can be solved very easily in $\mathbb{R}^3$. How does the corresponding decision surface look like in $\mathbb{R}$? First we note, that there is no vector mapping to w via Φ. But we can write w in the form $w = \sum_{i=1}^{3} \alpha_i y_i \Phi(x_i)$ where $\alpha = (1/2, 1, 1/2)$. Using the equality

$$\langle \Phi(x_1), \Phi(x_2) \rangle = x_1^2 x_2^2 + 2x_1x_2 + 1 = (x_1x_2 + 1)^2, \tag{5.2}$$

the decision surface becomes:

$$\begin{aligned} & \sum_{i=1}^{3} \alpha_i y_i \langle \Phi(x_i), \Phi(x) \rangle + \frac{1}{2} = 0 \\ \Longleftrightarrow \quad & \frac{1}{2}(-x+1)^2 + (-1) + \frac{1}{2}(x+1)^2 + \frac{1}{2} = 0 \\ \Longleftrightarrow \quad & x^2 = \frac{1}{2} \end{aligned}$$

This leads to two separation points in $\mathbb{R}$: $x_1 = 1/\sqrt{2}$ and $x_2 = -1/\sqrt{2}$. We have shown how a simple linear separation in the feature space separates our data in input space.

Now, $\mathcal{H}$ is possibly infinite dimensional, so working with Φ explicitly won't be easy. This motivates to search for ways to evalutate inner products in $\mathcal{H}$ without making direct use of $\mathcal{H}$ and Φ. The key to the solution of this problem is an observation we can make in the example above. Equation (5.2) tells us, that the inner product in $\mathcal{H}$ is equivalent to a function in $\mathcal{L}$, in this case a second degree polynomial. Such functions will be called *kernel functions*.

5.2 The Kernel Trick

Definition 5.1 (Kernel Functions) Given a mapping $\Phi : \mathcal{L} \to \mathcal{H}$ from input space $\mathcal{L}$ to an (inner product) feature space $\mathcal{H}$, we call the function $k : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ a kernel function, iff for all $x, z \in \mathcal{L}$

$$k(x, z) = \langle \Phi(x), \Phi(z) \rangle_{\mathcal{H}}. \tag{5.3}$$

The kernel function behaves like an inner product in $\mathcal{H}$, but can be evaluated as a function in $\mathcal{L}$. This allows to side-step possible computational problems inherent in evaluating the feature map. In the last section, we started by specifying a mapping Φ before applying the learning algorithm. Now we will invert the chain of arguments and start by choosing a kernel function k. This will implicitly define a mapping Φ.

We have already seen, that both learning and training step only depend on the value of inner products in feature space. Hence, they can be formulated in terms of kernel functions. Once we have choosen such a function, the decision rule (4.31) at page 46 can be written as

$$f(x) = sign\left(\sum_{i=1}^{l} \alpha_i y_i k(x_i, x) + b\right) \tag{5.4}$$

As a consequence, we do not need to know the underlying feature map to be able to solve the learning task in feature space! What functions can be choosen as kernels? Clearly, k has to be a symmetric function, as can easily be seen by

$$k(x, z) = \langle \Phi(x), \Phi(z) \rangle = \langle \Phi(z), \Phi(x) \rangle = k(z, x) \ ,$$

and satisfy the Cauchy-Schwarz inequality,

$$\begin{aligned} k(x, z)^2 &= \langle \Phi(x), \Phi(z) \rangle \ \leq \ \|\Phi(x)\|^2 \|\Phi(z)\|^2 \\ &= \langle \Phi(x), \Phi(x) \rangle \langle \Phi(z), \Phi(z) \rangle \ = \ k(x, x) k(z, z) \ , \end{aligned}$$

but what further conditions are to be met? The answer is given by Mercer's theorem. Before we state this theorem, we try to build up an intuition on Mercer kernels by studying an example.

5.2.1 Mercer kernels

Example. Consider an input space $\mathcal{L}$ containing a finite number of elements, i.e. $\mathcal{L} = \{x_1, \ldots, x_n\}$. The kernel matrix $K = (k(x_i, x_j))$ is by definition a symmetric matrix. We now try to construct a feature space and a feature mapping from the kernel k.

Since K is a symmetric matrix there exists an decomposition $K = U^t \Lambda U$, where $\Lambda = diag(\lambda_1, \ldots, \lambda_r)$ is the diagonal matrix of eigenvalues and $U = (u_1, \ldots, u_n)$ is an orthogonal $r \times n$ matrix with the corresponding eigenvectors as columns, and $r = rank(K)$. The feature space $\mathcal{H}$ consists in the set $\mathcal{H} = \{\Phi(x_i) : x_i \in \mathcal{L}, i = 1, \ldots, n\}$. If the eigenvalues are all positive, we define a mapping Φ by

$$\Phi(x_i) = \Lambda^{\frac{1}{2}} u_i.$$

This leads to a Gram matrix G given by

$$G_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathcal{H}} = (\Lambda^{\frac{1}{2}} u_i)^t (\Lambda^{\frac{1}{2}} u_j) = u_i^t \Lambda u_j = K_{ij}.$$

Note that positivity of the eigenvalues is equivalent to assume that K is positive semidefinite. This is necessary for k to be a kernel, for else we could not define the mapping Φ. A symmetric function on a finite input space $\mathcal{L}$ is a kernel function if and only if the matrix K is positive semi-definite. Mercer's theorem is an extension of this property based on the study of integral operator theory. The following version of Mercer's theorem can be found in (Cristianini and Shawe-Taylor, 2000, p35). We state it without proof.

Theorem 5.2 (Mercer's theorem) *Let $\mathcal{L}$ be a compact subset of $\mathbb{R}^n$. Suppose k is a continuous symmetric function such that the integral operator $T_k : L_2(\mathcal{L}) \to L_2(\mathcal{L})$,*

$$(T_k f)(\cdot) = \int_{\mathcal{L}} k(\cdot, x) f(x)\ dx \tag{5.5}$$

is positive, that is

$$\int_{\mathcal{L} \times \mathcal{L}} k(x, z) f(x) f(z)\ dxdz \geq 0, \tag{5.6}$$

for all $f \in L_2(\mathcal{L})$. Then we can expand $k(x, z)$ in a uniformly convergent series (on $\mathcal{L} \times \mathcal{L}$) in terms of T_k's eigen-functions $\phi_j \in L_2(\mathcal{L})$, normalised in such a way that $\|\phi_j\|_{L_2} = 1$, and positive associated eigenvalues $\lambda_j > 0$,

$$k(x, z) = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(z). \tag{5.7}$$

Examples of Mercer kernels. The first kernels investigated were the following

$$\text{polynomial} \qquad k(x, z) = (\langle x, z \rangle + 1)^p \tag{5.8}$$

$$\text{sigmoid} \qquad k(x, z) = \tanh(\kappa \langle x, z \rangle - \delta) \tag{5.9}$$

$$\text{radial basis function} \qquad k(x, z) = \exp(-\|x - z\|^2 / 2\sigma^2) \tag{5.10}$$

The hyperbolic tangent kernel only satisfies Mercer's condition for certain values of κ and δ. This was noticed experimentally in Vapnik (1995). In the next section, we will proof that (5.8) and (5.10) really are Mercer kernels and we will show how complicated kernels are created from simple building blocks.

5.2.2 Making kernels

First, we will work out a criterion to verify that a symmetric function is a kernel. The positivity condition (5.6) in Mercer theorem

$$\int_{\mathcal{L}\times\mathcal{L}} k(x,z)f(x)f(z)\ dxdz \geq 0 \qquad \forall f \in L_2(\mathcal{L})$$

corresponds to the positive semi-definite condition in the finite case. This connection will give us a rule to judge symmetric functions, as the following lemma shows:

Lemma 5.3 *The positivity condition (5.6) in Mercer theorem holds iff for any finite subset $\{x_1,\ldots,x_l\}$ of $\mathcal{L}$, the corresponding matrix $K = (k(x_i,x_j))$ is positive semi-definite, i.e. $v^tKv \geq 0$ for all $v \in \mathbb{R}^n,\ v \neq 0$.*

Proof: ($\Rightarrow$) The first part of the proof is expanded from the outline given in (Cristianini and Shawe-Taylor, 2000, p35). To use the positivity condition on a finite set of points $\{x_1,\ldots x_l\}$ we choose f as the weighted sum of indicator functions at each x_i:

$$f(x) := \sum_{i=1}^{l} v_i \mathbb{I}_{[x=x_i]}(x) \qquad \in L_2(\mathcal{L}) \tag{5.11}$$

Then the chain of equations holds

$$\begin{aligned} 0 &\leq \int k(x,z)f(x)f(z)\ dxdz \\ &= \int k(x,z)\sum_{i=1}^{l} v_i\mathbb{I}_{[x=x_i]}\sum_{j=1}^{l} v_j\mathbb{I}_{[z=x_j]}\ dxdz \\ &= \sum_{ij} v_iv_j \int k(x,z)\mathbb{I}_{[x=x_i]}\mathbb{I}_{[z=x_j]}\ dxdz \\ &= \sum_{ij} v_iv_jk(x_i,x_j) \ = \ v^tKv \end{aligned} \tag{5.12}$$

($\Leftarrow$) We show: if the positivity condition (5.6) does not hold, then no finite set $\{x_1,\ldots,x_l\}$ can be found with positive semi-definite kernel matrix $K = (k(x_i,x_j))$. This follows directly from the above chain of equations if we substitute '$\leq$' by '$>$', because for every $\{x_1,\ldots,x_l\}$ the function f can be defined as in 5.11. $\square$

This criterion is the basic tool to show how the composition of kernels leads to new ones. We will apply it to the construction of new kernels. The next theorem shows, that kernels satisfy a number of closure properties.

Theorem 5.4 (Cristianini and Shawe-Taylor, 2000) *Let k_1 and k_2 be two kernels over $\mathcal{L} \times \mathcal{L}$, $\mathcal{L} \subseteq \mathbb{R}^n$, $\Phi : \mathcal{L} \to \mathbb{R}^m$ and k_3 a kernel over $\mathbb{R}^m \times \mathbb{R}^m$, $a \in \mathbb{R}^+$, $f(\cdot)$ a real-valued function on $\mathcal{L}$, $p(x)$ a polynomial with positive coefficients, and B a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels:*

1. $k(x, z) = k_1(x, z) + k_2(x, z)$,
2. $k(x, z) = ak_1(x, z)$,
3. $k(x, z) = k_3(\Phi(x), \Phi(z))$,
4. $k(x, z) = k_1(x, z)k_2(x, z)$,
5. $k(x, z) = f(x)f(z)$,
6. $k(x, z) = x^t Bz$.
7. $k(x, z) = p(k_1(x, z))$
8. $k(x, z) = \exp(k_1(x, z))$

Proof:

1. Fix a finite set of points $\{x_1, \ldots, x_n\}$, and let K_1 and K_2 be the corresponding matrices obtained by restricting k_1 and k_2 to these points. Consider any vector $v \in \mathbb{R}^n$. Then the first statement of the proposition follows from lemma 5.3 and the relation
$$v^t(K_1 + K_2)v \;=\; v^t K_1 v + v^t K_2 v \;\geq\; 0.$$

2. Very similiar, the second statement results from
$$v^t a K_1 v \;=\; a v^t K_1 v \;\geq\; 0.$$

3. Since k_3 is a kernel, we can use lemma 5.3 and the matrix obtained by restricting k_3 to the points $\Phi(x_1), \ldots, \Phi(x_n)$ is positive semi-definite as required.

4. Let $K = K_1 \bigotimes K_2$ be the tensor product of the matrices K_1 and K_2. The tensor product of two positive semi-definite matrices is positive semi-definite since the eigenvalues of the product are all pairs of products of the eigenvalues of the two components. The matrix corresponding to the function $k_1 k_2$ is known as the Schur product H of K_1 and K_2 with entries the products of the corresponding entries in the two components. The matrix H is a principal submatrix of K defined by a set of columns and the same set of rows. Hence for $v \in \mathbb{R}^n$, there is a corresponding $v_1 \in \mathbb{R}^{n^2}$, such that
$$v^t H v \;=\; v_1^t K v_1 \;\geq\; 0,$$
and so H is positive semi-definite as required.

5. We can arrange the bilinear form as follows:

$$\begin{aligned} v^t K v &= \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j f(x_i) f(x_j) \\ &= \sum_{i=1}^{n} \alpha_i f(x_i) \sum_{j=1}^{n} \alpha_j f(x_j) = (\sum_{i=1}^{n} \alpha_i f(x_i))^2 \\ &\geq 0, \end{aligned}$$

   as required.

6. Consider the diagonalisation of $B = V^t \Lambda V$ by an orthogonal matrix V, where Λ is the diagonal matrix containing the non-negative eigenvalues. Let $\sqrt{\Lambda}$ be the diagonal matrix with the square roots of the eigenvalues and set $A = \sqrt{\Lambda} V$. We therefore have

$$k(x, z) = x^t B z = x^t V^t \Lambda V z = x^t V^t \sqrt{\Lambda}\sqrt{\Lambda} V z = x^t A^t A z = \langle Ax, Az \rangle,$$

   the inner product using the feature mapping A.

7. For a polynomial the result is immediate by combining the previous parts of the proposition. Note that the constant is covered by part 4 of the proposition.

8. The exponential function can be arbitrarily closely approximated by polynomials with positive coefficients and hence is a limit of kernels. Since kernels are clearly closed under taking pointwise limits, the result follows.

□

This proposition enables us to show that the examples of kernels in the last section really satisfy the Mercer condition. The polynomial kernel defined in (5.8) is easily built by combining parts of the proposition and in the case of radial basis functions (5.10) we can decompose as follows:

$$\exp\left(-\frac{\|x - z\|^2}{\sigma^2}\right) = \exp\left(-\frac{\|x\|^2}{\sigma^2}\right) \exp\left(-\frac{\|z\|^2}{\sigma^2}\right) \exp\left(\frac{2\langle x, z\rangle}{\sigma^2}\right).$$

The first two factors together form a kernel by part 4 of the proposition, while the third factor is a kernel by part 8.

5.3 Kernel based Classification

In the last two chapters we gathered all the ingredients to formulate the support vector concept. We combine the idea of the optimal separating hyperplane with the kernel-induced mapping to a high dimensional feature space. Roughly speaking, a Support

Vector Machine is a highdimensional maximal margin hyperplane evaluated in the input space. In this section we will restate our previous results obtained in the input space. The difference is that we now use an inner product in the feature space generated by a kernel function.

Hard margin. To construct a support vector machine which allows for no training errors, we have to solve the optimization problem:

$$\begin{aligned} &\textit{maximize} \quad && L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\textit{subject to} && \alpha_i \geq 0 \qquad i = 1, \ldots, l \\ & && \sum_{i=1}^{l} \alpha_i y_i = 0 \end{aligned} \tag{5.13}$$

Soft margin. We distinguish between a 1-Norm and a 2-Norm soft margin Support Vector Machine. In the first case the optimization problem comes out as

$$\begin{aligned} &\textit{maximize} \quad && L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\textit{subject to} && 0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \\ & && \sum_{i=1}^{l} \alpha_i y_i = 0 \end{aligned} \tag{5.14}$$

The difference to the hard margin case is the additional constraint on the lagrange multipliers, which gives this generalization the name *box constraint*.

Writing the 2-norm soft margin machine in a kernel-based version, we get

$$\begin{aligned} &\textit{maximize} \quad && L_D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (k(x_i, x_j) + \frac{1}{C}\delta_{ij}) \\ &\textit{subject to} && \alpha_i \geq 0 \qquad i = 1, \ldots, l \\ & && \sum_{i=1}^{l} \alpha_i y_i = 0 \end{aligned} \tag{5.15}$$

Comparing this to the hard margin case we see that the only change is the addition of $\frac{1}{C}$ to the diagonal of the inner product matrix associated with the training set. This results in adding $\frac{1}{C}$ to the eigenvalues of the kernel matrix K, because for an eigenvalue λ with corresponding eigenvector v holds:

$$Kv = \lambda v \;\Rightarrow\; (K + \frac{1}{C}I)v = Kv + \frac{1}{C}v = (\lambda + \frac{1}{C})v.$$

The new problem can be understood as simply a change of kernel

$$k'(x, z) := k(x, z) + \frac{1}{C}\delta_{xz}.$$

Imbalanced Datasets. This supplies us with a new view on the adaption of Support Vector Machines on imbalanced datasets (section 4.3), because we can write it as a change of kernel, too.

$$k'(x_i, z) = \begin{cases} k(x_i, z) + \frac{1}{C^+}\delta_{x_i z} & \text{for } y_i = +1 \\ k(x_i, z) + \frac{1}{C^-}\delta_{x_i z} & \text{for } y_i = -1 \end{cases} \tag{5.16}$$

Decision function of SVM. Written in a kernel-version, the linear decision function (4.19) at page 40 comes out as a (usually non-linear) function

$$f(x) \;=\; sign\left(\sum_{i=1}^{\#SV} \alpha_i y_i k(x, x_i^{SV}) + b^*\right) \tag{5.17}$$

Model selection. On this point we give an overview of the achieved results. The steps taken in Support Vector Classification are summarized in figure 5.1.

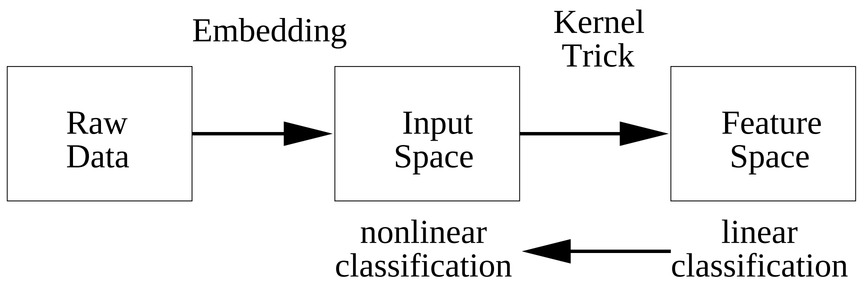

Figure 5.1: The course of Support Vector Classification. The data is embedded in a (low-dimensional) input space. By using kernel techniques separation takes place in a (high-dimensional) feature space. This results in a non-linear classifier in input space.

The problem of an appropiate embedding of the raw data is common to all mathematical classification approaches. For the case of biological sequence analysis we will deal with it in chapter 9 *Protein Fold Class Prediction*.

What is particular for the design of Support Vector Machines is the choice of kernel. This amounts to model selection. In the case of a parametrised kernel family model selection is then reduced to adjusting the parameters. In most cases it is possible to choose a kernel parameter that forces separation of the training examples, e.g. with decreasing width of a radial basis kernel all data will become separable. In general, this will not guarantee a high generalization ability but will easily lead to overfitting.

A trap. What is wrong with the following argument? 'Support Vector Machines construct a Maximal Margin Hyperplane between the elements of the positive and the negative class. This hyperplane only depends on the geometry and distances of the datapoints. Therefore, the prediction outcome does not change if we blow up the whole dataset by a *constant factor* $c > 0$.'

Indeed, if we look what happens in the optimization problem (4.6) at page 34 for x_i substituted by cx_i, we find that the same vector w^* will be found as a solution. But the above argument overlooks the fact, that SVM don't do the separation in input space but in a feature space defined by the kernel function. What happens to the kernel function if $x_i \mapsto cx_i$? A striking example is provided by radial basis function kernels.

For rbf-kernels the change is shown in the chain of equations:

$$k_\sigma(cx, cz) \;=\; \exp\left(\frac{-c^2\|x-z\|^2}{2\sigma^2}\right) \;=\; \exp\left(\frac{-\|x-z\|^2}{2(\sigma^2/c^2)}\right) \;=\; k_{\frac{\sigma}{c}}(x,z)$$

From this follows that $k(cx, cz)$ with width σ is identical to $k(x, z)$ with width σ/c. The mapping $x_i \mapsto cx_i$ results in a change of the kernel width, which can have tremendous impact on the generalisation performance.

A warning. Users of Support Vector Machines usually want to understand 'what this machine really is doing!' The example above shows that all considerations on how SVM work on the data mustn't be restricted to the input space. 'Linearly separable' means 'linearly separable *in feature space*' and the margin, too, is computed in a high-dimensional space we have no direct access to, but can only reach via the implicit mapping inherent in the kernel function.

Is SVM a black box method? It seems, that we have no direct grasp of the feature space and the datapoints mapped herein. The details of how Support Vector Machines handle the training examples are hidden from the user by the kernel function. Hence, SVM seem to fall into the same category of black box methods as Neural Networks.

This is only partly right. In chapter 7.2 we will show that for homogeneous polynomial Support Vector Machines the kernel mapping can be made explicit. If we are interested in how polynomial SVM treat the data, we can do all computations directly in the feature space. But the point in all kernel techniques is, that we don't *need* to do this. The kernel function provides an elegant way of working in the feature space avoiding all the troubles and difficulties inherent in high dimensions.

Even if we are not able to make the kernel mapping explicit, SVM give valuable information of how they work on the data. They provide us not only with a ready-to-use classifer, but also with a list of conspicuous training points, the support vectors. These points lie at the boundary of the classes and thus can be used to gain insight in the shape and form. This shows, that Support Vector Machines are no black box methods, but can be given a clear interpretation.

Chapter 6

Multiclass Classification

Until now we limited our study to the case of two-class discrimination. However, in real world problems we often have to deal with $n \geq 2$ classes. Our training set will consist of pairs (x_i, y_i), where $x_i \in \mathbb{R}^n$ and $y_i \in \{1, \ldots, n\}$ $, i = 1, \ldots, l$. How can we extend our methods to the multiclass case? The next two sections describe the reduction of multiple classification problems to the two-class case. The third section deals with a direct solution.

6.1 one-versus-rest

The construction of a n-class classifier using two-class discrimination methods is usually done by the following procedure:

Construct n two-class decision functions $f_k(x) = sign(\langle w_k, x\rangle + b_k)$, $k = 1, \ldots, n$, that separate examples of class k from the training points of all other classes, i.e.

$$f_k(x) = \begin{cases} +1 & \text{if } x \text{ belongs to class } k \\ -1 & \text{otherwise} \end{cases} \tag{6.1}$$

In the two-class case we would use this discrimination function to do the classification. But in the multiclass case this will lead to ambiguities, as the plane is divided into several areas where more than one classifier is active.

As a way out, we drop the sign and consider the vector of real-valued outputs $f(x) = (f'_1(x), \ldots, f'_n(x))$, where $f'_k(x) = \langle w_k, x\rangle + b_k$. Geometrically, each component of this vector is the distance of x to one of the n separating hyperplanes. We choose x to belong to the class corresponding to the maximal value of functions $f'_k(x)$, $k = 1, \ldots, n$:

$$k^* = \underset{k}{\operatorname{argmax}}\, f'_k(x) \tag{6.2}$$

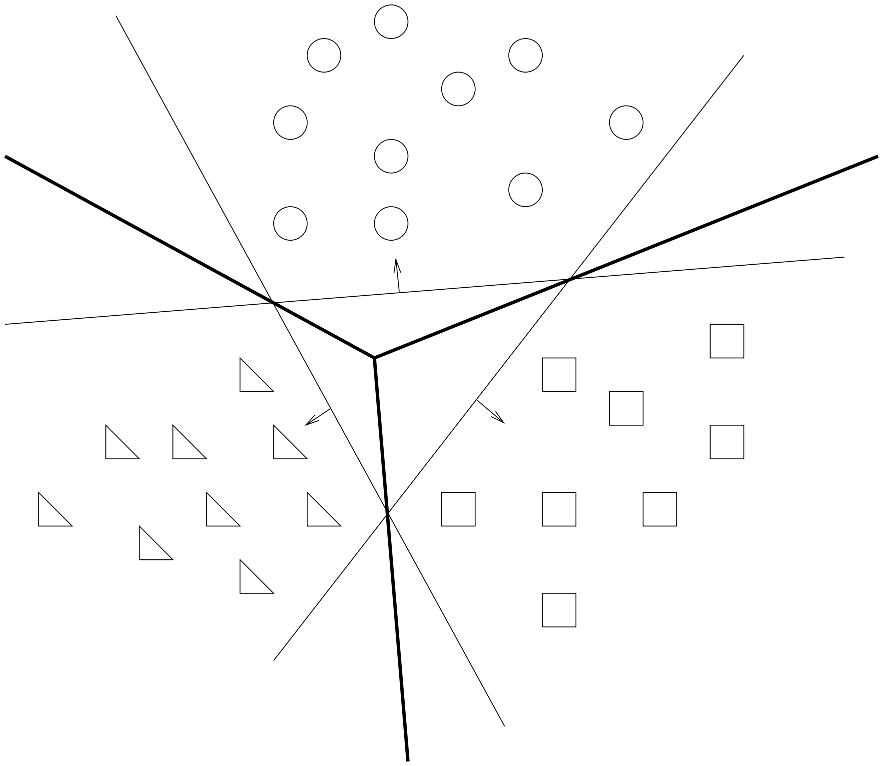

Figure 6.1: Winner-takes-all for a three class example. The thin lines represent the hyperplanes obtained from separating each class from all others (arrows point to the direction of the positive class); the resulting borderlines after tie breaking are drawn thick. Only three points coincide.

This approach is called *winner-takes-all*. The major drawback of this heuristic is, that only three points of the resulting borderlines coincide with the preliminary decision functions calculated by the n support vector machines. So we seem to loose the benefits of maximal margin hyperplanes. The heuristic improves the hard decisions of the binary classifiers, but does not yield optimal decision boundaries which could be obtained by simultaneous opimisation of all three borders. The next heuristic uses discrimination between all pairs of classes, which fits better into the support vector concept.

6.2 one-versus-one

We define a decision function $f_{kl} : \mathbb{R}^n \to \{+1, -1\}$ for each pair of classes (k, l):

$$f_{kl}(x) = \begin{cases} +1 & \text{if } x \text{ belongs to class } k \\ -1 & \text{if } x \text{ belongs to class } l \end{cases} \tag{6.3}$$

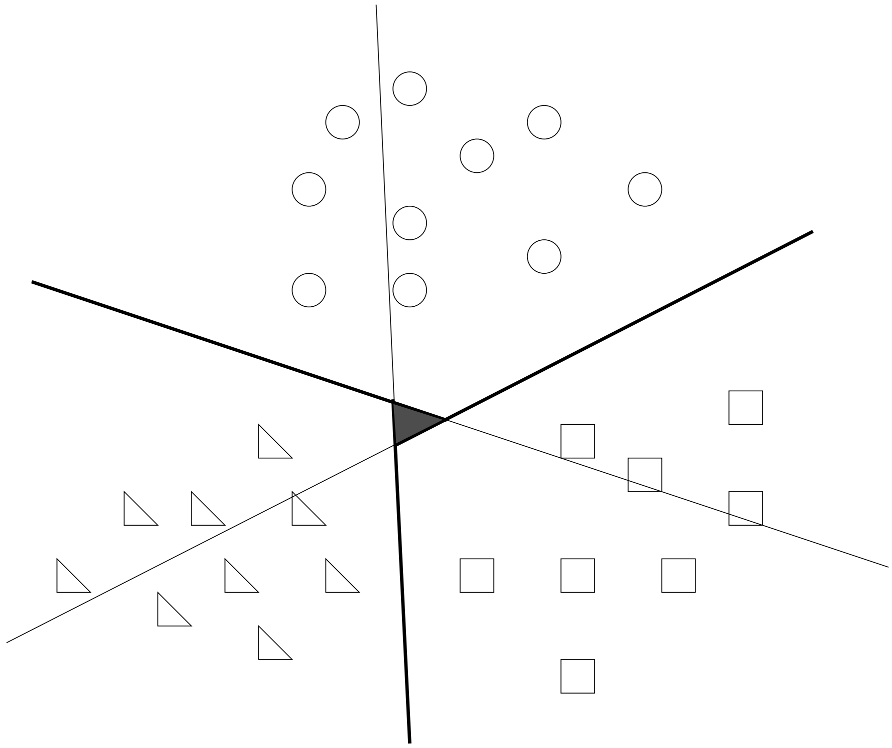

Figure 6.2: The decision functions between pairs are drawn in thin lines, the resulting borderlines are emphasized by thick lines. For the shaded region in the middle additional tie breaking is necessary.

Since the pairwise approach is symmetric, it follows $f_{kl} = -f_{lk}$, further we define $f_{kk} = 0$. We end up with $\binom{n}{2} = n(n-1)/2$ different pairwise decision functions. To reach a class decision we sum up according to:

$$f_k(x) = \sum_{l=1}^{n} f_{kl}(x) \tag{6.4}$$

As in the section above, the class decision is given by $\text{argmax}_k\, f_k$. Without ties, the condition holds $\max_k = n - 1$, i.e. the winning class gets exactly $n - 1$ positive votes. But again, there may be ambiguities. For the small shaded region in the picture above - where each class wins against one of the others and looses against the third - we need again the distance to the hyperplanes for tie-breaking. KRESSEL (1999) states, that in most examples tie situations only play a minor role for pairwise classification. The advantage of this method is, that the final borderlines are parts of the calculated pairwise decision functions, which was not the case in the one-versus-rest approach.

6.3 The direct Approach

This section is worked out from the brief outline given by Vapnik (1998). A similiar version can be found in (Weston and Watkins, 1998), who independently thought of a method to solve the multiclass classification problem directly. We will show how it can be done by Lagrange multipliers.

One can generalize the optimization problem of the maximal margin hyperplane for linear non-separable data to the following formulation. Given a training set $\mathcal{X} = \{(x_i, y_i) : \ i = 1, \dots, l\}$, where $x_i \in \mathbb{R}^n$ and $y_i \in \{1, \dots, n\}$. Class k occurs l_k times, $\sum_k l_k = l$. Then solve:

$$\begin{aligned} &\textit{minimize} && \frac{1}{2}\sum_{k=1}^{n} \|w_k\|^2 + C\sum_{k=1}^{n}\sum_{i=1}^{l_k} \xi_i^k \\ &\textit{subject to} && \langle w_k, x_i\rangle + b_k - \langle w_m, x_i\rangle - b_m \geq 1 - \xi_i^k \quad \textit{for } y_i = k, \end{aligned} \tag{6.5}$$

where $\xi_i^k \geq 0,\ k = 1, \dots, n,\ m \neq k,\ i = 1, \dots, l_k$. This gives the decision function

$$f(x) = \operatorname*{argmax}_k \ f_k(x) = \operatorname*{argmax}_k \ (\langle w_k, x\rangle + b_k), \quad k = 1, \dots, n. \tag{6.6}$$

The difference to the binary classification problem consists in the summation over all n classes. Introducing lagrange multipliers, we can state the problem in dual variables. Find the saddle point of the Lagrangian:

$$\begin{aligned} &L_P(w, b, \xi, \alpha, \beta) = \\ &\quad \frac{1}{2}\sum_{k=1}^{n} \|w_k\|^2 + C\sum_{k=1}^{n}\sum_{i=1}^{l_k} \xi_i^k - \sum_{k=1}^{n}\sum_{i=1}^{l_k} \beta_i^k \xi_i^k \\ &\quad - \sum_{k=1}^{n}\sum_{m\neq k}\sum_{i=1}^{l_k} \alpha_i^{k,m}[\langle (w_k - w_m), x_i^k\rangle + b_k - b_m - 1 + \xi_i^k] \end{aligned}$$

This can be rewritten as

$$\begin{aligned} &L_P(w, b, \xi, \alpha, \beta) = \\ &\quad \frac{1}{2}\sum_{k=1}^{n} \|w_k\|^2 - \sum_{k=1}^{n}\sum_{m\neq k}\sum_{i=1}^{l_k} \alpha_i^{k,m}[\langle w_k, x_i^k\rangle + b_k] \\ &\quad + \sum_{k=1}^{n}\sum_{m\neq k}\sum_{i=1}^{l_k} \alpha_i^{k,m}[\langle w_m, x_i^k\rangle + b_m] + \sum_{k=1}^{n}\sum_{m\neq k}\sum_{i=1}^{l_k} \alpha_i^{k,m} \\ &\quad + \sum_{k=1}^{n}\sum_{i=1}^{l_k}[C - \sum_{m\neq k} \alpha_i^{k,m} - \beta_i^k]\xi_i^k \end{aligned}$$

At the saddlepoint the derivatives with respect to w, b and ξ must vanish. This leads to the three equations:

$$\begin{aligned}
\frac{\partial L_P}{w_k} &= w_k - \sum_{m \neq k} \sum_{i=1}^{l_k} \alpha_i^{k,m} x_i^k + \sum_{m \neq k} \sum_{j=1}^{l_m} \alpha_j^{m,k} x_j^m \quad = 0, \\
\frac{\partial L_P}{b_k} &= \sum_{m \neq k} \sum_{i=1}^{l_k} \alpha_i^{k,m} - \sum_{m \neq k} \sum_{j=1}^{l_m} \alpha_j^{m,k} \quad = 0, \\
\frac{\partial L_P}{\xi_i^k} &= C - \sum_{m \neq k} \alpha_i^{k,m} - \beta_i^k \quad = 0.
\end{aligned}$$

The last equation cancels the terms in ξ in the primal Lagrangian. From the first equation we derive a expansion of the normal vector w_k of each of the n hyperplanes:

$$w_k = \sum_{m \neq k} \left(\sum_{i=1}^{l_k} \alpha_i^{k,m} x_i^k + \sum_{j=1}^{l_m} \alpha_j^{m,k} x_j^m \right) \qquad k = 1, \ldots, n \tag{6.7}$$

The second equation leads to constraints on the lagrange multipliers:

$$\sum_{m \neq k} \sum_{i=1}^{l_k} \alpha_i^{k,m} = \sum_{m \neq k} \sum_{j=1}^{l_m} \alpha_j^{m,k} \tag{6.8}$$

Resubstituting both equations into L_P yields the dual Lagrangian. Thus, the dual optimisation problem can be formulated as:

Maximize

$$\begin{aligned}
L_D(\alpha) &= \sum_{k=1}^{n} \sum_{m \neq k} \left[\sum_{i=1}^{l_k} \alpha_i^{k,m} - \frac{1}{2} \sum_{m^* \neq k} \left(\sum_{i,j=1}^{l_k} \alpha_i^{k,m^*} \alpha_j^{k,m} \langle x_i^k, x_j^k \rangle \right. \right. \\
& \left. \left. + \sum_{i=1}^{l_m} \sum_{j=1}^{l_{m^*}} \alpha_i^{m,k} \alpha_j^{m^*,k} \langle x_i^m, x_j^{m^*} \rangle \; - 2 \sum_{i=1}^{l_k} \sum_{j=1}^{l_m} \alpha_i^{k,m^*} \alpha_j^{m,k} \langle x_i^k, x_j^m \rangle \right) \right]
\end{aligned}$$

subject to the constraints

$$0 \leq \sum_{m \neq k} \alpha_i^{k,m} \leq C \,,$$

$$\sum_{m \neq k} \sum_{i=1}^{l_k} \alpha_i^{k,m} = \sum_{m \neq k} \sum_{j=1}^{l_m} \alpha_j^{m,k} \qquad k = 1, \ldots, n.$$

As a solution we get the functions $f_k(x)$, having the following expansion on support vectors

$$f_k(x) = \sum_{m \neq k} \sum_{i=1}^{l_k} \alpha_i^{k,m} \langle x_i^k, x \rangle + \sum_{m \neq k} \sum_{j=1}^{l_m} \alpha_j^{m,k} \langle x_j^m, x \rangle \; + b_k. \tag{6.9}$$

For $n = 2$ this solution coincides with the solution given in the binary case. For $n > 2$ we have to estimate simultaneously $l(n-1)$ parameters $\alpha_i^{k,m}$.

As before, to construct the non-linear SV machine we only have to exchange the inner product $\langle x_i^r, x_j^s \rangle$ by the kernel $k(x_i^r, x_j^s)$ in the corresponding equations.

6.4 Comparison of Multiclass Methods

The direct approach is a straightforward generalization of the support vector concept to more than two classes. The pairwise one-versus-one method conserves most of the maximal margin hyperplanes, but the simple one-versus-rest scheme coincides only in single points with the constructed optimal separating hyperplanes.

Why does Vapnik himself use one-versus-rest in his experiments (Vapnik, 1998, p498)? The advantage of one-versus-rest over one-versus-one is that we only have to construct one hyperplane for each of the n classes instead of $\binom{n}{2}$ pairwise decision functions. This decreases the computational effort by a factor of $(n-1)/2$. Work was done to reduce this factor. In some examples it can be brought down to $(n-1)/6$ (Kressel, 1999, p261). But in datasets containing many different classes this will be still to large for efficient computation.

In one-versus-rest, the hyperplanes are determined by ln Lagrange multipliers, which is almost the same number as for the direct approach, where we have to compute $l(n-1)$ parameters. What makes one-versus-rest advantageous here is the opportunity to choose different kernels for each separation, which is impossible in a joint computation (Vapnik, 1998, section 12.2.2).

Chapter 7

Generalization Theory

7.1 A Distribution Free Bound on the Risk

In chapter 1 we introduced the principle of empirical risk minimization (ERM). Recall the definitions

$$\begin{aligned} R(\alpha) &= \int L(z,\alpha)dP(z) & R_{emp}(\alpha) &= \frac{1}{l}\sum_{i=1}^{l} L(z_i,\alpha),\ z_i \in \mathcal{X} \\ \alpha^* &= \operatorname*{argmin}_{\alpha\in\Lambda}\ R(\alpha) & \alpha &= \operatorname*{argmin}_{\alpha\in\Lambda} R_{emp}(\alpha), \end{aligned}$$

where $z = (x, y)$ is a labeled observation, $L(z, \alpha) := \frac{1}{2}|y - f(x, \alpha)|$ is the loss function and $\mathcal{X} = \{(x_1, y_1), \ldots, (x_l, y_l) : x_i \in \mathbb{R}^n, y_i \in \{-1, +1\}\}$ is the training set.

The goal of this section is to estimate the generalization ability of the ERM principle. We have to answer the questions

1. What value of risk does $L(z, \alpha)$ provide?
2. How close is $R(\alpha)$ to $R(\alpha^*)$?

To answer these questions we have to estimate $R(\alpha)$ and the difference $\Delta(\alpha) = R(\alpha) - R(\alpha^*)$. Both answers are based on the study of the rate of uniform convergence

$$\sup_{\alpha\in\Lambda} |R(\alpha) - R_{emp}(\alpha)| \xrightarrow{P} 0 \quad \text{as } l \to \infty.$$

7.1.1 Entropy and Growth Function

Let $z_1, \ldots, z_l$ be random independent observations. Consider the number of different separations of the sample by a given set of indicator functions $\{f(z, \alpha) : \alpha \in \Lambda\}$:

$$N^{\Lambda}(z_1, \ldots, z_l) = \#\{f(z_1, \alpha), \ldots, f(z_l, \alpha) : \alpha \in \Lambda\} \leq 2^l$$

Using this number we define the *entropy* $H^\Lambda(l)$, *annealed entropy* $H^\Lambda_{ann}(l)$ and *growth function* $G^\Lambda(l)$ for a set of indicator functions on a sample of size l by:

$$\begin{aligned} H^\Lambda(l) &= \mathbb{E}\, ln\, N^\Lambda(z_1, \ldots, z_l), \\ H^\Lambda_{ann}(l) &= \ln\, \mathbb{E}\, N^\Lambda(z_1, \ldots, z_l), \\ G^\Lambda(l) &= \ln \sup_{z_1,\ldots,z_l} N^\Lambda(z_1, \ldots, z_l). \end{aligned}$$

Lemma 7.1 *These quantities are constructed such that the inequality holds:*

$$H^\Lambda(l) \;\le\; H^\Lambda_{ann}(l) \;\le\; G^\Lambda(l).$$

Proof. The first inequality follows from from the Jensen inequality, since $ln(x)$ is a concave function. The second inequality is obvious. □

The growth function does not depend on a probability measure and can be shown to either (a) satisfy $G^\Lambda(l) = l \ln 2$ or (b) to be bounded by

$$G^\Lambda(l) \begin{cases} = & l \ln 2 & l \le h \\ \le & h\left(1 + \ln \frac{l}{h}\right) & l > h \end{cases}$$

where h is the largest integer for which $G^\Lambda(l) = h \ln 2$ (Vapnik, 1998, p145).

7.1.2 The Main Theorem

Theorem 7.2 *The inequality holds true:*

$$P\,\{\sup_{\alpha\in\Lambda} |R(\alpha) - R_{emp}(\alpha)| \;>\; \epsilon\} \;<\; 4 \exp\left(H^\Lambda_{ann}(2l) - l\left(\epsilon - \frac{1}{l}\right)^2\right)$$

Idea of Vapniks proof. The complete proof is stated in Appendix D. Here, we restrict our presentation to a short summary of the main idea.

A sample of size $2l$ is split randomly into two half-samples $z_1, \ldots, z_l$ and $z_1, \ldots, z_{2l}$. The difference between the empirical risk and the actual risk can be upper bounded by the difference between the frequency of errors on the first half-sample, $R^{(1)}_{emp}(\alpha)$, and on the second half-sample, $R^{(2)}_{emp}(\alpha)$:

$$P\,\{\sup_{\alpha\in\Lambda} |R(\alpha) - R^{(1)}_{emp}(\alpha)| > \epsilon\} \;<\; 2\; P\,\{\sup_{\alpha\in\Lambda} |R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| > \epsilon - \frac{1}{l}\}\;, \quad (7.1)$$

where $\epsilon_* := \epsilon - 1/l$. By using the triangle inequality and the Chernoff-bound

$$P\,\{|R(\alpha) - R_{emp}(\alpha)| > \epsilon\} \;\le\; \exp(-2\epsilon^2 l)$$

we get, that for a fixed sample of size $2l$ the inequality holds true:

$$P\ \{|R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| > \epsilon_*\}\ <\ 2\ \exp(-\epsilon_*^2 l) \tag{7.2}$$

The advantage of considering errors over a finite set of $2l$ sample points is that the hypothesis space has effectively become finite, since there cannot be distinguished more than $N^\Lambda(z_1, \ldots, z_{2l})$ classification functions on $2l$ points, i.e. we reduced the problem to a finite set of loss functions $\{L(z, \alpha') :\ \alpha' \in \Lambda' \subset \Lambda\}$:

$$\begin{aligned} & P\ \{\sup_{\alpha\in\Lambda} |R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| > \epsilon_*\ \mid z_1, \ldots, z_{2l}\} \\ &\quad =\ P\ \{\sup_{\alpha'\in\Lambda'} |R^{(1)}_{emp}(\alpha') - R^{(2)}_{emp}(\alpha')| > \epsilon_*\ \mid z_1, \ldots, z_{2l}\} \end{aligned}$$

Replacing the sup operation by summation we get:

$$\begin{aligned} \ldots\ \le\ & \sum_{\alpha'\in\Lambda'} P\ \{|R^{(1)}_{emp}(\alpha') - R^{(2)}_{emp}(\alpha')|\ >\ \epsilon_*\ \mid z_1, \ldots, z_{2l}\} \\ \le\ & 2\ N^\Lambda(z_1, \ldots, z_{2l})\ \exp(-\epsilon_*^2 l), \end{aligned}$$

by the cardinality of Λ' and by (7.2). To get bounds for a random sample of size $2l$ it is sufficient to take expectation with respect to the probability measure on the sample space:

$$\begin{aligned} & P\ \{\sup_{\alpha\in\Lambda} |R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| > \epsilon_*\} \\ &\quad =\ \mathbb{E}\ P\ \{\sup_{\alpha\in\Lambda} |R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| > \epsilon_* \mid z_1, \ldots, z_{2l}\} \\ &\quad <\ 2\ \exp(H^\Lambda_{ann}(2l) - \epsilon_*^2 l) \end{aligned} \tag{7.3}$$

Combining the bound (7.3) with (7.1) and recalling that $\epsilon_* := \epsilon - 1/l$ proves the theorem.

□

Theorem 7.2 will answer the two questions about the generalization ability of the ERM principle. We can rewrite the inequality

$$P\ \{\sup_{\alpha\in\Lambda} |R(\alpha) - R_{emp}(\alpha)|\ >\ \epsilon\}\ <\ 4\ \exp\left(H^\Lambda_{ann}(2l) - l\left(\epsilon - \frac{1}{l}\right)^2\right)$$

in an equivalent form by equating the right-hand side to some positive value $0 < \eta < 1$ and solve with respect to ϵ:

$$\begin{aligned} 4\exp\left(H^\Lambda_{ann}(2l) - l\left(\epsilon - \tfrac{1}{l}\right)^2\right)\ &=\ \eta \quad \Longleftrightarrow \\ H^\Lambda_{ann}(2l) - l\left(\epsilon - \tfrac{1}{l}\right)^2\ &=\ -\ln\tfrac{4}{\eta} \quad \Longleftrightarrow \\ \epsilon = \sqrt{\tfrac{1}{l}\left(H^\Lambda_{ann}(2l) + \ln\tfrac{4}{\eta}\right)} + \tfrac{1}{l} \end{aligned}$$

Then using this ϵ we get

$$P\left\{\sup_{\alpha\in\Lambda} |R(\alpha) - R_{emp}(\alpha)| \;>\; \epsilon\right\} \;<\; \eta$$

This implies: *With probability at least $1-\eta$ simultaneously for all functions in the set $\{L(z,\alpha)\}_{\alpha\in\Lambda}$ the inequality holds true:*

$$R(\alpha) \;\leq\; R_{emp}(\alpha) + \sqrt{\frac{1}{l}\left(H_{ann}^{\Lambda}(2l) + \ln\frac{4}{\eta}\right)} + \frac{1}{l}. \tag{7.4}$$

Because we know that $H_{ann}^{\Lambda}(2l) \;<\; G^{\Lambda}(2l) \;<\; h(1+\ln(2l/h))$ we can rewrite the last equation as:

$$R(\alpha) \;\leq\; R_{emp}(\alpha) + \sqrt{\frac{1}{l}\left(h\left(1+\ln\frac{2l}{h}\right) + \ln\frac{4}{\eta}\right)} + \frac{1}{l}. \tag{7.5}$$

In particular this inequality holds for α, which minimizes the empirical risk on the training set. Thus we have solved the first problem of estimating $R(\alpha)$.

By use of Chernoff bounds Vapnik shows that for α^* minimizing the expected risk, with probability $1-\eta$ the inequality

$$R(\alpha^*) \;>\; R_{emp}(\alpha^*) - \sqrt{\frac{-\ln\eta}{2l}}$$

is valid. The combination of these two inequalities solves the second problem of estimating the difference $\Delta(\alpha) = R(\alpha) - R(\alpha^*)$. With probability at least $1-2\eta$ the inequality holds true:

$$\Delta(\alpha) \;<\; \sqrt{\frac{1}{l}\left(h\left(1+\ln\frac{2l}{h}\right) + \ln\frac{4}{\eta}\right)} - \sqrt{\frac{-\ln\eta}{2l}} + \frac{1}{l}.$$

The bound (7.4) still depends on the distribution of the data, but this dependence is removed in (7.5) by using an upper bound of the growth function. (7.5) is a constructive bound, since the VC dimension in many cases is easily constructed as we have seen in the second chapter.

In the first chapter we used the bound derived from theorem 7.2 to motivate the structural risk minimization principle: the bound on the risk is minimized by a decision function which achieves a low error rate on the training set and belongs to a function class of low capacity. Does this explain the excellent performance, Support Vector Machines display in various applications? Do SVM combine a low training error with a low VC dimension? Clearly, they are constructed to minimize the empirical risk. Hence, to answer the question we have to study the capacity of SVMs measured by the VC dimension.

7.2 VC dimension of Support Vector Machines

We will compute the VC dimension of SVM constructed on homogeneous polynomial kernels and radial basis kernels, i.e.

$$k(x_1, x_2) = \langle x_1, x_2 \rangle^d \quad \text{and} \quad k(x_1, x_2) = \exp\left(\frac{-\|x_1 - x_2\|^2}{2\sigma^2} \right).$$

Theorem 7.3 (VCdim of SVM) *Support Vector Machines with radial basis kernels have infinite VC dimension.*

Proof. By Theorem 2.3 at page 20 we have to show: for all l exists $x_1, \ldots, x_l$ such that $\{\Phi(x_1), \ldots, \Phi(x_l)\}$ are linearly independent in $\mathcal{H}$.

From the chain of equations for all $a = (a_1, \ldots, a_l) \neq 0$

$$\begin{aligned} 0 \;\leq\; \|\sum_{i=1}^{l} a_i \Phi(x_i)\|^2 \;&=\; \langle \sum_{i=1}^{l} a_i \Phi(x_i), \sum_{j=1}^{l} a_j \Phi(x_j) \rangle \\ &=\; \sum_{i,j=1}^{l} a_i a_j \langle \Phi(x_i), \Phi(x_j) \rangle \\ &=\; \sum_{i,j=1}^{l} a_i a_j k(x_i, x_j) \;=\; a^t K a \end{aligned}$$

follows that the proposition to be shown is equivalent to $K = (k(x_i, x_j))$ being positive definite for all l. By Lemma 5.3 at page 51 we know that all Mercer kernels are positive *semi*definite. What remains to show is $a^t K a \neq 0$ for all $a \neq 0$.

For radial basis kernels we can choose the data such that

$$a^t K a \;=\; \sum_{i,j=1}^{l} a_i a_j k(x_i, x_j) \;\approx\; \sum_{i=1}^{l} a_i^2 \;>\; 0,$$

because $k(x_i, x_j) \to 0$ as $\|x_i - x_j\| \to \infty$ and $k(x_i, x_i) = 1$. But even if the training data is restricted to lie in a bounded subset of $\mathbb{R}^{d_L}$ we can achieve $k(x_i, x_j) \to 0$ by $\sigma \to 0$. □

Against the background of theorem 7.2 this is a disappointing result, because with infinite VC dimension the bound (7.5) becomes useless. In the case of polynomial SVM we will now show the VC dimension to be finite, but growing rapidly with the degree of the polynomial. From now on, we consider SVM for the separable case or where the error weight C is allowed to take all values (so that, if the points are linearly separable, a C can be found such that the solution does indeed separate them).

A kernel function k implicitly defines a mapping to some highdimensional space $\Phi : \mathcal{L} \to \mathcal{H}$. The space $\mathcal{H}$ with minimal dimension is called the minimal embedding space of k.

Lemma 7.4 *Let k be a Mercer kernel with minimal embedding space $\mathcal{H}$. Then the VC dimension h of the corresponding SVM satisfies*

$$h = dim(\mathcal{H}) + 1$$

Proof. (Burges, 1998) This is a corollary to theorem 2.3 at page 20. If the minimal embedding space has dimension d_H, then d_H points can be found whose position vectors in $\mathcal{H}$ are linearily independent. From theorem 2.3 these vectors can be shattered by hyperplanes in $\mathcal{H}$. Thus the family of SVM with kernel k can also shatter these points and hence has VC dimension $h = d_H + 1$. $\square$

We will use this lemma to compute the VC dimension of SVM with homogeneous polynomial kernels of degree p.

Theorem 7.5 *Let $dim(\mathcal{L}) = d_L$. Then for homogeneous polynomial kernels $k(x_1, x_2) = \langle x_1, x_2 \rangle^p$, p fixed, the VC dimension h of the corresponding SVM is*

$$h = \binom{p + d_L - 1}{p} + 1$$

Proof. (Burges, 1998) We will show: if the data lives in an input space $\mathcal{L}$ of dimension d_L, the dimension of the minimal embedding space for homogeneous polynomial kernels of degree p is $\binom{p+d_L-1}{p}$.

Similiar to the example on page 48, one can explicitely construct the feature map Φ. In a homogeneous polynomial, all monoms have the same degree p. We can label the components of $\Phi(x)$ by the powers $r = (r_1, \ldots, r_{d_L}) \in \mathbb{N}^{d_L}$ of the components of x:

$$\Phi_r(x) \;=\; \sqrt{\frac{p!}{r_1! r_2! \cdots r_{d_L}!}}\; x_1^{r_1} x_2^{r_2} \cdots x_{d_L}^{r_{d_L}}, \tag{7.6}$$

where $\sum_1^{d_L} r_i = p$. Each component Φ_r is uniquely identified by the choice of d_L positive integers r_i, which sum up to p. How many of such components are there? Consider p objects distributed among d_L compartments. This can be done in $\binom{p+d_L-1}{p}$ ways. ($d_L - 1$ is the number of partitions between the compartments. Thus, we have $p + d_L - 1$ entities (objects plus partitions) from which p are to be chosen as partitions.) Suppose m objects fall into compartment q. Let this correspond to a term x_q^m in the product in equation (7.6). Thus the number of distinct terms Φ_r equals the number of different distributions of the p objects among d_L compartments. This shows that the number of components of Φ is equal to $\binom{p+d_L-1}{p}$.

The components of $\Phi: \mathcal{L} \to \mathcal{H}$ are linearily independent functions and span a linear space $\Phi(\mathcal{L}) \subseteq \mathcal{H}$. We will show that $\Phi(\mathcal{L}) = \mathcal{H}$. For suppose that $\Phi(\mathcal{L}) \subset \mathcal{H}$, then there exists a fixed nonzero vector $v \in \mathcal{H}$ such that

$$\langle v, \Phi(x)\rangle = 0 \quad \text{for all } x \in \mathcal{L} \,. \tag{7.7}$$

Since the last equation holds for all x, and since the mapping Φ in equation (7.6) certainly has all derivatives defined, we can apply the operator

$$\left(\frac{\delta}{\delta x}\right)^{r} = \left(\frac{\delta}{\delta x_1}\right)^{r_1} \cdots \left(\frac{\delta}{\delta x_{d_L}}\right)^{r_{d_L}}$$

to equation (7.7), which will pick that one term with corresponding powers of x_i in equation (7.6), giving $v_{r_1,\ldots,r_{d_L}} = 0$. Because this is true for all choices of $r_1, \ldots, r_{d_L}$, $\sum_1^{d_L} r_i = p$, we get $v \equiv 0$. Hence $\Phi(\mathcal{L}) = \mathcal{H}$. □

The VC dimension $h = \binom{d_L+p-1}{p} + 1$ gets very large very quickly. So theorem 7.5 presents a new proof of theorem 7.3 in the polynomial case: clearly, for all l we can choose a degree p, such that $h > l$.

7.3 Maximal Margin Bounds

The next two sections give an overview of new developments in generalization theory based on the article *Generalization Performance of Support Vector Machines and Other Pattern Classifiers* by BARTLETT and SHAWE-TAYLOR (1999). We show generalizations of Vapniks theory.

The distribution free bound on the risk of theorem 7.2 cannot explain the excellent generalization performance of SVM, since the VC confidence term depends on the VC dimension which grows too quickly for the bound to be informative.

To gain better bounds we have to use the fact, that the Optimal Separating Hyperplane achieves maximal margin of separation. We will replace the analysis of VC dimension by the analysis of the so called fat-shattering dimension.

Definition 7.6 (Fat-shattering Dimension) Let $\mathcal{F}$ be a class of real-valued functions defined on a domain X. We say that a set of points $\mathcal{X} = \{x_1, \ldots, x_l\} \in X^l$ is γ-shattered by $\mathcal{F}$ relative to $r = (r_i)_{i=1,\ldots,l}$ if there are real numbers r_i, $i = 1, \ldots, l$ such that for every binary classification $b \in \{-1, +1\}^l$ there exists $f_b \in \mathcal{F}$ such that

$$f_b(x_i) \begin{cases} \geq \; r_i + \gamma & \text{if } b_i = +1 \\ < \; r_i - \gamma & \text{if } b_i = -1 \end{cases}$$

The fat-shattering dimension $fat_{\mathcal{F}}: \mathbb{R}^+ \to \mathbb{N}$ maps a value γ to the size of the largest γ-shattered set, if this is finite, or to infinity otherwise.

The fat-shattering dimension resembles the VC dimension obtained by thresholding, but further requires the outputs to be γ above or beyond this threshold.

We will consider $\mathcal{F} = \{x \mapsto \langle w, x \rangle \ : \ \|w\| \leq 1, \ \|x\| \leq R\}$, where the datapoints are all contained in the ball X of radius R in an Hilbert space H: $X = \{x \in H \ : \ \|x\| \leq R\}$.

Theorem 7.7 (Fat-shattering Dimension of SVM)

$$fat_{\mathcal{F}}(\gamma) \ \leq \ \left(\frac{R}{\gamma}\right)^2$$

Theorem 7.8 (Expected Risk of SVM) *There is a constant c such that for all probability distributions with probability at least $1-\eta$ over l independently generated training examples, if a classifier $sign(f) \in sign(\mathcal{F})$ has margin at least γ on all training examples, then the error of $sign(f)$ is no more than*

$$\frac{c}{l}\left(\frac{R^2}{\gamma^2}\log^2 l + \log\left(\frac{1}{\eta}\right)\right)$$

Furthermore, with probability at least $1-\eta$, every classifier $sign(f) \in sign(\mathcal{F})$ has error no more than

$$\frac{b}{l} + \sqrt{\frac{c}{l}\left(\frac{R^2}{\gamma^2}\log^2 l + \log\left(\frac{1}{\eta}\right)\right)},$$

where b is the number of labelled training examples with margin less than γ.

Proof: (Shawe-Taylor et al., 1998, Theorem 5.5) □

By lemma 4.7 at page 37 we can replace te factor $1/\gamma^2$ by the 1-Norm $\|\alpha^*\|_1$ of the solution coefficents of the SVM dual optimization problem.

This bound even holds if the VC dimension of SVM is infinite. Theorem 7.8 provides a distribution free bound on the generalization error of a decision function drawn from a space of infinite VC dimension. So it can be used to explain how a classifier can give better generalization than would be predicted by the analysis of its VC dimension.

7.4 Do SVM implement SRM?

Do Support Vector Machines implement the Structural Risk Minimization principle? Attentive readers of Vapniks *Statistical Learning Theory* will recall that theorem 7.8 bounds the fat-shattering dimension by the same quantity Vapnik uses for the VC dimension (Vapnik, 1998, p413).

By this bound a structure can be defined on the hypothesis space $S = \{f(x,\alpha) : \alpha \in \Lambda\}$ by a sequence of increasing margins, where we would define an element of this structure to be the space

$$S_\gamma = \{f(x,\alpha) : \; f \text{ has margin at least } \gamma \text{ on the training set}\},$$

which corresponds to a structure of nested subsets with decreasing VC dimension. This seems to satisfy the demands of the SRM principle as they are stated in definition 1.2 at page 16: to perform SRM we should use a function $f(x,\alpha)$ from a hypothesis space S_γ such that the sum of empirical risk and VC dimension becomes minimal.

The problem with this approach is, that we do not know to which space a particular hyperplane belongs before we measure its margin on the training data. But this violates one of the assumptions in proving the SRM theorems (Vapnik, 1998, chapter 6): the sequence of hypothesis spaces must be chosen in advance. So Vapniks approach fails because the defined hierarchy depends on the data.

The methods of structural risk minimization can be generalized to encompass decomposition made *after* the data has been seen. The main tool is the fat-shattering dimension. Shawe-Taylor et al. (1998) show that if a classifier correctly classifies a training set with a large margin, and if its fat-shattering function at a scale related to this margin is small, then the generalization error will be small.

Part III

SVM in Bioinformatics

Chapter 8

Biological Background

Although the subject of Bioinformatics is not well defined, the name is usually used synonymous with computational molecular biology. This shows that the whole field of biological applications is restricted to one special branch. Bioinformatical topics range from database-searching and clustering of similar DNA or protein sequences to the statistical analysis of microarray data.

To understand what problems we are faced with, we need a basic understanding of the biological background. In this chapter we review some fundamental biological ideas from a rather formal point of view. This will be sufficient for our mathematical purpose, but is of course overly simplified from a biological point of view.

8.1 The Central Dogma

The main characters in molecular biology are DNA, RNA and proteins, which are related by what is called the *Central Dogma*: "DNA makes RNA makes protein". We will now expand the biological processes summarized in this short principle in the case of eucaryotic cells.

8.1.1 DNA

Nucleotides. *Deoxyribonucleic acid* (DNA) is a macromolecule contained in the nucleus of every organism's cells. The small building blocks of this molecule are called *nucleotides*. They can be further divided into three components: a phosphoryl group, a sugar deoxyribose, and one of the four bases *adenine* (`A`), *guanine* (`G`), *cytosine* (`C`) or *thymine* (`T`). All nucleotides share the first two components but differ with respect to the bases. Hence, there are four types of nucleotides which are named after the particular base they contain.

Double helix. Binding the phosphoryl group of one nucleotide to the sugar of a second one builds a directional linear polymer, a so called *single-strand polynucleotide*. Two of these nucleotide chains are connected by hydrogen bonds between pairs of bases and form the famous *Double Helix*. This structure was first shown by Rosalind Franklin and Maurice Wilkins using X-ray diffraction in 1953 and is the central component of the DNA model created by Watson and Crick.

Formally, the sequence of nucleotides in a DNA strand can be defined as a word (i.e. a finite sequence) over the four-letter alphabet $\Sigma_{\mathtt{DNA}} = \{\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{T}\}$.

Base pairing. Not all pairings between two different bases are possible. `A` always pairs with `T` and `G` with `C`. Thus, one strand of the DNA is always complementary to the other: if one nucleotid sequence is given, and we exchange `A` with `T` and `G` with `C` we gain the sequence of the pairing strand. Base pairing is a fundamental concept in molecular biology, because it allows to express the genetic information encoded in the DNA sequence.

Coding regions. Not the whole genomic DNA encodes proteins, some regions code for different kinds of RNA. Large parts of the DNA sequence even seem to be free of valuable information. The functional units that code for RNA or proteins are called *genes*. A gene is usually found on a single contiguous segment of DNA. Some parts called *introns* are removed in the following step. The other parts that actually encode information are called *exons*.

8.1.2 RNA

Differences to DNA. Like DNA, *ribonucleic acid* (RNA) is a polynucleotide. But there are some differences between both molecules. The DNA base thymine (`T`) is replaced by uracil (`U`), so the alphabet becomes $\Sigma_{\mathtt{RNA}} = \{\mathtt{A},\ \mathtt{C},\ \mathtt{G},\ \mathtt{U}\}$. The sugar group of the nucleotides is a ribose and not a deoxyribose as in DNA. RNA does not form a double helix, but remains a single strand polynucleotide.

Transcription. There are different types of RNA that take part in the transition of DNA to protein. We will describe *messenger RNA* (mRNA) and later *transfer RNA* (tRNA). mRNA is a complementary copy of a subsequence from the DNA template strand. This process is called *transcription*. Formally, it can be written as a bijection between the DNA and RNA sequence: $\mathtt{A} \mapsto \mathtt{U}$, $\mathtt{C} \mapsto \mathtt{G}$, $\mathtt{G} \mapsto \mathtt{C}$, $\mathtt{T} \mapsto \mathtt{A}$. Thus, the RNA sequence matches the DNA sequence of the *non*-template strand except for the replacement of `T` by `U`.

Splicing. After the transripition step, RNA contains both introns and extrons, but only the later carry the information needed in protein synthesis. In a process called *splicing* the introns are removed from the transcribed product. What remains are concatenated exons.

8.1.3 Protein

Structure. Like DNA and RNA, proteins are directional linear polymers. They are made of 20 different building blocks called *amino acids*. Since the amino acids are linked by peptid-bounds, the whole polymer is called a *polypeptid*. We can view a protein of length q as a word $\mathcal{P}$ over the 20-letter alphabet of amino acids $\Sigma_{\texttt{Prot}} = \{A_1, \ldots, A_{20}\}$:

$$\mathcal{P} \;=\; (s_1, \ldots, s_q) \qquad s_i \in \Sigma_{\texttt{Prot}}$$

How can the information contained in the 4-letter mRNA sequence be expressed as a protein sequence? The answer is the *genetic code*.

Genetic Code. The letters of the RNA sequence are put together in triplets, so called *codons*. Each codon (and therefore also each DNA triplet) encodes one amino acid or a stop signal denoting the end ot the translation process. Because there are $4^3 = 64$ possible triplets in RNA but only 20 letters in protein, there is usually more than one codon that encodes the same amino acid. While the biological details are quite complex, we can view the genetic code as a map $c : (\Sigma_{\texttt{RNA}})^3 \to \Sigma_{\texttt{Prot}} \cup \{\texttt{STOP}\}$.

Translation. In a cellular structure called *ribosomes*, the mRNA molecules are converted into proteins. This translation from codons to amino acids is performed by *transfer RNA* (tRNA).

Proteins are not sufficiently described by just stating their amino acid sequence. An important feature of protein function is the folding structure of the molecule. This is the reverse formulation of the central dogma: The function of a protein depends on its structure which is determined by the amino acid sequence which is a translation product of the mRNA copied from the DNA.

8.2 Protein Folding

Protein Function. Proteins carry out almost all of the functions of the living cell. Neither growth nor development would be possible without proteins. Each of the huge number of protein functions demands its own protein structure, because interaction with other molecules depends on a specific three-dimensional build.

Four different levels of protein structure can be distinguished:

1. **Primary.** The primary structure is the amino acid sequence itself. It contains all the information necessary to specify higher-level structures.

2. **Secondary.** Some regions of the protein are ordered in repeating patterns. Two patterns are common to many proteins: the α-helix and the β-sheet. In drawings of protein structure they are represented as helices and arrows (see figure 9.6 at page 97, or figure 9.7 at page 98). Collectively these repeating patterns are

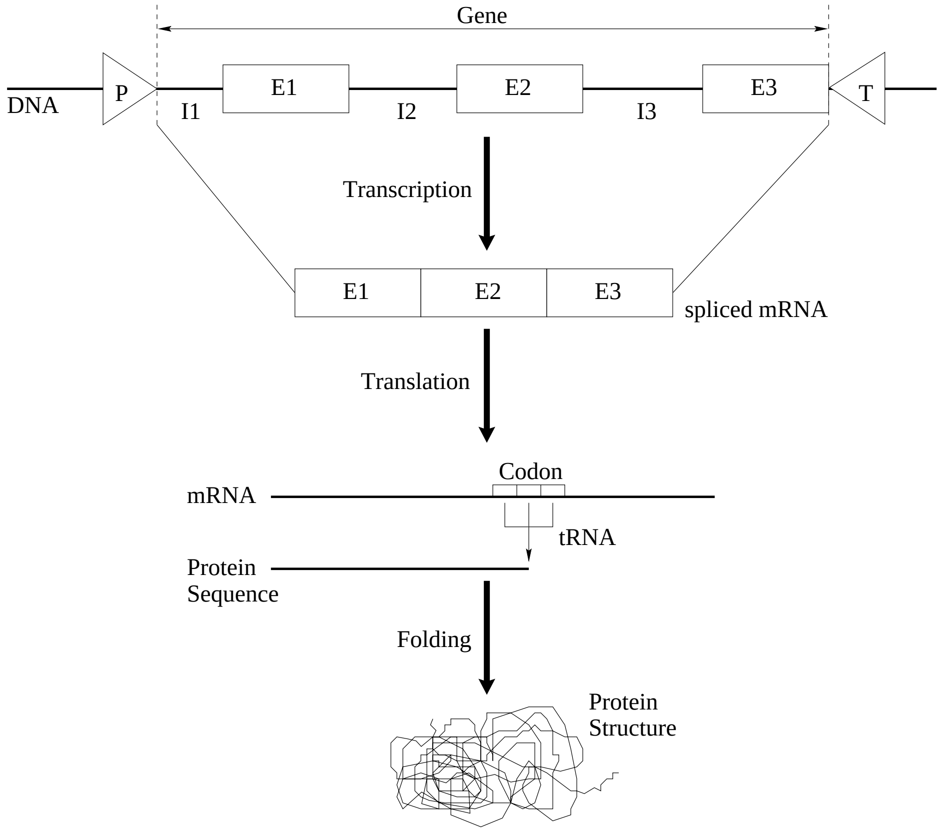

Figure 8.1: **From DNA to Protein:** Transcription is initiated at special sites called promotors (P) and stopped at terminators (T). Introns (I1,I2,I3) are spliced out, and only the exons (E1,E2,E3) are retained for further use in protein synthesis. In the next step, tRNA translates each codon of the mRNA into an amino acid. The amino acid sequence determines the folding structure of the protein.

known as secondary structure. They do not encompass the whole protein, some regions lack any ordered patterns. The whole polypeptide is a series of elements of secondary structure and less ordered sectors.

3. **Tertiary.** The three-dimensional arrangement of the amino acids is unique to each protein. It is built by components of secondary structure. This complex is called the tertiary structure.

4. **Quaternary.** Many proteins associate to form multiple molecular structures that are held together tightly by the same forces that form the tertiary structure. The three-dimensional arrangement of protein units as they fit together to form a multimolecular structure is called the quaternary structure.

The folding of a protein sequence cannot be random. Even for small proteins it would take billions of years to try all of the possible structural conformations available. But folding is a *rapid* process, it often happens on a time scale of seconds. Folding is *robust*, because proteins reach their functional states under a wide regime of chemical conditions. It is *reproducible*, as native proteins may resemble each other closely on an atomic scale, and it is *redundant*, since a large number of sequences map into the same region of structure space. This tells us that there must be a folding pathway and the process is not random. How protein structure actually is determined by the amino acid sequence is still the subject of intensive biological research.

Experimental identification. How can protein structure be identified? Experimentally it is usually done by *X-ray diffraction*: Crystallized protein is shot with X-rays, which are scattered by the atoms of the crystals giving rise to a diffraction pattern. From the map of these patterns details of the structure can be inferred. A second method is called *Nuclear magnetic resonance spectroscopy* (NMR). It monitors the absorption of energy associated with transitions of nuclei between adjacent nuclear magnetic spin levels. After identification of individual nuclei, distances between neighboring nuclei are estimated. In contrast to X-ray diffraction, NMR is limited to small molecules.

Theoretical modeling. The folding states of RNA molecules can be successfully modeled by thermodynamical methods. The problem of inferring the folded structure from a given sequence is then reduced to a minimization problem: Find the structure minimizing the potential-energy functions. But these techniques do not work satisfactory on protein sequences. Here the problem is much harder and lacks a good physical model. The prediction of protein structure and thus detailed molecular function from sequence data alone remains one of the great challenges in computational biology.

Machine Learning. This is where the theory of learning from examples comes in. If we are not able to conveniently model the problem directly, why not try to learn prediction rules from a set of given examples? The most simple situation is this, where we know a number of different folding classes and are asked to give a rule which distributes new sequences into these classes. We do not model *how* the protein folds into a specific structure, and we can not discover any *new* folding classes.

We will consider this approach in the next chapter. First, we describe a database of protein sequences with already identified structure, and then we discuss the performance of several statistical classification methods based on training and test data from this database.

Chapter 9

Protein Fold Class Prediction

As we have seen in the last chapter, knowledge of the 3D structure of a protein is essential for describing and understanding its function. Today there is a large number of sequenced proteins facing a much smaller number of proteins whose structure has been determined. To bridge this gap, methods of statistical classification can be used for the prediction of protein fold classes based on a given amino acid sequence.

The problem we will solve is this: given a number of different protein fold classes, find a decision function which assigns a protein sequence to the right class.

In this chapter we will first describe a database of expected fold-classes from which training and test data are taken. Several statistical methods for discrimination and prediction are introduced in the second section. The main part of this chapter deals with our own work on prediction by Support Vector Machines and the comparison of the achieved results to those obtained by the competing methods.

9.1 The Dataset

For training and testing we take a data set provided by Janet Grassmann et al., in the study *Protein Fold Class Prediction: New Methods of Statistical Classification* (Grassmann et al., 1999). It stems from the *Database for Expected Fold-classes* (DEF) by Reczko and Bohr (1994).

Fold Classes. In the DEF a sequence of amino acids is assigned a specific overall fold-class and a super fold-class with respect to secondary structure content. The four super classes are obtained by characterizing the protein structure by the presence or absence of *alpha*-helices and *beta*-sheets, as is shown in table 9.1.

If the three-dimensional structure of a protein is known, it can be grouped into tertiary structural classes. According to topological similarities of the backbone a set of 38 fold

α	Only α-helical regions.
β	Only β-sheets.
α / β	α-helices and β-sheets alternating.
$\alpha + \beta$	α-helices and β-sheets separated in distinct domains.

Table 9.1: Definition of the four classes of SSC.

classes was defined (Pascarella and Argos, 1992) and later enlarged to 42 classes of tertiary structure (Reczko and Bohr, 1994). The 4-class distinction will be called *super secondary classification* (SSC), and the 42 classes will be denoted as 42-CAT. The relations between SSC and 42-CAT are listed in the appendix.

Embedding. Working in the space of amino acid sequences directly is complicated by the fact that in our case sequence lengths vary roughly between 50-300 amino acids. Therefore, we have to transform each protein to a vector in a space more suitable for statistical classification. The most simple transformation is to map the sequence $\mathcal{P}$ of length q to the vector x of its amino acid frequencies in $\mathbb{R}^{20}$:

$$x_i \;=\; \#\{s_k = A_i : \; k = 1, \ldots, q\} \qquad i = 1, \ldots, 20$$

This representation lacks any information about the neighborhood relationships of amino acids. But of course certain conformations of the amino acid chain can only occur if the size and the chemical properties of *neighboring components* fit together. So, the next step is to exploit the information contained in m-tuples of amino acids ($m \geq 2$). To keep the space of embedded sequences at a reasonable low dimension, we will only use the frequencies of pairs of neighboring amino acids, the so called *dipeptide frequency* ($m = 2$). This results in a 20×20 matrix of dipeptide frequencies

$$x_{ij} \;=\; \#\{(s_k, s_{k+1}) = (A_i, A_j) : \; k = 1, \ldots q-1\} \qquad i, j = 1, \ldots, 20,$$

which can be written as a vector in $\mathbb{R}^{400}$. To make the embedding independent of the sequence length, we normalize such that

$$\sum_{i=1}^{20} x_i = 1 \qquad \text{and} \qquad \sum_{i,j=1}^{20} x_{ij} = 1, \tag{9.1}$$

thus achieving *relative* frequencies. This way of embedding has several advantages. All proteins are transformed into one input pattern of fixed size. Insertions and deletions from the protein sequence cause anly small changes in the dipeptide frequencies. The same holds true for rearrangments of larger elements in the sequence. There are many cases where members of the same fold class differ mostly by permutations of sequence elements. This yields very similar dipeptide matrices and thus supports similar classification results.

In the next two sections we will only consider embedding by dipeptide-frequencies. Other ways to represent the data are dealt with in section 9.4.

Test and Training Set. From the DEF a set of 268 sequences and their classification into 42-CAT and SSC are used, divided into a training set of 143 and a test set of 125 sequences. This division was done randomly, balanced with respect to the prior probabilities of the 42 classes. Typical for protein sequence data, there is no uniform distribution of the classes. Figure 9.1 shows the distribution of the 42 classes of 42-CAT and the 4 classes of SSC in the training and test set.

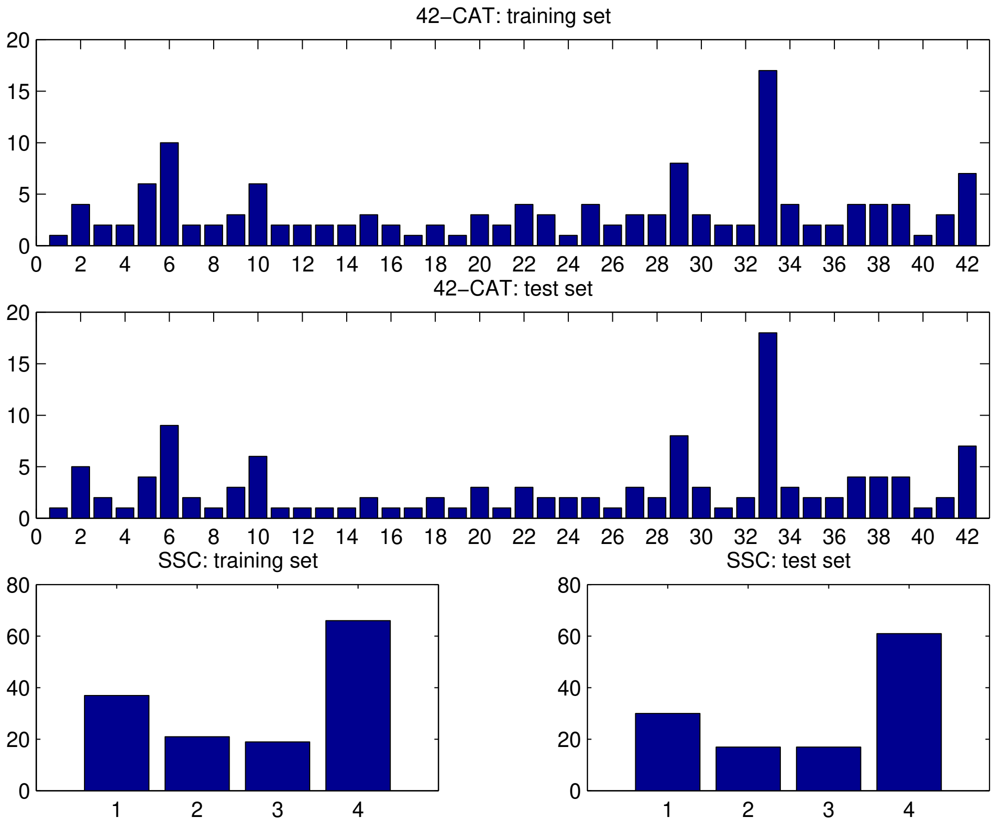

Figure 9.1: Distribution of the 42 classes of 42-CAT (4 classes of SSC) in the training and test set. Obviously, the sequences are not uniformly distributed over the classes. Most classes contain less than five sequences.

9.2 Results by Competitors

In (Grassmann et al., 1999) several statistical methods are used on the training data: Feed Forward Neural Networks (NN), Additive Model BRUTO, Projection Pursuit Regression (PPR), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Flexible Discriminant Analysis (FDA) and the k-Nearest-Neighbor Rule (kNN). In Appendix B we give a short description of each of these classification methods.

We compare the different statistical classification methods in three ways: the error on the training set (apparent prediction error, APE), the error on the test set (test prediction error, TPE) and the ten-fold cross-validation error (CV(10)). The error on the training set is the first naive estimate of the prediction error and is usually overoptimistic. A more reliable estimate is the performance of the classifier on a separate test-set. The 10-fold cross-validation error is obtained by first pooling test and training data and then randomly splitting it into 10 disjoint subsets. Each of these sets in turn is used as a test set for a classifier trained on the remaining 9 subsets of the whole data set. At the end of the cross validation procedure each element of the pooled dataset is assigned to a predicted class.

42-CAT. The results of Grassmann et al. are summarized in table 9.2. For Neural Networks only a subset of models including the best results is shown by selecting among the number of hidden layers (0, 5 and 9). In the original paper PPR is reported for 1, 4 and 6 additive terms, but since they all display totally disappointing performance we show a lower bound for all three models in the last column. The additive model BRUTO is not shown at all, because Grassmann et al. claim that its results were 'definitely worse than others' and thus do not report it themselves.

Error (%)	kNN	LDA	QDA	NN(0)	NN(5)	NN(9)	PPR(·)
APE	0	0	0	9.8	11.2	2.1	>50
TPE	33.6	34.4	83.2	39.2	36.8	28.8	>50
CV(10)	34.0	30.2	38.4	36.9	38.8	32.5	>50

Table 9.2: Competing results [42-CAT]: k-Nearest-Neighbor, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Neural Networks and Projection Pursuit Regression.

Error (%)	kNN	LDA	QDA	BRUTO	NN(0)	NN(10)	PPR(2)	PPR(10)
APE	0	7	0	39.2	13.3	0	13.3	0
TPE	26.4	15.2	65.6	33.6	21.6	23.2	28.8	28.0
CV(10)	23.5	26.9	37.3	35.4	28	22.4	34.3	32.8

Table 9.3: Competing results [SSC]: k-Nearest-Neighbor, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Additive Model BRUTO, Neural Networks and Projection Pursuit Regression.

SSC. Table 9.3 shows the results for the classification into the four SSC. The more flexible nonlinear procedures as PPR now give reasonable low error rates. However, they are still higher than those obtained with Neural Networks. NN(10) was best among all NN(·), just beating the k-Nearest-Neighbor classifier.

The best results are 28.8% test error by a Neural Network with 9 hidden layers and 30.2% cross-validation error by Linear Discriminant Analysis in 42-CAT and in SSC $\sim$15% test error by LDA and $\sim$22% CV(10) error by NN(10).

In the next section we will describe how Support Vector Machines can be used to do classification on the same protein dataset. We will show, that SVM perform better than all competing methods.

9.3 Prediction by SVM

9.3.1 Software

Detailed information of all available SVM software can be found in the software section of `http://www.kernel-machines.org/`.

We implemented Support Vector Machines in MATLAB by using parts of Steve Gunn's MATLAB SVM Toolbox (Gunn, 1998) which contains many functions for SV classification and SV regression that can be used via a graphical user interface. From all these functions we only used the basic routine `svc.m` (which needs `svkernel.m` and `svtol.m`) and modified it in various ways.

1. We changed the quadratic optimiser from the old `qp` to the newer `quadprog`.
2. Gunn's functions only implement the 1-Norm Soft Margin SVM, so we extended it to cover the 2-Norm case (see section 4.2.1 and 4.2.2).
3. In the original toolbox one fixed C parameter is used for both classes. We adapted it to handle different error weights for each of the two classes. This is important in imbalanced datasets (see section 4.3).
4. From these 2-class functions we built a n-class classifier using the methods *one-versus-rest* and *one-versus-one* of chapter 6.

9.3.2 Kernel Selection

To avoid confusion between the different classification tasks, we will first describe in detail the steps taken in the classification of 42-CAT. The adjustments necessary to SSC will be described in section 9.3.5.

The standard kernels used in classification are *polynomial* kernels and *radial basis function* kernels, i.e.

$$k_{poly}(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^d \quad \text{and} \quad k_{rbf}(x_1, x_2) = \exp\left(\frac{-\|x_1 - x_2\|^2}{2\sigma^2}\right).$$

Once we have decided which kernel to use, we have to specify the kernel parameters (degree of polynomial d, width σ) and the error weight C.

Kernel parameters. As we can see in table 9.1, the linear model LDA performs very well compared to the other more complex models. This led us to expect good results with polynomials of low degree, even if the problem of learning 42 classes from 143 examples looks quite difficult. Thus, we constructed polynomial SVM of degree 1, 2 and 3 (called *poly1-SVM*, *poly2-SVM* and *poly3-SVM*).

For radial basis kernels the problem of choosing a sensible width σ is not so straightforward. In the case of *rbf-SVM* the decision function (5.17) at page 55 reads

$$f(x) \;=\; sign\left(\sum_{i=1}^{\#SV} \alpha_i y_i \exp\left(-\frac{\|x - x_i^{SV}\|^2}{2\sigma^2}\right) + b^*\right)$$

Essentially, this is a sum of gaussian hats of width σ with centers at the support vectors. For this function to be a reliable representation of the data we have to choose σ adequate to the distances between the datapoints. To achieve this aim, σ is set equal to the median of the Euclidean distance from each positive example to the nearest negative example. This idea was first introduced by Jaakkola et al. (2000). Roughly speaking, this guarantees that, on average, the (approximate) distance from a protein sequence in the class of interest to its nearest non-member is one.

In our setting this results in the choice of $\sigma = 0.106$. The Support Vector Machine with radial basis function kernel of width 0.106 will be called *rbf-SVM*. (We chose σ from the data in the training set. Repeating the calculation for the combined training and test data, we found $\sigma = 0.1024$, which is not far from our choice and, as we will later see in figure 9.2, achieves the same results.)

The error weight C. While the choice of rbf width is at least guided by a heuristic, no theoretical hint is available on how to choose the error weight C. In most papers it is called 'done by cross-validation', which amounts to trying through a whole range of possible candidates and choosing the one with the best result.

We face a further problem in the use of *one-versus-all* as n-class prediction method. In each step we separate one class from all the other classes. In our case we separate the proteins belonging to one class from the whole of proteins belonging to the other 41 classes. With only 143 training examples distributed among 42 classes this is, in the mean, equivalent to separating 3 examples from 140. Thus, we encounter a very imbalanced training set in each step of *one-versus-all*. We combat this by assigning different error weights to the class of interest (+) and to the rest (–):

$$C_+ \;=\; \lambda\,\frac{n_+ + n_-}{n_+} \quad \text{and} \quad C_- \;=\; \lambda\,\frac{n_+ + n_-}{n_-},$$

where n_+ (n_-) is the number of elements in the positive (negative) class and λ is a scaling parameter. λ determines the order of magnitude while $\frac{n_+ + n_-}{n_+}$ and $\frac{n_+ + n_-}{n_-}$

distribute the weight. To choose a successfull λ we vary it through a wide range of possible values and take the one providing the lowest test error.

In our experiments a choice of $\lambda = 2000$ attained minimum error. This high value seems reasonable if we recall that linear methods perform very well. The dataset seems to be almost linear separable, so we should allow training errors but avoid inviting them by a low C parameter.

Influence of C: The error weight only enters the optimization problem (4.21, p41) if there are x_i with positive slack variable ξ_i. It has no influence on the prediction, if all training examples are classified correctly and lie outside the margin or on the margin boundary (recall figure 4.2 at page 41), and is only important for the points which are misclassified or violate the margin. How many points are this? The number of misclassified examples (the training error) can be found by taking the training set as a test set, but the number of margin violations is not so easily discovered. In the case of 1-Norm Soft Margin, an upper bound on the number of both misclassifications and margin violations can be attained by the Kuhn-Tucker complementarity condition (4.35), i.e. $\xi_i(\alpha_i - C) = 0$ for all i. From this follows that non-zero slack variables can only occur if $\alpha_i = C$ and the number of misclassifications and margin violations is upper bounded by

$$\#\{\xi_i > 0 : \ i = 1, \ldots, l\} \ \leq \ \#\{\alpha_i = C : \ i = 1, \ldots, l\}$$

(It is only an upper bound and not the correct number, because it can happen that $\xi_i = 0$ and at the same time $\alpha_i = C$.) Of course, we can enforce a high number of misclassifications by setting C to a very low value or even forbid any training errors by $C = \infty$. We consider here C chosen such, that the error rates observed on the test set are minimized. Then, the above bound gives an impression of how well the separating hyperplane fits the data.

In our example we see that rbf-SVM achieves zero number of $\alpha_i = C$ for almost all binary separations of the training set of 143 examples and while doing cross-validation. In the few cases where we observed α_i at the upper bound, the total number was never above 10% and mostly below 5%. This indicates that different choices of C will only lead to minor changes in the classification outcome. We confirmed this by trying through a wide range of C parameters and always observing very similiar performance. This behaviour confirms that radial basis kernels with small enough width make any training set linearly separable.

In polynomial SVM we observed a larger number of $\alpha_i = C$. On the training set of 143 examples we found, in the mean, 75.8% for poly1-SVM and for the second and third degree polynomial SVM even 100%! Since $\alpha_i = C$ implies $\alpha_i > 0$, we see that *all* training examples will be support vectors for *all* separations of one class from the rest by poly2-SVM and poly3-SVM. This shows that polynomials are not so flexible as radial basis functions and, more important, that the linear SVM fits the data better than the other two polynomials. These observations could be made in the training phase, before we obtained the test results.

9.3.3 Prediction Error

In table 9.4 we summarize the results of support vector classification using the kernels and parameters described above. The first observations is that indeed the linear poly1-SVM show excellent performance. With 28.8% test error and 29.1% CV error they overcome the best competitors and are only surpassed by rbf-SVM.

Error (%)	rbf	poly1	poly2	poly3
APE	0	0	4.2	1.4
TPE	23.2	28.8	32	32.8
CV(10)	30.2	29.1	35	34.2

Table 9.4: Support Vector Machine results [42-CAT]

APE. The zero training error of rbf-SVM was expected from the observed number of α_i at the upper bound C, but the low values of the three polynomial SVM may look surprising at first sight. This is no contradiction to the observations above. The high number of $\alpha_i = C$ for polynomial SVM occured in the 2-class separations, which are only an intermediate step in the n-class classification. Even if all SVM separating one class form the rest are loaded with misclassifications and margin violations, this does not imply that the resulting n-class classifier performs poorly. (On the other hand, even if all $\alpha_i < C$, we do not know for sure that no training errors will happen.) Thus, the low value of APE shows that the borders obtained by *one-versus-all* are a good representation of the 42 classes in the training set, even if the components are only crude descriptions of the data.

TPE. Both rbf-SVM and poly1-SVM achieve better results than the competing methods. rbf-SVM attains the lowest test error of all used methods (23.2%) and poly1-SVM is as good as NN(9) in classifying the data (28.8%), but of course has much lower capacity. poly2-SVM and poly3-SVM exhibit only slight differences on the test set (32% and 32.8%).

CV(10). For the polynomial SVM there are no great differences between the performance on the test set and the behaviour in 10-fold cross validation. TPE seems to be a reliable estimate of generalization ability. rbf-SVM shows a larger gap between TPE and CV(10). The low test error of 23.2% has to be attributed to the choice of the test set and not to the generalization ability of the Support Vector Machine.

How good is the choice of kernel width σ? To answer this question we constructed figure 9.2. Different values of kernel width σ are plotted against the TPE achieved by rbf-SVM using this σ. The steep increase on the left ($\sigma \to 0$) shows overfitting, because decreasing σ results in increasing capacity. The line indicates our choice of $\sigma = 0.106$. The curve runs almost constantly at TPE = 23.2% until $\sigma = 1.5$, where a sharp ascent begins. The width choosing heuristic described above leads us into a 'valley of test error'. Varying σ from 0.01 to 0.4 does not result in big jumps or a dramatic increase.

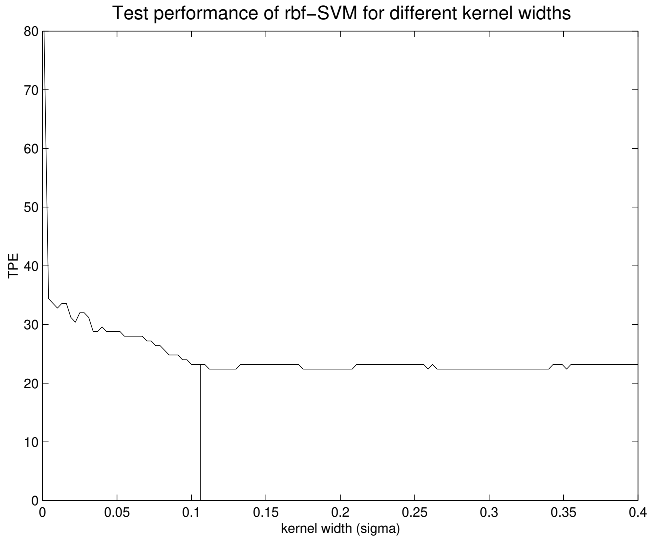

Figure 9.2: Performance of rbf-SVM on the test set with different kernel widths σ (stepwidth 0.003). The line indicates our choice of $\sigma = 0.106$.

Polynomials of higher degree. In addition to the polynomials of degree 1 to 3 reported above, we tried as well polynomials of up to tenth degree. We could observe no remarkable increase in performance. The error on the training set (APE) is constantly 0.7%, i.e. one sequence is misclassified. The test error (TPE) varies between 31.2% and 32.8% and the CV(10) error was never less than 32.1%. SVM with polynomial kernels of degree 4 to 10 seem to perform slightly better than poly2-SVM and poly3-SVM, but do not challenge the success of poly1-SVM.

Are always the same sequences misclassified? To compare the output of the four Support Vector Machines we study the number of common misclassifications. The three polynomial SVM have more than 50% misclassifications in common and share nearly all of these with rbf-SVM. For a more detailed view we constructed table 9.5 of pairwise common misclassifications. On the main diagonal the number of test errors of each SVM is shown, while the off-diagonal elements present the number of test errors common to different SVM.

	rbf	poly1	poly2	poly3
rbf	29	28	22	22
poly1		36	22	24
poly2			40	39
poly3				41

Table 9.5: Pairwise common test errors [42-CAT]

We see that poly3-SVM and poly2-SVM fail on an almost identical set of test points. With one exception, all errors made by rbf-SVM are also done by poly1-SVM. The high number of common test errors shows, that misclassifications can not purely be attributed to the particularities of the different Support Vector Machines, but can mainly be explained by the difficult test examples.

Confusion Matrices. We are not only interested in the number of misclassifications, but also in the pattern of classification errors. Are all errors uniformly distributed among the 42 classes, or are some classes more likely to be falsely predicted than others? This question is answered by the construction of a *Confusion Matrix.* This is a matrix $M = (m_{ij})$ with entries

$$m_{ij} \;=\; \#\{x \in \mathcal{T} : \; x \text{ is predicted as class } i \text{ and truely belongs to class } j\},$$

where $\mathcal{T}$ is the test set. We built confusion matrices for poly1-SVM and rbf-SVM based on the output of CV(10). Figure 9.3 shows the result for rbf-SVM and figure 9.4 for poly1-SVM.

The most striking feature of both matrices is the high frequency of misclassifications into class 22 and class 29. Both Support Vector Machines have almost the same CV(10) error rate, but poly1-SVM shows a much more concentrated pattern than rbf-SVM. Here, the errors are scattered over the whole matrix, while the first degree polynomial SVM has only isolated misclassifications outside the columns 22 and 29.

We compared our results to the Confusion Matrices for NN(9) and LDA given in (Grassmann et al., 1999). Here, both matrices look equally scattered, the predictions of the linear method are not more concentrated than the outcome of the flexible Neural Network. In NN(9) the columns 22, 29 and 33 are largely filled, while LDA predicts most errors into class 22 and 29. This shows that the high number of misclassifications into class 22 and 29 is no attribute only of Support Vector Machines, but of other methods as well. In section 9.5 we will lay out strong biological evidence, that in fact we do not observe an artifact of mathematical methods but a feature inherent in the structural build of the proteins belonging to these classes.

In some positions of the main diagonal we see no entries, i.e. no member of this class could be correctly classified. This only happens in classes containing two or three sequences. We made sure that in the crossvalidation procedure these few examples did not all fall into the same partition. Since 2 or 3 misclassifications happen in almost all classes, we conclude that the difficulty lies only in the small size of the classes and not in the properties of the according proteins.

9.3.4 Support Vectors

One of the main advantages of SVM over Neural Networks is its transparency. All Neural Networks lack an interpretation of the weights achieved in the optimization process and the solution is not unique, because quite different weights and weight

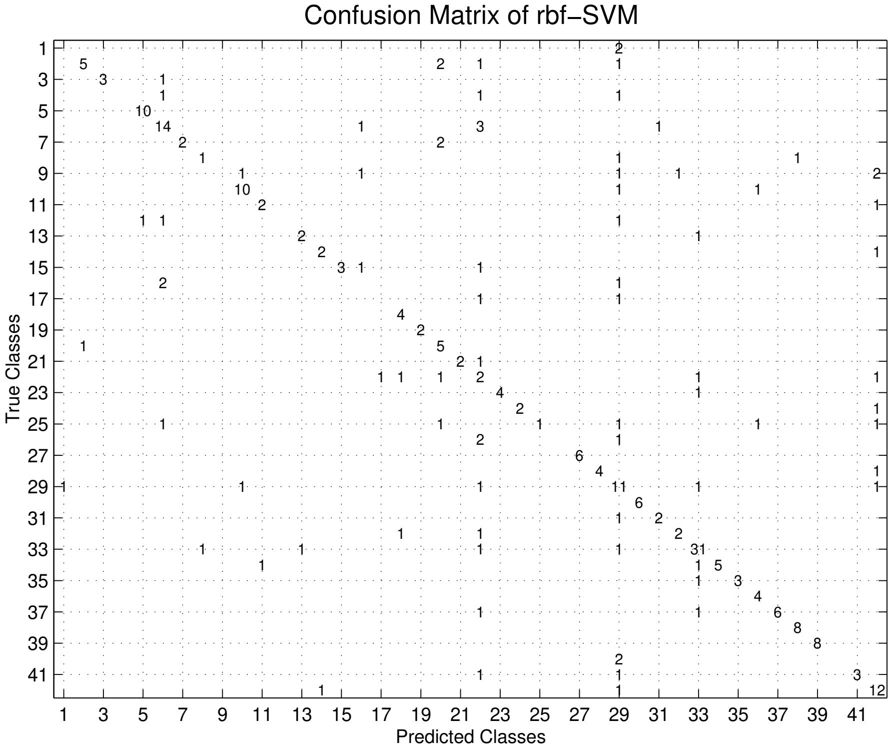

Figure 9.3: Confusion Matrix of rbf-SVM [42-CAT]. The lines represent the true fold classes according to Reczko and Bohr (1994) and the columns represent the fold classes predicted by rbf-SVM ($\sigma = 0.106$). Note the high number of false positives in class 22 (15 sequences) and class 29 (18 sequences). The two classes attract misclassifications.

patterns can lead to the same prediction outcome. Because of this Neural Networks are often called *black box* methods. The situation in Support Vector Machines is different. Even if the implicit mapping defined by the kernel function brings a black box moment into the whole method, we are able to interpret the outcome in terms of the support vectors. In the case of polynomial kernels, we are even able to make the kernel mapping explicit, as was shown in the proof of theorem 7.5 at page 68. But of course the point in Support Vector Machines is, that we don't *need* this direct grasp to the feature space.

Number of SV. Support Vector Machines constitute a data-reduction procedure, because separable datasets can be correctly classified if only a subset of all datapoints, the set of support vectors, is known. In our problem we expect this reduction not to be very effective, i.e. the number of support vectors to be very high, since with 143 examples distributed among 42 classes almost all datapoints will be support vectors.

Because of the construction of *one-versus-all*, we can only access the 2-class SVM

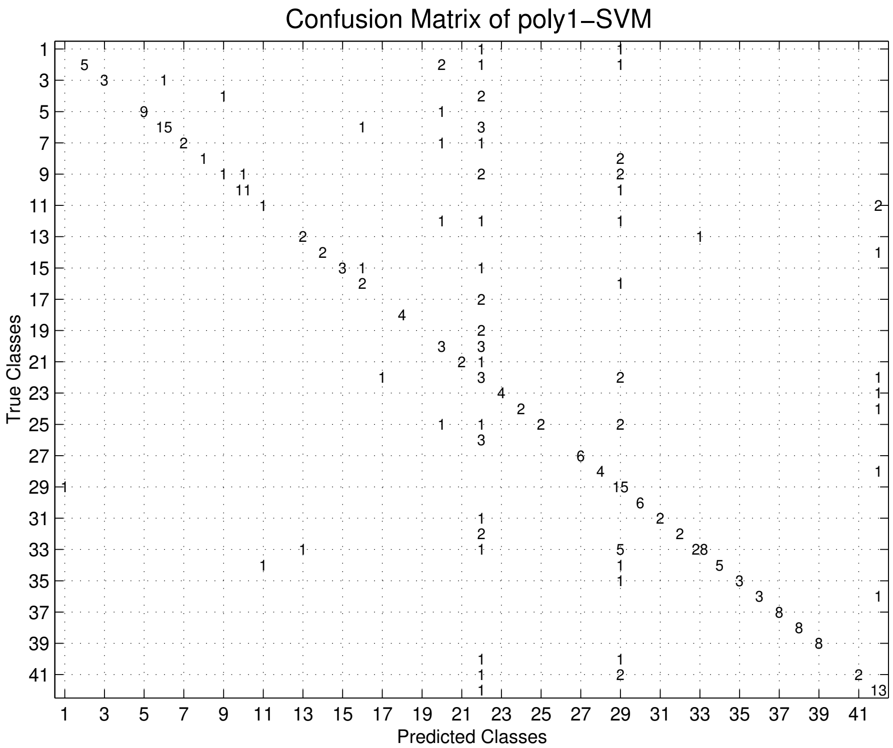

Figure 9.4: Confusion Matrix of poly1-SVM [42-CAT]. The pattern of classification errors is less scattered than in figure 9.3. Again notice the numerous misclassifications into class 22 (31 sequences) and class 29 (23 sequences).

directly. We state the number of support vectors as the percentage of the total number of training points. poly2-SVM and poly3-SVM reach 100% in all 2-class separations. poly1-SVM needs less datapoints to do the classification: the median of the number of support vectors in the 42 2-class separations lies at 92% (mean 89%). The least number of training examples was needed by rbf-SVM: median and mean lie slightly above 43%.

Common Support Vectors. As an example we compare the behaviour of poly1-SVM and rbf-SVM in the separation of class 24 from the rest. Here, both Support Vector Machines achieve their minimum number of support vectors: poly1-SVM 64% (92 SV) and rbf-SVM 24% (34 SV). Both classifiers work in completely different feature spaces, but a large part (23 SV) of the radial basis support vectors can also be found in the polynomial machine.

9.3.5 Results for SSC

In figure 9.3 we see that the more flexible nonlinear procedures as PPR give reasonably low error rates. This led us to expect, that polynomial Support Vector Machines of a higher degree will be more successfull in classifying this data, than they were in the case of 42 classes. We show the results for polynomial kernels of degree 1 to 7.

Error (%)	rbf	poly1	poly2	poly3	poly4	poly5	poly6	poly7
APE	0	21	9.8	4.9	1.4	0	0	0
TPE	14.4	28.8	23.2	16.8	16	16	16	15.2
CV(10)	17.9	28.7	28	21.6	19.4	18.3	17.2	16.8

Table 9.6: Support Vector Machine results [SSC]

The width σ of the radial basis kernel was chosen by the same heuristic as described in 9.3.2. The median of inter-class distances is 0.097 both on the separated training set and on the complete pooled dataset. The Support Vector Machine using a radial basis kernel with $\sigma = 0.097$ will be called *rbf-SVM*. The error weight C again depends on the size of the classes, λ was set equal to 2000. This value is a good trade-off between allowing training errors and trying to avoid them.

Comparison to competitors. On the test set Support Vector Machines don't do much better than LDA (15% TPE), the TPE varies around 15% for both rbf-SVM and poly7-SVM. But we see a larger gap in the CV(10): 22% by NN(10) are clearly beaten by the results of rbf-SVM and poly7-SVM (17.9% and 16.8%). As in the classification of 42-CAT, SVM are superior to the other methods.

Confusion Matrices. We computed Confusion Matrices for selected Support Vector Machines (rbf-SVM, poly1-SVM, poly4-SVM and poly7-SVM) to show a cross-section of the studied kernels (figure 9.5). We don't see such a clear structure as in the 42-CAT confusion matrices in figure 9.3 and figure 9.4. But again we observe a very concentrated pattern in the linear poly1-SVM, in contrast to the more evenly distributed classification errors in the other SVM.

Higher degree polynomials. The error rates don't change much if we choose polynomial kernels of a higher degree. The cross validation error varies between 16.8% and 14.9% for polynomials of up to degree 60. We constructed polynomial kernels of such a high degree, because we were interested in witnessing the point, where overfitting the data becomes evident. Thus, we computed CV(10) starting with a polynomial kernel of degree 10 and going up in steps of 10 degrees to a kernel of degree 100. We observed only slight changes between poly10-SVM and poly60-SVM, but in poly70-SVM generalization performance broke down to a CV(10) of 60.4%. poly80-SVM was already at 99% and in poly90- and poly100-SVM we saw 100% error. This shows that Support Vector Machines are stable for a wide range of polynomial kernels, but then quickly collaps.

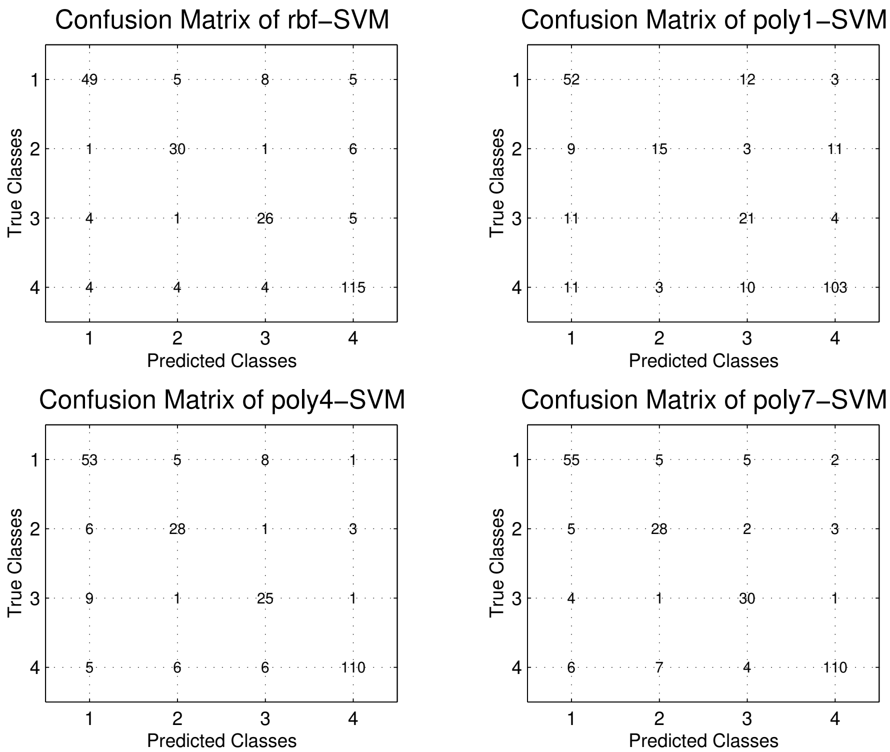

Figure 9.5: Confusion Matrices of four SVM classifying SSC: (i) radial basis function SVM, (ii) first degree polynomial, (iii) fourth degree polynomial and (iv) seventh degree polynomial SVM.

Why is poly1-SVM much more successfull in 42-CAT than in SSC? It is a striking feature in the comparison of the different Support Vector Machines, that the linear poly1-SVM performs so much better in the difficult 42 classes task than in the more simple 4 classes case. A reason for this is that the four classes of SSC are composed of the 42 classes of 42-CAT. Thus, each of the SSC classes is less homogeneous than the classes of 42-CAT. Flexible methods are better equipped than poly1-SVM to separate complex classes, even if it outperforms them in separating the homogeneous classes.

9.4 What do Learning Machines learn?

Which attributes of the sequences make it possible for Learning Machines to distinguish between them? To answer this question it is necessary to study *different embeddings* of the sequences, because all the information needed for the decision between different classes has to be drawn not from the sequences directly, but from their representatives in some vector space. In the last two sections we compared different classification methods for predicting the fold classes of proteins embedded by their dipeptid frequencies. The methods varied, while the embedding was fixed. Now we will turn the tables and vary the embedding, while holding on to SVM as the method of classification. The goal is to see, how much the generalisation performance depends on the embedding of the data. What information do SVM need to be good at classifying the sequences? How much does the outcome change, if we do the embedding by the frequency of single peptides ($m = 1$) or 3-tuples of amino acids ($m = 3$)?

A technical detail. In the computations above we used data from the DEF, because then all methods could be compared on exactly the same dataset. Since the data in the DEF is already embedded by dipeptide-frequencies, trying through a variety of different embeddings is made impossible. Thus, we got all sequences directly from PDB and embedded them on our own. For the case $m = 2$ (embedding by dipeptides) we observed only minor changes to the results obtained on the DEF data. These differences seem to be due to errors in rounding: our data is normalized such that the sum of all entries equals 1 (as shown in equation (9.1)), while in the DEF data this sum varies between 0.95 and 1.05.

Embedding by amino acid frequencies. The sequences are embedded in the $\mathbb{R}^{20}$, no neighborhood information is contained in this representation. So, we can expect the error rates to be much higher than those obtained with the data embedded by dipeptide-frequencies. We did a classification of the dataset into 42-CAT. The results are shown in table 9.7. Again, σ is chosen as the median of inter-class distances ($\sigma = 0.0846$) and

Error (%)	rbf	poly1	poly2	poly3
APE	0	18.2	38.5	51
TPE	30.4	40	57.6	58.4
CV(10)	29.5	46.6	52.2	59.3

Table 9.7: Support Vector Machine results [42-CAT] for sequences embedded by amino acid frequencies ($m = 1$). Compare this to table 9.4 at page 87.

λ is set equal to 2000. The results of poly4- to poly10-SVM differ only slightly from the results of poly3-SVM and thus are not shown. As we expected, the error rates are higher than before. This is very clear for polynomial Support Vector Machines, where we have a difference of more than 10% TPE for poly1-SVM and almost 25% TPE for poly2- and poly3-SVM. The number of errors on the training set has drastically increased. The zero training error and the small difference of only 7.2% TPE for rbf-SVM is surprising. The CV(10) error remains on the same level for rbf-SVM, but polynomial SVM show an increase of 15-25%.

The amino-acid distribution differs between α-helices and β-sheets. Our results show the generalization performance of decision rules built on this difference. Support Vector Machines with radial basis function kernels still show excellent performance, while a noticeable worsening can be observed in polynomial kernels.

Embedding by tripeptide frequencies. Now we move on to an embedding into a very high dimensional space. Embedding by the frequency of 3-tuples of amino acids leads to an input space of dimension $20^3 = 8000$. While this representation of the data carries much information about neighborhood relations, the high dimensionality enforces a breakdown in most statistical methods. Reduction procedures like Principal Component Analysis are needed to gain good performance of these methods. The results in table 9.8 show, that Support Vector Machines perform very well in very high dimensional spaces. No decrease in performance can be observed even in a space of 8000 dimensions. In $\mathbb{R}^{8000}$ the training set becomes linearly separable (zero training error for

Error (%)	rbf	poly1	poly2	poly3	poly4	poly5	poly6	poly7
APE	0	0	0	0	0	0	0	0
TPE	24	23.2	19.2	20	19.2	19.2	18.4	18.4
CV(10)	22.4	23.5	22.8	22.8	23.5	23.2	23.5	23.5

Table 9.8: Support Vector Machine results [42-CAT] for sequences embedded by tripeptide frequencies ($m = 3$).

all SVM) and the TPE slightly decreases with increasing complexity of the polynomials. The CV(10) always remains at the same level (somewhere around 23%) for all SVM. In this high dimensions, there is no real difference in generalization performance between Support Vector Machines of varying complexity.

Comparison of CV(10). How do different embeddings change the generalization performce? As an answer we summarize the CV(10) error rates for direct comparison in table 9.9. Each single row was already treated, but this presentation allows to

CV(10)	rbf	poly1	poly2	poly3	poly4	poly5	poly6	poly7
1-tuple	29.5	46.6	52.2	59.3	59.7	59.7	59.7	58.9
2-tuple	30.2	29.1	35	34.2	33.2	33.6	32.8	32.1
3-tuple	22.4	23.5	22.8	22.8	23.5	23.2	23.5	23.5

Table 9.9: Comparison of 10-fold Cross Validation error for different embeddings of the protein sequences: amino acid frequencies (1-tuple, $\mathbb{R}^{20}$), dipeptide frequencies (2-tuple, $\mathbb{R}^{400}$) and tripeptide frequenices (3-tuple, $\mathbb{R}^{8000}$).

directly view the changes in generalization performance due to different embeddings by studying the columns. If the size of the tuples gets greater, more information is contained in the embedding and we expect the error rates to improve. In all polynomial kernel SVM this can clearly be seen, while the rbf-SVM has almost the same CV(10) error for data embedded by single or pairs of amino acids. Generally, the gap between the first and the second row is bigger than the gap between the second row and the third. SVM mainly learn the amino acid frequencies. Including information about the direct neigborhood improves the results, but further information about more distant neighborhood relations has less impact on the classification.

Proposal. We only studied embedding by *direct* neighbors. Maybe we gain an embedding more adequate to protein strucure, if we still embed by triplets of amino acids but allow placeholders between the positions we count, i.e.

$$x_{ijl} \;=\; \#\{(s_k, s_{k+2}, s_{k+4}) = (A_i, A_j, A_l) : \; k = 1, \dots q-4\} \qquad i,j,l = 1, \dots, 20.$$

This results in an input space of 8000 dimensions. The advantage is that it fits better to the construction of α-helices and β-sheets. There are 3.6 amino acid residues per turn in an α-helix, so the embedding covers a whole turn of the helix. In β-strands the side chains of the amino acids point alternatively above and below the strand. Thus, the embedding looks at amino acids that are neighbors in space (but not in the sequence).

9.5 Biological Interpretation

To gain insight into the attractiveness of classes 22 and 29 for misclassifications, we investigate into the biological properties of the according proteins. The goal is to find out, which feature of this classes accounts for the striking pattern of the Confusion Matrices in figures 9.3 and 9.4. Sequences of all other classes are falsely predicted as belonging to class 22 or 29. The sequences of class 29 are very well recognized by Support Vector Machines, while more than half of the sequences in class 22 are misclassified themselves. What is the biology behind the numbers? In the next two sections we will describe the α/β-barrel structure of proteins in class 22 and the structure of virus capsid proteins (class 29). The presentation is mainly based on Branden and Tooze (1991), *Introduction to Protein Structure.*

9.5.1 Class 22: TIM Barrels

The barrel structure is distinguished by a core of eight parallel β-strands arranged close togehter, like staves, into a barrel. The β-strands are connected by α-helices, all lying on the outside of this barrel. This structure is often called a *TIM barrel*, because it was first observed in the enzyme *triosephosphate isomerase.* The eight-stranded α/β-barrel structure is one of the largest and most regular of all protein domain structures. It has been found in many different proteins with completely different amino acid sequences and different functions. The sequence similiarity is generally poor among these proteins. This suggests that the eightfold α/β-barrel could be a rather nonspecific stable motif that is quite tolerant to sequence variations and onto which different functionalities can be designed (Scheerlinck et al., 1992). As an example we show the structure of *Glycolate Oxidase*, which consists solely in a barrel shape (figure 9.6).

The low sequence similiarity between barrel proteins also gives a reason for the observed misclassifications. Class 22 is very heterogeneous, containing an amplitude of diverse

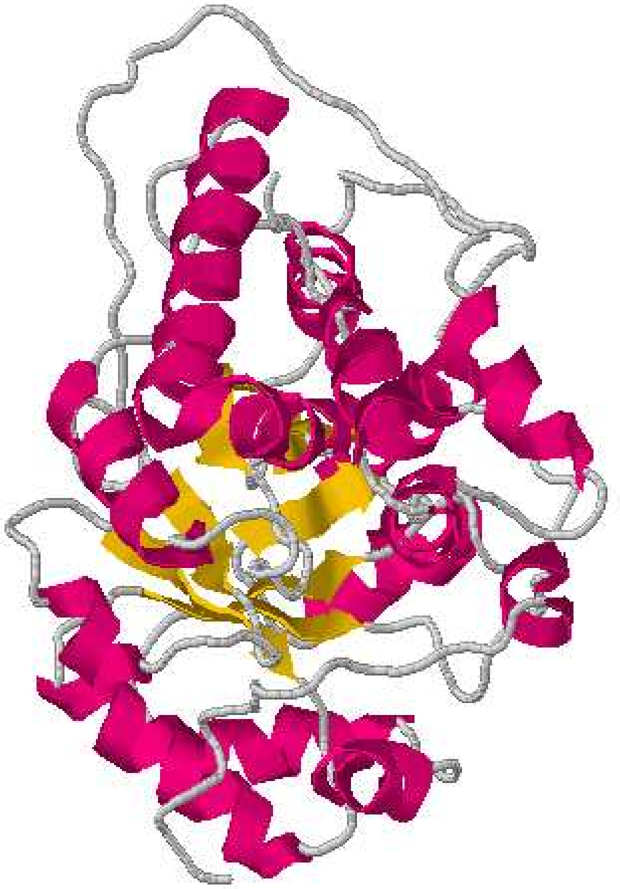

Figure 9.6: Example of α/β-barrel structure: *Glycolate Oxidase* (PDB-ID: *1gox*). The yellow arrows in the middle represent the eight β-strands, which are arranged into a barrel. They are connected by (red-colored) α-helices. *1gox* belongs to the SSC class α/β: there are as many helices as strands and they are not separated in distinct domains, but alternate. This motif is common to many proteins with completely different amino acid sequence.

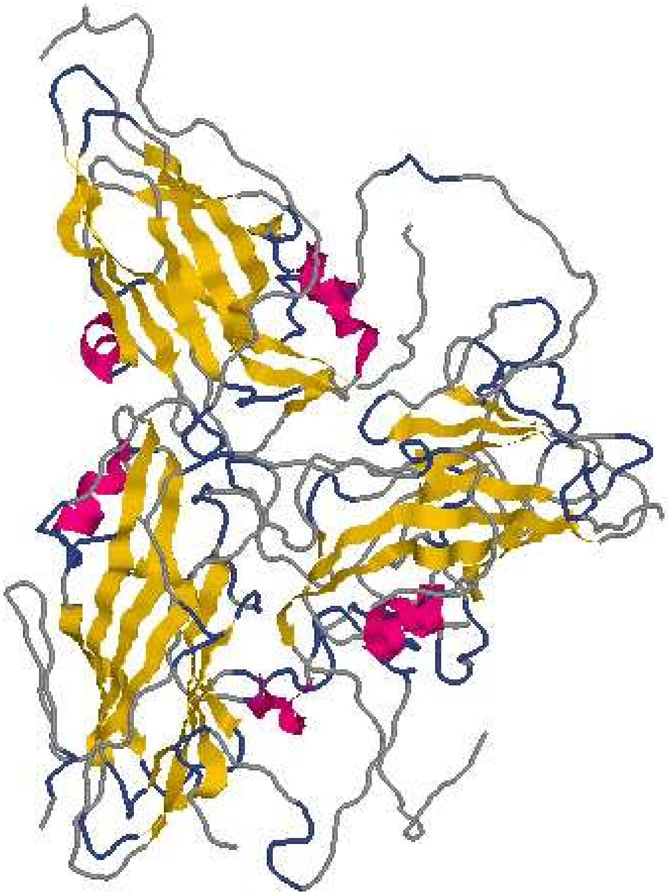

Figure 9.7: Example of a virus capsid protein: *Mengo Encephalomyocarditis Virus Coat Protein* (PDB-ID: *2mev*). 60 copies of this triangular shaped molecule build the shell of the virus. The subunits are arranged to form a icosahedron. The three chains of this protein have very low sequence similiarity. *2mev* belongs to the SSC class β, and indeed we see almost no α-helical regions.

sequences. This explains why its elements are often misclassified. In addition, the extent of this class causes the high number of misclassified sequences from other classes. From the diversity of barrel sequences follows that the hyperplanes at the boundary of this class are chosen to embrace much volume. And this means that often sequences from other classes will happen to fall into this space where they do not belong.

9.5.2 Class 29: Virus Capsid Proteins

Viruses are maybe the simplest form of life. They are constructed from two basic components: a *genomic nucleic acid* molecule, which is surrounded by a protein shell, a *capsid* (and sometimes by an additional *envelope*). No nucleic acid can code for a single protein molecule that is big enough to enclose it. Therefore, the protein shell of a virus is built up from many copies of one or a few polypeptide chains.

As an example we study the shell of *picorna viruses*. It is a spherical construction of 60 copies each of four polypeptide chains, called VP1 to VP4. The first three chains VP1, VP2 and VP3 are the main building blocks, whereas VP4 is much smaller and can be considered as a continuation of VP2 that has become detached. We find examples of these chains in our dataset by the PDB codes *2mev-1* to *2mev-3* (see figure 9.7 for illustration) or *2plv-1* to *2plv-3*.

The three subunits VP1, VP2 and VP3 exhibit no significant amino acid sequence similiarity. So here again we encounter a class of proteins belonging to the same folding class but with great differences on the sequence level. As in the case of *TIM barrels*, this explains the extent and diversity of the class, which in turn results in many false positives.

In contrast, the same subunit from different *picorna viruses* show a high rate of sequence identity, for example 30% for VP2 from *Mengo virus* and *rhinovirus*. Roughly speaking, the whole class is divided into three clusters according to the three protein subunits. This explains the high number of true positives: every sequence in the test set has nearly related sequences in the training set.

9.6 Summary

With the biological excursus of the last section we have ended our analysis of the protein folding dataset. We will now sum up the main results of this study, ranging from technical topics, as the problem of black box methods and performance in high dimensional spaces, to topics in application like a sensible choice of embedding and the biological interpretation. In the assessment of the following points, it should always be remembered that they were achieved on only one particular dataset, even if they are formulated very general and pointedly.

SVM are no black box method. We already addressed this question at the end of section 5.3 at page 56 and derived arguments from the theory of kernel functions. In addition we can now also point out the analysis done in this chapter. The discussion of the error weight C and the analysis of the support vectors show, that Support Vector Machines are a transparent method and can be given a clear interpretation.

Only few parameters are to be chosen. Once the kernel is chosen, we have only to adjust two parameters: first, the width σ of radial basis function kernels or the degree d of polynomial kernels, and second, the error weight C. We do not have to choose a complicated architecture as in Neural Networks. SVM with rbf-kernels have a broad 'valley of test-error' and we can reach it by a simple heuristic; SVM with polynomial kernels are stable over a wide range of degrees and in our example show overfitting only for $d > 60$.

Excellent performance in very high-dimensional spaces. Our results proof that SVM are able to achieve high performance in spaces where many other methods break down. The competing methods could achieve their best results only after reducing the dimensionality by Principal Component Analysis, while Support Vector Machines work without decrease of performance even in an 8000 dimensional input space!

SVM outperform classical methods. The ability of SVM to handle high dimensional data leads to classification results clearly surpassing those of the competitors. Even when the sequences were embedded by their dipeptide-frequencies, the error rates were better than those achieved with the classical methods, but the real power of Support Vector Machines shows in the embedding by the frequency of 3-tuples of amino acids. The exciting thing is not the increase in generalization performance (this was to be expected from the plus of information contained in this embedding), but the fact that SVM can handle this high-dimensional data without the use of any reduction methods.

What do SVM learn? We studied how the generalization ability changes with the information content of the embedding. Support Vector Machines primarily learn the peptide frequency and only secondarily neighborhood relations.

The results can be interpreted biologically. In general, Support Vector Machines do better than other methods. Whenever errors occur in large numbers, they can be explained by pecularities of the underlying proteins, long known in biological literature. In comparison to all other methods, Support Vector Machines bring out the biological structures most clearly.

Part IV

Appendices

Appendix A

Protein Fold Classes

In this appendix we have listed all the data used in the Protein Fold Class Prediction by SVM and other statistical classification methods. The tables can be found at `http://www.dkfz.de/biostatistics/protein/gsme97.html` as an addenda to *Protein Fold Class Prediction: New methods of Statistical Classification* by Grassmann et al. (1999).

The first table shows the names of the classes belonging to of 42-CAT and SSC. Also, this table shows how SSC is subdivided into 42-CAT.

The second and the third table give the identification-codes of Brookhaven Protein Data Base (PDB) (Berman et al., 2000) for the sequences belonging to the training and test set used in the construction of the classifiers. PDB is an international repository for the processing and distribution of 3-D macromolecular structure data primarily determined experimentally by X-ray crystallography and NMR. It can be found at `http://www.rcsb.org/pdb/`.

A.1 42-CAT and SSC

42-CAT	short name	long name	SSC
1	gap	catabolite gene activation	$\alpha + \beta$
2	cytc	cytochromes c	α
3	hmr	hemerythrins	α
4	wrp	trp repressor	α
5	ca-bind	calcium binding proteins	α
6	globin	globins	α
7	lzm	lysozymes	α
8	crn	crambin	$\alpha + \beta$
9	cyp	cytochrome c peroxidase	α
10	ac-prot	acid proteinases	β
11	pap	sulphur proteinases	α
12	256b	cytochromes c, b562	α
13	hoe	alpha-amilase inhibitor	β
14	sns	staphylococcal nuclease	$\alpha + \beta$
15	ferredox	ferredoxins	β
16	cpp	cytochrome P450	$\alpha + \beta$
17	pgk	phosphoglicerate kinase	α / β
18	xia	xylose isomerases	α / β
19	kinase	phosphofroctokinases	α / β
20	binding	periplasmic binding proteins	α / β
21	tln	thermolysin	α
22	barrel	alpha/beta barrel proteins	α / β
23	inhibit	proteinases inhibitors	α / β
24	pti	trypsin inhibitor	β
25	plasto	plastocyanins	β
26	cts	citrate synthase	α
27	rdx	rubredoxins	β
28	plipase	phopsholipases	$\alpha + \beta$
29	virus	virus capsid proteins	β
30	virus-prot	virus proteinases	β
31	cpa	carboxypeptidase a	β
32	dfr	dihydrofolate reductase	α / β
33	igb	immunoglobulins	β
34	il	interleukin domains	β
35	fxc	chloroplast-type ferredoxin	β
36	sbt	subtilisins	α / β
37	gcr	gamma crystallin domains	β
38	tox	toxins	β
39	wga	germ agglutinin domains	$\alpha + \beta$
40	eglin	proteinases inhibitors	α / β
41	ltn	lectins	β
42	s-prot	serine proteinases	$\alpha + \beta$

A.2 Training Set of 143 Proteins

CAT	short name	Training-Set (PDB-Code, first residue, last residue)
1	gap	3gap-A 1 208
2	cytc	155c 1 134,1cc5 5 87,1ccr 1 111,1cyc 1 103
3	hmr	1hmd,1hrb
4	wrp	2wrp-R 5 108,3pgm 29 222
5	ca-bind	1cb1,1pal,5tnc,1trc,3cln 5 147,3icb 1 75
6	globin	1eca 1 136,1fdh,1fdh-G 1 146,1hds,1mba 1 146,1mbd 1 153, 1nih,1pbx,1pmb-A 1 153,1mbs 1 153
7	lzm	1alc 1 122,1lz1 1 130
8	crn	1crn 1 46,9api
9	cyp	1ccd,1utg 1 70,2cyp
10	ac-prot	1cms 1 175,1cms 176 323,2apr 1 178,2apr 179 325,3app 1 174, 3app 175 323
11	pap	1ppo,2act 1 218
12	256b	2ccy 2 128,2tmv-P 19 147
13	hoe	1hoe 1 74,1hsa-B 6 83
14	sns	2pka-B 100 245,2snm
15	ferredox	1fdx 1 54,1fdx 27 54,1fxb 1 81
16	cpp	1lh7,2cpp 10 414
17	pgk	3pgk
18	xia	1xis,3xia 1 377
19	kinase	1pfk-A 1 319
20	binding	1abp 1 306,2gbp 1 309,2lbp 1 36
21	tln	1npc,3tln 1 316
22	barrel	1fcb-A 100 511,1gox 1 359,1tim-A 1 248,1wsy-A 1 265
23	inhibit	1ovo-A 1 56,1tgs-I 1 56,2ovo 1 56
24	pti	5pti 1 58
25	plasto	1acx 1 106,1azu 4 127,2paz,1pcy 1 99
26	cts	1sdh-A 14 146,2cts 1 437
27	rdx	1caa,1rdg 1 52,3rxn 1 52
28	plipase	1bp2 1 123,1p2p 1 124,1pp2-R 1 133
29	virus	1r1a-1 5 287,1r1a-2 11 263,1r1a-3 1 238,2mev-1 1 268,2mev-2 8 256, 2mev-3 1 231,2plv-1 6 302,2plv-2 5 272
30	virus-prot	1hhp,1mvp,2rsp-A 1 124
31	cpa	1lap 170 484,5cpa 1 307
32	dfr	1dhf-A 4 185,3dfr 1 162
33	igb	1cd4 1 98,1f19-H 1 123,1f19-H 124 220,1f19-L 1 108,1f19-L 109 215, 1fc2-D 340 443,1mcp-H 1 122,1mcp-L 1 113,1mcw-W 1 111,1pfc, 2fb4-H 1 118,2fb4-H 119 221,2fb4-L 1 109,2fb4-L 110 214,2fbj, 1fc2-D 238 339,1rei-A 1 107
34	il	1i1b 108 153,1i1b 3 51,1i1b 52 107,1sn3 1 65
35	fxc	3fxc 1 98,1fxi
36	sbt	1cse-E 1 275,1sbt 1 275
37	gcr	1gcr 1 39,1gcr 129 173,1gcr 40 87,2gcr 1 39
38	tox	1ctx,1ctx 1 71,1nbt 1 66,1nea
39	wga	7wga-A 1 43,7wga-A 130 171,7wga-A 44 86,7wga-A 87 130
40	eglin	1cse-I 8 70
41	ltn	2ltn-A 1 108,2ltn-A 109 181,2ltn-B 1 47
42	s-prot	1hne-E 16 243,1sgt 16 245,1tld,1ton 16 246,2alp 15 245,2kai, 2pka-A 16 246

A.3 Test Set of 125 Proteins

CAT	short name	Test-Set (PDB-Code, first residue, last residue)
1	gap	r09-2 15 250
2	cytc	1ycc,2c2c,3c2c 1 112,451c 1 82,5cyt-R 1 103
3	hmr	2hmz-A 1 113,2mhr 1 118
4	wrp	7icd 292 383
5	ca-bind	4cln,4tnc 3 162,5cpv 1 108,5pal
6	globin	2dhb,2lh1 1 153,2lhb 1 149,2mb5,2mhb-A 1 141,2mhb-B 1 146 4hhb-A 1 141,4hhb-B 1 146,2mm1
7	lzm	2lzt 1 129,3lzm 1 164
8	crn	9api-A 56 161
9	cyp	2gls-A 301 449,3ccp,5er2-E 21 295
10	ac-prot	3er5,3psg,4ape -2 174,4ape 175 326,4pep 1 174,4pep 175 326
11	pap	9pap 1 212
12	256b	256b 2 128
13	hoe	3ait
14	sns	2sns 1 141
15	ferredox	4fd1 1 106,4fd1 31 57
16	cpp	2lh4 1 153
17	pgk	1tpt 75 313
18	xia	4xia-A,6xia
19	kinase	3pfk 1 320
20	binding	2liv 1 344,3gbp,5abp
21	tln	5p21 2 166
22	barrel	1ypi-A 2 248,2taa-A 1 478,6tim
23	inhibit	3sgb-I 7 56,4ovo
24	pti	1lts-A 4 116,8pti 1 58
25	plasto	2aza-A 1 129,7pcy
26	cts	6cts
27	rdx	4rxn 1 54,6rxn 1 52,7rxn
28	plipase	3bp2,4p2p
29	virus	2plv-3 1 235,2rm2,2stv 12 195,2tbv-A 102 272,4rhv-1 17 289 4rhv-3 1 236,4sbv-A 62 260,4rhv-2 8 262
30	virus-prot	3hvp 1 99,5hvp,9hvp
31	cpa	6cpa
32	dfr	4dfr-A 1 159,8dfr 1 186
33	igb	2fbj-H 1 118,2fbj-H 119 220,2fbj-L 1 106,2fbj-L 107 213, 2hfl-L 1 105,2hla,2mcg,2rhe 1 114,3fab-H 1 117,3fab-H 118 220, 3fab-L 110 214,3hfm-L 1 108,3hla-A 183 270,3hla-B 1 99, 2hfl-H 117 213, 3fab-L 1 109,2hfl-H 1 116,4fab-L 1 112
34	il	2gn5 1 87,2i1b 5 106,4i1b 56 146
35	fxc	1ubq 1 73,1fxa
36	sbt	1tec-E 1 279,2prk 1 279
37	gcr	1gcr 88 128,2gcr 129 174,2gcr 40 87,2gcr 88 128
38	tox	1ntx 1 62,1nxb 1 62,2abx-A 1 74,6ebx
39	wga	9wga-A 1 43,9wga-A 130 171,9wga-A 44 86,9wga-A 87 130
40	eglin	2ci2-I 19 83
41	ltn	3cna 1 237,4cna
42	s-prot	2ptn 16 245,2sga 16 242,2trm 16 245,3est 16 245,3rp2-A 16 243 3sgb-E 16 242,4cha-A 1 245

Appendix B

The Competitors

Grassmann et al. (1999) use several statistical methods on the training data: Feed Forward Neural Networks, Additive Model, Projection Pursuit Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Flexible Discriminant Analysis and the k-Nearest-Neighbor Rule. We give a short description of each of these classification methods. As the comparison to Neural Networks is most important in the discussion of SVM, we will introduce them in greater detail.

B.1 k-Nearest Neighbor (kNN)

This a very simple but effective classification scheme. All we need is a distant function between pairs of observations. For each element of the test set we then find the k closest observations in the learning set and predict the class by majority vote, i.e. choose the class that is most common among the k neighbors. Usually, k is chosen by cross-validation.

B.2 Discriminant Analysis (LDA, QDA, FDA)

In this section we will discuss the three methods called Linear Discriminant Analysis (LDA), Quadratic Disrcriminant Analysis (QDA) and Flexible Discriminant Analysis (FDA). They all result from a maximum likelihood approach to classification.

If the class conditional densities $p(x|y = k)$ are known, the maximum likelihood rule assigns a test point x to the calss $\hat{y}$ which gives the largest likelihood to x:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ p(x|y = k)$$

If the class conditional densities were fully known, a learning set would not be needed and the classifier is simply $\hat{y}$. Because we don't have full information, the training set is used for estimating the parameters of the densities.

QDA. In the case of a normal distribution $x|y = k \sim \mathcal{N}(\mu_k, \Sigma_k)$, the ML rule is

$$\hat{y} = \underset{k}{\operatorname{argmin}} \{(x - \mu_k)\Sigma_k^{-1}(x - \mu_k)^t + \log|\Sigma_k|\}.$$

In general, this is a quadratic discriminant rule.

LDA is a special case, where the class densities have the same covariance matrix $\Sigma_k = \Sigma \; \forall k$. The linear discriminant rule is based on the Mahalanobis distance between the test point and the centroids of the classes:

$$\hat{y} = \underset{k}{\operatorname{argmin}} \, (x - \mu_k)\Sigma^{-1}(x - \mu_k)^t$$

For the constant covariance matrix Σ, the pooled estimate of the within-class covariance matrices is used.

FDA generalizes LDA to a non-parametric post-processing multi-response regression using an optimal scoring technique to improve the classification. For details see HASTIE ET AL. (1994).

B.3 Projection Pursuit Regression (PPR)

Classification problems can be formulated as regression problems: fit the function $f(x) = \mathbb{E}(Y|X = x)$. We then need only to know whether $f(x)$ is below or above 0.5 to reach a class decision. One way of fitting that allows linear combinations (projections) of the data points is PPR.

The projection pursuit regression of FRIEDMAN and STÜTZLE (1981) has the form

$$\hat{y} = \alpha + \sum_{i=1}^{I} \gamma_i \phi_i(\alpha_i + \beta_i^t x)$$

where the number of terms I has to be selected appropriatly. The ϕ_i are normalized (zero-mean, uni-variance) smooth functions. See (RIPLEY, 1994).

B.4 Additive Model (BRUTO)

The additive model of Hastie and Tibshirani [1990] is given by

$$f_k(x) = a_k + \sum_{j=1}^{p} \phi_{kj}(x_j)$$

where the ϕ's play the role of smoothing functions suitable for non-linearities. Grassmann uses a special case of this additive model with the ϕ's being smoothing splines. This is known as the BRUTO method, (HASTIE and TIBSHIRANI, 1990).

B.5 Neural Networks (NN)

We now come to a major competitor of SVM from the field of machine learning. It was inspired by the neuro-physiological analogy of superposition of several neurons. Neural Networks are also called Multilayer-Perceptrons and generalize Rosenblatt's perceptron [1950s] which only consists of one input and one output layer without intermediate levels. A Neural Network is a combination of several levels of sigmoid elements, where the outputs of one layer form the input for the next layer. A Neural Network containing k levels of intermediate or "hidden" layers is called NN(k). See figure B.1 for illustration.

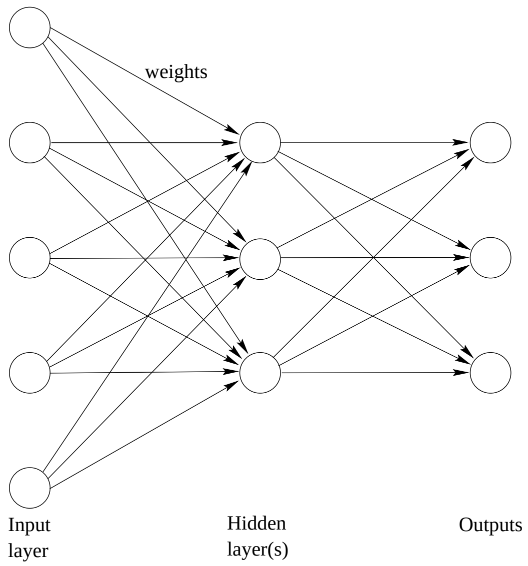

Figure B.1: NN(1). An example of a Feed Forward Neural Network with one hidden layer. All units in one layer are connected to all in the next layer. The connection is described by the $n_k \times n_{k-1}$-matrix of weights.

The input units at level 0 just provide a "fan-out" and distribute the inputs to the next hidden layer. Each unit x_i^k in level $k = 1, \ldots, m$ can be written as taking a fixed *activation function* ϕ_k of a linear combination of the n_{k-1} units in layer $k-1$:

$$x_i^k \;=\; \phi_k \left(\sum_{j=1}^{n_{k-1}} w_j^k x_j^{k-1} \right) \;=\; \phi_k \left(\langle w^k, x^{k-1} \rangle \right),$$

where $k = 1, \ldots, m$ and $i = 1, \ldots, l$. The connection of layer $k-1$ with layer k is given by the $n_k \times n_{k-1}$-matrix of all the weights w_j^k.

As activation functions normally gaussian or sigmoid type are taken, e.g. the Fermi function

$$\phi(z) = \frac{1}{1+\exp(-z)}.$$

In the case of binary classification, the last layer contains only one output unit and is controlled by an indicator activation function.

The weights have to be chosen to minimize the empirical risk $R = R_{emp}(w)$. A simple example is the least squares distance of the output $\hat{y}_i$ for the ith example pattern to the true value y_i. In the binary classification case this is just the number of errors on the training set:

$$R = \sum_{i=1}^{l} (\hat{y}_i - y_i)$$

Other measures have been proposed, like minimizing the Kullback-Leibler divergence

$$R = \sum_{i=1}^{l} \sum_{k} y_i^k \log\left(\frac{y_i^k}{\hat{y}_i^k}\right) + (1 - y_i^k) \log\left(\frac{1-y_i^k}{1-\hat{y}_i^k}\right).$$

The derivatives of a fit criterion R with respect to the weights can be calculated recursively from output to input by using the chain rule. This procedure is known as *back propagation*. In the classic algorithm R is minimized by taking fixed steps in the direction of the steepest descent.

The main problems with the neural net approach to learning are as follows:

1. Convergence of gradient based methods is rather slow.
2. The risk functional may have many local minima. Standard optimization techniques guarantee convergence to one of them. We don't know the quality of the obtained solution, because we don't know the distance to the global minimum.
3. Choosing a problem-adequate topology of a Neural Network is a very difficult task. Much more difficult, than deciding which kernel function to use in a Support Vector Machine.
4. Neural Networks are black box methods, they lack an interpretation of the weights achieved in the optimization process and the solution is not unique, because quite different weights and weight patterns can lead to the same prediction outcome.

Therefore, neural networks are not well-controlled learning machines. In the four points above, they are clearly surpassed by Support Vector Machines.

Appendix C

Proof of Theorem 7.2

Theorem 7.2 (page 64) *The inequality holds true:*

$$P\left\{\sup_{\alpha\in\Lambda}|R(\alpha)-R_{emp}(\alpha)| \;>\; \epsilon\right\} \;<\; 4\exp\left(H^{\Lambda}_{ann}(2l)-l\left(\epsilon-\frac{1}{l}\right)^2\right)$$

The proof of this theorem is based on the following lemma. Let us denote by $Z(2l)$ the space of random independent observations of size $2l$, and a specific sample from this space by $Z^{2l}=(z_1,\dots z_l,z_{l+1},\dots,z_{2l})$. For any function in $\{L(z,\alpha)\}_{\alpha\in\Lambda}$ we determine the frequency of error $R^{(1)}_{emp}(\alpha)=\frac{1}{l}\sum_{i=1}^{l}L(z_i,\alpha)$ on the first half of the sample $Z_1=(z_1,\dots,z_l)$ and the frequency $R^{(2)}_{emp}(\alpha)=\frac{1}{l}\sum_{i=l+1}^{2l}L(z_i,\alpha)$ on the second half $Z_2=(z_{l+1},\dots,z_{2l})$. We define the random variables

$$\begin{aligned}
\rho^{\Lambda}(\alpha,Z^{2l}) &:= |R^{(1)}_{emp}(\alpha)-R^{(2)}_{emp}(\alpha)| \\
\rho^{\Lambda}(Z^{2l}) &:= \sup_{\alpha\in\Lambda}\rho^{\Lambda}(\alpha,Z^{2l}) \\
\pi^{\Lambda}(\alpha,Z_1) &:= |R(\alpha)-R^{(1)}_{emp}(\alpha)| \\
\pi^{\Lambda}(Z_1) &:= \sup_{\alpha\in\Lambda}\pi^{\Lambda}(\alpha,Z_1)
\end{aligned}$$

We assume $\pi^{\Lambda}(Z_1)$ and $\rho^{\Lambda}(Z^{2l})$ to be measurable with respect to P.

Basic Lemma, Vapnik 1998. The distribution of $\pi^{\Lambda}(Z_1)$ is connected with the distribution of $\rho^{\Lambda}(Z^{2l})$ by the inequality

$$P\left\{\pi^{\Lambda}(Z_1)>\epsilon\right\} \;<\; 2\,P\left\{\rho^{\Lambda}(Z^{2l})>\epsilon-\frac{1}{l}\right\}.$$

We omit the lengthy proof of this lemma in favour of presenting Vapniks proof of the main theorem.

Proof of Theorem 7.2: (Vapnik, 1998, pp131-137) We are interested in the rate of convergence to zero of $\pi^{\Lambda}(Z_1)$, but according to the lemma we can estimate the rate of convergence of $\rho^{\Lambda}(Z^{2l})$. For simplicity we denote $\epsilon_* := \epsilon - 1/l$. $\mathbb{I}[A]$ is the indicator function that becomes 1 if the event A occurs and 0 otherwise.

Consider the permutations $T_i: \ Z(2l) \to Z(2l), \ i = 1, \dots, (2l)!$ of the elements of the sequence Z^{2l}. By symmetry of measure we gain

$$\int_{Z(2l)} f(Z^{2l}) \, dF(Z^{2l}) \ = \ \int_{Z(2l)} f(T_i Z^{2l}) \, dF(Z^{2l})$$

for all integrable functions $f(Z^{2l})$. Therefore:

$$\begin{aligned} P\{\rho^{\Lambda}(Z^{2l}) > \epsilon_*\} \ &= \ \int_{Z(2l)} \mathbb{I}\left[\rho^{\Lambda}(Z^{2l}) \ > \ \epsilon_*\right] dF(Z^{2l}) \\ &= \ \int_{Z(2l)} \frac{1}{(2l)!} \sum_{i=1}^{(2l)!} \mathbb{I}\left[\rho^{\Lambda}(T_i Z^{2l}) \ > \ \epsilon_*\right] dF(Z^{2l}) \end{aligned} \tag{C.1}$$

Observe that

$$\begin{aligned} \mathbb{I}\left[\rho^{\Lambda}(Z^{2l}) \ > \ \epsilon_*\right] \ &= \ \mathbb{I}\left[\sup_{\alpha \in \Lambda} |R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| \ > \ \epsilon_*\right] \\ &= \ \sup_{\alpha \in \Lambda} \mathbb{I}\left[|R^{(1)}_{emp}(\alpha) - R^{(2)}_{emp}(\alpha)| \ > \ \epsilon_*\right] \end{aligned}$$

If two functions $L(z, \alpha_1)$ and $L(z, \alpha_2)$ are nondistinguishable on the sample $z_1, \dots, z_{2l}$, then $\rho^{\Lambda}(\alpha_1, T_i Z^{2l}) \ = \ \rho^{\Lambda}(\alpha_1, T_i Z^{2l}) \ \forall T_i$, i.e. the deviations in frequencies for these two functions are the same for all permutations T_i.

Therefore, for each class of equivalent functions we can choose only one representative function $L(z, \alpha')$, which forms a finite set of functions $\{L(z, \alpha')\}_{\alpha' \in \Lambda' \subset \Lambda}$ such that

$$\sup_{\alpha \in \Lambda} \rho(\alpha, T_i Z^{2l}) \ = \ \sup_{\alpha' \in \Lambda'} \rho(\alpha', T_i Z^{2l})$$

The number of functions in the set Λ' is finite and does not exceed $N^{\Lambda}(z_1, \dots, z_{2l})$.

$$\begin{aligned} \sup_{\alpha \in \Lambda} \mathbb{I}\left[\rho^{\Lambda}(\alpha, T_i Z^{2l}) \ > \ \epsilon_*\right] \ &= \ \sup_{\alpha' \in \Lambda'} \mathbb{I}\left[\rho^{\Lambda'}(\alpha', T_i Z^{2l}) \ > \ \epsilon_*\right] \\ &\leq \ \sum_{\alpha' \in \Lambda'} \mathbb{I}\left[\rho^{\Lambda'}(\alpha', T_i Z^{2l}) \ > \ \epsilon_*\right] \end{aligned}$$

This allows to bound in integrand in (C.1) by

$$
\begin{aligned}
\frac{1}{(2l)!}\sum_{i=1}^{(2l)!}\mathbb{I}\left[\rho^{\Lambda}(T_i Z^{2l}) \;>\; \epsilon_*\right] &= \frac{1}{(2l)!}\sum_{i=1}^{(2l)!}\sup_{\alpha'\in\Lambda'}\mathbb{I}\left[\rho^{\Lambda'}(\alpha', T_i Z^{2l}) \;>\; \epsilon_*\right] \\
&\leq \sum_{\alpha'\in\Lambda'}\frac{\sum_{i=1}^{(2l)!}\mathbb{I}\left[\rho^{\Lambda'}(\alpha', T_i Z^{2l}) \;>\; \epsilon_*\right]}{(2l)!} \\
&< 2\sum_{\alpha'\in\Lambda'}\exp\{-\epsilon_*^2 l\} \\
&= 2\; N^{\Lambda}(z_1,\ldots,z_{2l})\;\exp\{-\epsilon_*^2 l\}
\end{aligned}
$$

The last inequality holds because the number of summands is bounded by $2\exp\{-\epsilon_*^2 l\}$ (Vapnik, 1998, p163).

Substituting this bound into (C.1) we obtain

$$
\begin{aligned}
P\{\rho^{\Lambda}(Z^{2l}) > \epsilon_*\} &< 2\;\mathbb{E}N^{\Lambda}(z_1,\ldots,z_{2l})\;\exp\{-\epsilon_*^2 l\} \\
&= 2\;\exp\{H_{ann}^{\Lambda}(2l) - \epsilon_*^2 l\}
\end{aligned}
$$

Recalling that $\epsilon_* := \epsilon - 1/l$, the theorem is proved. $\square$

Bibliography

Aizerman, M. A., Braverman, E. M., and Rozonoer, L. I., 1964: Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Autom. Remote Control* 25.

Bartlett, P. and Shawe-Taylor, J., 1999: Generalization Performance of Support Vector Machines and Other Pattern Classifiers. In: B. Schölkopf, C. J. C. Burges, and A. J. Smola (eds.), *Advances in Kernel Methods — Support Vector Learning*. The MIT Press, Cambridge, MA.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P., 2000: The Protein Data Bank. *Nucleic Acids Research* 28, 235–242.

Billingsley, P., 1995: *Probability and Measure*. John Wiley and Sons, N.Y.

Branden, C. and Tooze, J., 1991: *Introduction to Protein Structure*. Garland Publishing, N.Y.

Brown, T. A., 1998: *Genetics: A Molecular Approach*. Stanley Thornes, Cheltenham, 3rd edn.

Burges, C. J. C., 1998: A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining* 2(2).

Cristianini, N. and Shawe-Taylor, J., 2000: *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK.

Edler, L. and Grassmann, J., 1999: Protein Fold Class Prediction is a new field for statistical classification and regression. In: F. Seillier-Moisewitsch (ed.), *Statistics in Molecular Biology and Genetics*, vol. 33 of *IMS Lecture Notes — Monograph Series*. 288–313.

Edler, L., Grassmann, J., and Suhai, S., 2001: Role and Results of Statistical Methods in Protein Fold Class Prediction. *Mathematical and Computer Modelling* 33, 4201–4217.

Efron, B., 1982: *The Jackknife, the Bootstrap and Other Resampling Plans*. Society for Industrial and Applied Mathematics, Philadelphia.

Efron, B. and Tibshirani, R., 1993: *An Introduction to the Bootstrap*. Chapman and Hall, Cambridge.

Fletcher, R., 1987: *Practical Methods of Optimization*. Wiley, N.Y.

Friedman, J. H. and Stützle, W., 1981: Projection Pursuit Regression. *J. Amer. Statist. Assoc.* 76, 817–823.

Grassmann, J., Reczko, M., Suhai, S., and Edler, L., 1999: Protein Fold Class Prediction – New Methods of Statistical Classification. In: T. Lengauer (ed.), *Proceedings of the ISMB 1999, Heidelberg, Aug. 6-10*. AAAI Press, 106–112.

Gunn, S., 1998: Support Vector Machines for Classification and Regression. Tech. rep., Image Speech and Intelligent Systems Group (ISIS) University of Southampton.

Hastie, T. and Tibshirani, R., 1990: *Generalized Additive Models*. Chapman and Hall, Cambridge.

Hastie, T., Tibshirani, R., and Buja, A., 1994: Flexible Discriminat Analysis by Optimal Scoring. *J. Amer. Statist. Assoc.* 89, 1255–1270.

Jaakkola, T., Diekhans, M., and Haussler, D., 2000: A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology* 7(1,2), 95–114.

Karakoulas, G. and Shawe-Taylor, J., 1999: Optimizing classifiers for imbalanced training sets. In: M. Kearns, S. Solla, and D. Cohn (eds.), *Advances in Neural Information Processing Systems 11*. The MIT Press, Cambridge, MA.

Kressel, U. H.-G., 1999: Pairwise Classification and Support Vector Machines. In: B. Schölkopf, C. J. C. Burges, and A. J. Smola (eds.), *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA.

Markowetz, F., Edler, L., and Vingron, M., 2001: Support Vector Machines for Protein Fold Class Prediction. To appear.

Pascarella, S. and Argos, P., 1992: A databank merging related protein structures and sequences. *Protein Engineering* 5, 121–137.

Reczko, M. and Bohr, 1994: The DEF data base of sequence based protein fold class predictions. *Nucleic Acids Research* 22(17), 3616–3619.

Ripley, B. D., 1994: Neural Networks and Related Methods for Classification. *J. R. Statist. Soc* 56(3), 409–456.

Scheerlinck, J.-P., Lasters, I., Claessens, M., de Maeyer, M., Pio, F., Delhaise, P., and Wodak, S., 1992: Recurrent $\alpha\beta$ Loop Structures in TIM Barrel Motifs Show a Distinct Pattern of Structural Features. *Proteins: Structure, Function, and Genetics* 12, 299–313.

Schölkopf, B., Burges, C. J. C., and Smola, A. J. (eds.), 1999: *Advances in Kernel Methods — Support Vector Learning*. The MIT Press, Cambridge, MA.

Shawe-Taylor, J., Bartlett, P., Williamson, R., and Anthony, M., 1998: Structural Risk Minimization over Data-Dependent Hierarchies. *IEEE Transactions on Information Theory* To appear.

Shawe-Taylor, J. and Cristianini, N., 1998: Data-Dependent Structural Risk Minimization for Perceptron Decision Trees. *Advances in Neural Information Processing Systems 10* .

Steipe, B., 2000: Introduction to Protein Structure. In: *Bioinformatics Summer School (BISS) Reader*. Practical Informatics, Bielefeld University.

Vapnik, V., 1995: *The Nature of Statistical Learning Theory*. Springer, N.Y.

Vapnik, V., 1998: *Statistical Learning Theory*. Wiley, N.Y.

Vapnik, V. and Chervonenkis, A., 1964: A Note on one Class of Perceptrons. *Automation and Remote Control* 25.

Vapnik, V. and Lerner, A., 1963: Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control* 24.

Veropoulos, K., Campbell, C., and Cristianini, N., 1999: Controlling the Sensitivity of Support Vector Machines. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99)*. Stockholm, Sweden.

Walsh, G., 1975: *Methods of Optimization*. John Wiley and Sons, N.Y.

Weston, J. and Watkins, C., 1998: Multi-class Support Vector Machines. Tech. rep., Royal Holloway, University of London.