Diplomarbeit

Support Vector Machines in der digitalen Mustererkennung

Ausgeführt bei der Firma Siemens VDO in Regensburg

vorgelegt von: Christian Miklos
St.-Wolfgangstrasse 11
93051 Regensburg

Betreuer: Herr Reinhard Rösl
Erstprüfer: Prof. Jürgen Sauer
Zweitprüfer: Prof. Dr. Herbert Kopp

Abgabedatum: 03.03.2004
Acknowledgements

This work was written as my diploma thesis in computer science at the university of applied sciences Regensburg, Germany, under the supervision of Prof. Dr. Jürgen Sauer.

The research was carried out at Siemens VDO in Regensburg, Germany. In Reinhard Rösl I found a very competent advisor there, whom I owe much for his assistance in all aspects of my work. Thank you very much!

For the help during writing this document I want to thank all colleagues at the department at Siemens VDO. I have enjoyed the work there very much in any sense and learned a lot which sure will be useful in the upcoming years.

My special thanks go to Prof. Jürgen Sauer who helped me out in any questions arising during this work.
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>NOTATIONS</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>INTRODUCTION</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>AN INTRODUCTION TO THE LEARNING THEORY AND BASICS</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>SUPERVISED LEARNING THEORY</td>
<td>10</td>
</tr>
<tr>
<td>1.1</td>
<td>Modelling the Problem</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>LEARNING TERMINOLOGY</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>Risk Minimization</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Structural Risk Minimization (SRM)</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>The VC Dimension</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>The VC Dimension of Support Vector Machines, Error Estimation and Generalization Ability</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>PATTERN RECOGNITION</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Feature Extraction</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Classification</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>OPTIMIZATION THEORY</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>The Problem</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Lagrangian Theory</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Duality</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Kuhn-Tucker Theory</td>
<td>33</td>
</tr>
<tr>
<td>II</td>
<td>SUPPORT VECTOR MACHINES</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>LINEAR CLASSIFICATION</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Linear Classifiers on Linear Separable Data</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>The Optimal Separating Hyperplane for Linear Separable Data</td>
<td>39</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Support Vectors</td>
<td>46</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Classification of unseen data</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>The Optimal Separating Hyperplane for Linear Non-Separable Data</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1</td>
<td>1-Norm Soft Margin - or the Box Constraint</td>
<td>52</td>
</tr>
<tr>
<td>5.3.2</td>
<td>2-Norm Soft Margin - or Weighting the Diagonal -</td>
<td>54</td>
</tr>
<tr>
<td>5.4</td>
<td>The Duality of Linear Machines</td>
<td>57</td>
</tr>
<tr>
<td>5.5</td>
<td>Vector/Matrix Representation of the Optimization Problem and Summary</td>
<td>58</td>
</tr>
</tbody>
</table>
The Support Vector Machine (SVM) is a new and very promising classification technique developed by Vapnik and his group at AT&T Bell Laboratories. This new learning algorithm can be seen as an alternative training technique for Polynomial, Radial Basis Function and Multi-Layer Perceptron classifiers. Recently it has shown very good results in the pattern recognition field of research, such as hand-written character and digit or face recognition but they also proofed themselves reliable in text categorization. It is mathematically very funded and of great growing interest nowadays in many new fields of research such as Bioinformatics.

NOTATIONS

\( \mathbf{x} \)  
input vector (input during training, already labelled)

\( \mathbf{z} \)  
input vector (input after training, to be classified)

\( y \)  
output: class of input \( \mathbf{x} \) (\( \mathbf{z} \))

\( X \)  
input space

\( \mathbf{x}' \)  
vector \( \mathbf{x} \) transposed

\( \langle \mathbf{a} \cdot \mathbf{b} \rangle \)  
inner product between vector \( \mathbf{a} \) and \( \mathbf{b} \) (dot product)

\( \text{sgn}(f(\mathbf{x})) \)  
the signum function: +1 if \( f(\mathbf{x}) \geq 0 \) and -1 else

\( I \)  
training set size

\( S \)  
training set \( \{(\mathbf{x}_i, y_i): i = 1..I\} \)

\( (\mathbf{w}, b) \)  
defines the hyperplane \( H = \{\mathbf{x} : \langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0\} \)

\( \alpha_i \)  
Lagrange multipliers

\( \xi_i \)  
slack variables (for linear non-separable datasets)

\( \gamma_i \)  
margin of a single point \( \mathbf{x}_i \)

\( L_P, L_D \)  
Lagrangian: primal and dual

\( C \)  
error weight

\( K(\mathbf{a}, \mathbf{b}) \)  
kernel function calculated with vectors \( \mathbf{a} \) and \( \mathbf{b} \)

\( K \)  
kernel matrix \( (k(\mathbf{x}_i, \mathbf{x}_j)) \)

\( SVM \)  
Support Vector Machine

\( SV \)  
support vectors

\( nsv \)  
number of support vectors

\( rbf \)  
radial basis functions

\( LM \)  
learning machine

\( ERM \)  
Empirical Risk Minimization

\( SRM \)  
Structural Risk Minimization
Chapter 0

Introduction

In this work the rather new concept in learning theory, the Support Vector Machine, will be discussed in detail. The goal of this work is to give an insight into the methods used and to describe them in a way a person with not so much funded mathematical background could understand them. So the gap between theory and practice could be closed. It is not the intention of this work to look in every aspect and algorithm available in the field of this learning theory but to understand how and why it even works and why it is of such rising interest at the time.

This work should lay the basics for understanding the mathematical background, to be able to implement the technique and to do further research whether this technique is suitable for the wanted purpose at all.

As a product of this work the Support Vector Machine will be implemented both in Matlab and C++. The C++ part will be a module integrated into the so called “Neural Network Tool” already used in the department at Siemens VDO, which already implements the Polynomial and Radial-Basis Function classifiers. This tool is for testing purposes to test suitable techniques for the later integration into the lane recognition system for cars currently under development there.

Support Vector Machines for classification are a rather new concept in learning theory. It’s origins reach back to the early 60’s (VAPNIK and LEARNER 1963; VAPNIK and CHERVONENKIS 1964), but it stirred up attention only in 1995 with Vladimir Vapnik’s book *The Nature of Statistical Learning Theory* [Vap95]. In the last few years Support Vector Machines proofed excellent performance in many real-word applications such as hand-written character recognition, image classification or text categorization.

But because many aspects in this theory are still under intensive research the number of introductory literature is very limited. The two books by Vladimir Vapnik (*The Nature of Statistical Learning Theory* [Vap 95] and *Statistical Learning Theory* [Vap98] present only a general high-level introduction to this field. The first tutorial purely on Support Vector Machines was written by C. Burges in 1998 [Bur98]. In the year 2000 CHRISTIANIN and SHAWE-TAYLOR published *An introduction to Support Vector Machines* [Nel00], which was the main source for this work.
All these books and papers give a good overview of the theory behind Support Vector Machines, but they don't give a straightforward introduction to application. Here this work puts on.

This work is divided into four parts:

**Part I** gives an introduction into the supervised learning theory and the ideas behind pattern recognition. Pattern recognition is the environment in which the Support Vector Machine will be used in this work. The next chapter will lay the mathematical basics for the optimization problem arising.

**Part II** then introduces the Support Vector Machine itself with its mathematical background. For a better understanding the case of classification will be restricted to the two-class problem first but later one can see that this is no problem because it then can easily be extended to the multi-class case. Here also the long studied kernel technique will be analysed in detail which gives the Support Vector Machines their superior power.

**Part III** then analyses the implementation techniques for Support Vector Machines. It will be shown that there are many approaches for solving the arising optimization problem but only the most used and best performing algorithms for a great amount of data will be discussed in detail.

**Part IV** in the end is intended as a manual for the implementation done in Matlab and C++. There will also be given a list of widely used toolboxes for Support Vector Machines, both in C/C++ and Matlab.

Last but not least in the appendix some real-world applications, some calculation examples on the arising mathematical problems, the rather simple Perceptron algorithm for classification and the pseudo code used for the implementation will be stated.
Part I

An introduction to the Learning Theory and Basics
Chapter 1

Supervised Learning Theory

When computers are applied to solve a practical problem it is usually the case that the method of deriving the required output from a set of inputs can be described explicitly. But there arise many cases where one wants the machine to perform tasks that cannot be described by an algorithm. Such tasks cannot be solved by classical programming techniques, since no mathematical model exists for them or the computation of the exact solution is very expensive (it could last for hundreds of years, even on the fastest processors). As examples consider the problem of performing hand-written digit recognition (a classical problem of machine learning) or the detection of faces on a picture.

There is need for a different approach to solve such problems. Maybe the machine is teachable, as children are in school? Meaning they are not given abstract definitions and theories by the teacher but he points out examples of the input-output functionality. Consider the children learning the alphabet. The teacher does not give them precise definitions of each letter, but he shows them examples. Thereby the children learn general properties of the letters by examining these examples. In the end these children will be able to read words in script style, even if they were taught only on types.

In other more mathematical words this observations leads to the concept of classifiers. The purpose of learning such a classifier from few given examples already correctly classified by the supervisor, is to be able to classify future unknown observations correctly.

But how can learning from examples, which is called supervised learning, be formulized mathematically to let it be applied to a machine?
1.1 Modelling the Problem

Learning from examples can be described in a general model by the following elements:
The generator of the input data \(x\), the supervisor who assigns labels/classes \(y\) to the data for learning and the learning machine that returns some answer \(y'\) hopefully close to the one of the supervisor.

The labelled/preclassified examples \((x, y)\) are referred to as the training data. The input/output pairings typically reflect a functional relationship, mapping the inputs to outputs, though this is not always the case, for example when the outputs are corrupted by noise. But when an underlying function exists it is referred to as the target function. So the goal is the estimation of this target function which is learnt by the learning machine and is known as the solution of the learning problem. In case of classification problems, e.g. “this is a man and this is a woman”, this function is also known as the decision function. The optimal solution is chosen from a set of candidate functions which map from the input space to the output domain. Usually a set or class of candidate functions is chosen known as hypotheses. As an example consider so-called decision trees which are hypotheses created by constructing a binary tree with simple decision functions at the internal nodes and output values at the leaves (the \(y\)-values).

A learning problem with binary outputs (0/1, yes/no, positive/negative, …) is referred to as a binary classification problem, one with a finite number of categories as a multi-class classification one, while for real-valued outputs the problem is known as regression. In this diploma thesis only the first two categories will be considered, although the later discussed Support Vector Machines can be “easily” extended to the regression case.

A more mathematical interpretation of this will be given now.
The generator above determines the environment in which the supervisor and the learning machine 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” output values according to a conditional distribution function \(P(y|x)\) (output is dependent on input). This assumption leads to the case \(y = f(x)\) in which the supervisor associates a fixed \(y\) with every \(x\).

The learning machine then is defined by a set of possible mappings \(x \rightarrow f(x, \alpha)\) where \(\alpha\) is element of a parameter space. An example of a learning machine according to binary classification is defined by oriented hyperplanes \(\{x : \langle w \cdot x \rangle + b = 0\}\) where \(\alpha \equiv (w, b) \in \mathbb{R}^{n+1}\) determines the
position of the hyperplanes in $\mathbb{R}^n$. As a result the following learning machine (LM) is obtained:

$$LM = \{ f(x,(w,b)) = \text{sgn}(\langle w \cdot x \rangle + b) ; (w,b) \in \mathbb{R}^{n+1} \}$$

The functions $f : \mathbb{R}^n \rightarrow \{-1,+1\}$, mapping the input $x$ to the positive (+1) or negative (-1) class, are called the decision functions. So this learning machine works as follows: the input $x$ is assigned to the positive class, if $f(x) \geq 0$, and otherwise to the negative class.

The above definition of a learning machine is called a Linear Learning Machine because of the linear nature of the function $f$ used here. Among all possible functions, the linear ones are the best understood and simplest to apply. They will provide the framework within which the construction of more complex systems is possible and will be done in later chapters.

There is need for a choice of the parameter $\alpha$ based on $l$ observations (the training set):

1 This method of a learning machine will be described in detail in Part II, because Support Vector Machines implement this technique.
This is called the training of the machine. The training set $S$ is drawn accordingly to the distribution $P(x, y)$.

If all this data is given to the learner (the machine) at the start of the learning phase, this is called batch learning. But if the learner receives only one example at a time and gives an estimation of the output before receiving the correct value, it is called online learning. In this work only batch learning is considered. Also each of these two learning methods can be subdivided into unsupervised learning and supervised learning.

Once a function for appropriate mapping the input to the output is chosen (learned), one wants to see how well it works on unseen data. Usually the training set is split into two parts: the labelled training set above and the so-called labelled test set. This test set is applied after training, knowing the expected output values, and comparing the results of the classification of the machine with the expected ones to gain the error rate of the machine.

But simply verifying the quality of an algorithm in such a way is not enough. It is not only the goal of a gained hypothesis to be consistent with the training set but also to work fine on future data. But there are also other problems inside the whole process of generating a verifiable consistent hypothesis. First the function tried to learn may have not a simple representation and hence may not be easily verified in this way. Second the training data could be frequently noisy and so there is no guarantee that there is an underlying function which correctly maps the training data.

But the main problem arising in practice is the choice of the features. Features are the components the input vector $x$ consists of. Sure they have to describe the input data for classification in an “appropriate” way. Appropriate means, for example, no or less redundancy. Some hints on choosing a suitable representation for the data will be given in the upcoming chapters but not in detail because this would blow up the frame.

As an example to the second problem consider the classification of web pages into categories, which can never be an exact science. But such data is increasingly of interest for learning. So there is a need for measuring the quality of a classifier in some other way: Good generalization.

The ability of a hypothesis/classifier to correctly classify data, not only the training set, or in other words, make precise predictions by learning from few examples, is known as its generalization ability, and this is the property which has to be optimized.
Chapter 2

Learning Terminology

This chapter is intended to stress the main concepts arising from the theory of statistical learning [Vap79] and the VC Theory [Vap95]. These concepts are the fundamentals of learning machines. Here terms such as generalization ability and capacity will be described.

2.1 Risk Minimization

As seen in the last chapter, the task of a learning machine is to infer general features from a set of labelled examples, the training set. It is the goal to generalize from the training example to the whole range of possible observations. The success of this is measured by the ability to correctly classify new unseen data not belonging to the training set. This is called the Generalization Ability.

But as in the training a set of possible hypotheses arise, there is need for some measure to choose the optimal one what is the same as later measuring the generalization ability.

Mathematically this can be expressed using the Risk Function, a measure of quality, using the expected classification error for a trained machine. This expected risk (the test error), is the possible average error committed by the chosen hypothesis \( f(x, \alpha) \) on the unknown example drawn randomly from the sample distribution \( P(x, y) \):

\[
R(\alpha) = \int \frac{1}{2} |y - f(x, \alpha)| dP(x, y)
\]  

(2.1)

Here the function \( \frac{1}{2} |y - f(x, \alpha)| \) is called the loss (difference between expected output (by supervisor) and the response of the learning machine). \( R(\alpha) \) is referred to as the Risk Function or simply the risk. The goal is to
find parameters $\alpha^*$ such that $f(\mathbf{x}, \alpha^*)$ minimizes the risk over the class of functions $f(\mathbf{x}, \alpha)$. But since $P(\mathbf{x}, y)$ is unknown, the value of the risk for a given parameter $\alpha$ cannot be computed directly. The only available information is contained in the given training set $\mathcal{S}$.

So the empirical risk $R_{\text{emp}}(\alpha)$ is defined to be just the measured mean error rate on the training set of finite length $l$:

$$R_{\text{emp}}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)|$$

(2.2)

Note that here no probability distribution appears and $R_{\text{emp}}(\alpha)$ is a fixed number for a particular choice of $\alpha$ and for a training set $\mathcal{S}$.

For further considerations, assume binary classification with outputs $y_i \in \{-1, 1\}$. Then the loss function can only produce the outputs 0 or 1. Now choose some $\eta$ such that $0 < \eta < 1$. Then for losses taking these values, with probability $1 - \eta$, the following bound holds [Vap95]:

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l}}$$

(2.3)

where $h$ is a non-negative integer called the Vapnik Chervonenkis (VC) dimension. It is a measure of the notion of capacity. The second summand of the right hand side is called the VC confidence.

Capacity is 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. The most popular concept to describe the richness of a function class in machine learning is the Vapnik Chervonenkis (VC) dimension.

Burges gives an illustrative example on capacity in his paper [Bur98]: “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 he has seen before. A machine with too little capacity is like the botanist’s lazy brother, who declares that if it is green, it is a tree. Neither can generalize well.”
To conclude this subchapter there can be drawn three key points about the bound of (2.3):

First it is independent of the distribution $P(x, y)$. It only assumes that the training and test data are drawn independently according to some distribution $P(x, y)$. Second, it is usually not possible to compute the left hand side. Third, if $h$ is known, it is easily possible to compute the right hand side.

The bound also shows that low risk depends both on the chosen class of functions (the learning machine) and on the particular function chosen by the learning algorithm, the hypothesis, which should be optimal. The bound decreases if a good separation on the training set is achieved by a learning machine with low VC dimension. This approach leads to principles of the \textit{structural risk minimization (SRM)}.

\section{2.2 Structural Risk Minimization (SRM)}

Let the entire class of functions $K = \{f(x, \alpha)\}$, be divided into nested subsets of functions such that $K_1 \subset K_2 \subset \ldots \subset K_n$. For each subset it must be able to compute the VC dimension $h$, or get a bound on $h$ itself. Then SRM consists of finding that subset of functions which minimizes the bound on the risk. This can be done by simply training a series of machines, one for each subset, where for a given subset the goal of training is simply to minimize the empirical risk. Then the trained machine in the series whose sum of empirical risk and VC confidence is minimal.

So overall the approach is working as follows: The confidence interval is kept fix (by choosing particular $h$’s) and the empirical risk is minimized. In the neural network case this technique is adapted by first choosing an appropriate architecture and then eliminating classification errors. The second approach is to keep the empirical risk fixed (e.g. equal to zero) and minimize the confidence interval. Support Vector Machines will also implement the principles of SRM, by finding the one canonical hyperplane among all, which minimizes the norm $\|w\|^2$ in the definition of a hyperplane by: $\langle w \cdot x \rangle - b = 1$.\footnote{SVMs, hyperplanes, canonical hyperplanes and why minimizing the norm will be explained in part II of this work in detail, here only a reference is given to this.}
2.3 The VC Dimension

The VC dimension is a property of a set of functions \( \{f(\alpha)\} \) and can be defined for various classes of functions \( f \). But again, here only the functions corresponding to the two-class pattern case with \( y \in \{-1, 1\} \) are considered.

**Definition 2.1 (Shattering)**

If a given set of \( l \) points can be labelled in all possible \( 2^l \) ways, and for each labelling, a member of the set \( \{f(\alpha)\} \) can be found which correctly assigns those labels (classifies), this set of points is said to be shattered by that set of functions.

**Definition 2.2 (VC Dimension)**

The Vapnik Chervonenkis (VC) Dimension of a set of 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.

Note that if the VC dimension is \( h \), then there exists at least one set of \( h \) points that can be shattered, but in general it will not be true that every set of \( h \) points can be shattered.

As an example consider shattering points with oriented hyperplanes in \( \mathbb{R}^n \). To give an introduction assume the data lives in the \( \mathbb{R}^2 \) space and the set of functions \( \{f(\alpha)\} \) consists of oriented straight lines, so that for a given line, all points on one side are assigned to the class +1, and all points on the other one to the class -1. The orientation in the following figures is indicated by an arrow, specifying the side where the points of class +1 are lying. While it is possible to find three points that can be shattered (figure 2.1) by this set of functions, it is not possible to find four. Thus the VC dimension of the set of oriented lines in the \( \mathbb{R}^2 \) is three.

Without proof (this can be found in [Bur98]) it can be stated that the VC dimension of the set of oriented hyperplanes in the \( \mathbb{R}^n \) is \( n+1 \).
Figure 2.1: Three points not lying in a line can be shattered by oriented hyperplanes in the $\mathbb{R}^2$. The arrow points in the direction of the positive examples (black). Whereas four points can be found in the $\mathbb{R}^2$, which cannot be shattered by oriented hyperplanes.

2.4 The VC Dimension of Support Vector Machines, Error Estimation and Generalization Ability

It should be said first that this subchapter does not claim completeness in any sense. There will be no proofs on the conclusions stated and the contents written are only excerpts of the theory. This is because the theory stated here is beyond the intention of this work. The interested reader can refer to the books about the Statistical Learning Theory [Vap79], VC Theory [Vap95] and many other works on this. Here only some important subsets for Support Vector Machines of the whole theory will be shown.

Imagine points in the $\mathbb{R}^2$, which should be binary classified: class +1 or class -1. They are consistent with many classifiers (hypotheses, set of
functions). But how can one minimize the room of the hypothesis set? One approach is to apply a margin to each data point (figures 1.1 and 2.2), then, the broader that margin, the smaller the room for hypotheses is. This approach is justified by Vapnik’s learning theory.

![Figure 2.2: Reducing the room for hypothesis by applying a margin to each point](image)

The main conclusion of this technique is, that a wide margin often leads to a good generalization ability but can restrict the flexibility in some cases.

Therefore the later introduced *maximal margin* approach for Support Vector Machines is a practicable way. And this technique means that Support Vector Machines implement the principles of Structural Risk Minimization.

The actual risk of Support Vector Machines was bounded by [Vap95] alternatively. The term *Support Vectors* here will be explained in part II of this work, the bound is only stated here but is really general, because often one can see that the bound behaves in the other direction: Few Support Vectors, but high bound.
$$E[P(error)] \leq \frac{E[\text{number of support vectors}]}{\text{Numer of training samples}} \tag{2.4}$$

Where $P(error)$ is the risk for a machine trained on $l - 1$ examples, $E[P(error)]$ is the expectation over all choices of training sets of size $l - 1$ and $E[\text{number of support vectors}]$ is the expectation of the number of support vectors over all choices of training sets of size $l - 1$.

To end this sub chapter some known VC dimensions of the later introduced Support Vector Machines should be stated, but without proof:

Support Vector Machines implementing Gaussian Kernels\(^3\) have infinite VC dimension and the ones using polynomial Kernels of degree $p$ have VC dimension of $\binom{p + d_i - 1}{p} + 1^4$ where $d_i$ is the dimension where the data lives, e.g. $\mathbb{R}^2$. So here the VC dimension is finite but grows rapidly with the degree.

Against the bound of (2.3) this result is a disappointing one, because of the infinite VC dimension when using Gaussian Kernels and therefore the bound becoming useless.

But because of new developments in generalization theory, the usage of even infinite VC dimensions becomes practicable. The main theory is about Maximal Margin Bounds and gives another bound on the risk, which is even applicable in the infinite case. The theory works with a new analysis method in contrast to the VC dimension: The fat-shattering dimension.

To look in the future: The generalization performance of Support Vector Machines is excellent in contrast to other long studied methods, e.g. classification based on the Bayesian theory.

But as this is beyond this work, only a reference will be given here:

The paper “Generalization Performance of Support Vector Machines and Other Pattern Classifiers” by Bartlett and Shawe-Taylor (1999).

Now the theoretic groundwork for looking into Support Vector Machines has been laid and why they work at all.

\(^3\) See chapter 6

\(^4\) $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, called the binomial coefficient
Chapter 3

Pattern Recognition

Figure 3.1: Computer vision: Image processing and pattern recognition. The whole problem is split in sub problems to handle.

Pattern recognition is arranged into the computer vision part. Computer vision tries to “teach” the human part of noticing and understanding the environment to a machine. The main problem thereby arising is the illustration of the three-dimensional environment by two-dimensional sensors.
Definition 3.1  (Pattern recognition)
Pattern recognition is the theory of the best possible assignment of an unknown pattern or observation to a meaning-class (classification). In other words: The process of identification of objects, with help of already learned examples.

So the purpose of a pattern recognition program is to analyze a scene (mostly in the real world, with aid of an input device such as a camera, for digitization) and to arrive at a description of the scene which is useful for the accomplishment of some task, e.g. face detection or hand-written digit recognition.

3.1  Feature Extraction

This part are the procedures for measuring the relevant shape information contained in a pattern, so the task of classifying the pattern is made easy by a formal procedure. For example, in character recognition a typical feature might be the height-to-width ration of a letter. Such a feature will be useful for differentiating between a W and an I but distinguishing between E and F this feature would be quite useless. So more features are necessary or the one given above has to replaced by another. The goal of feature extraction is to find as few features as possible that adequately differentiate the pattern in a particular application into their corresponding pattern classes. Because the more features there are, the more complicated the task of classification could be, because the degree of freedom (the dimension of vectors) grows and for each new feature introduced you usually need some hundreds of new training points to get reliable statements on their derivation. To give a link to the Support Vector Machines here: The theory on feature extraction is the main problem in practice, because of the proper selection you have to define (avoid redundancy) and because of the amount of test data you have to create for training for each new feature introduced.

3.2  Classification

The step of classification is concerned with making decisions concerning the class membership of a pattern in question. The task is to design a de-
cision rule that is easy to compute and that will minimize the probability of misclassification. To get a classifier, the one decided to fulfill this step has to be trained by already classified examples, to get the optimal decision rule, because when dealing with high complexity classes the classifier will not be describable as a linear one.

![Diagram of classifier development steps](image)

**Figure 3.2:** Development steps of a classifier.

As an example consider the distinction between apples and pears. Here the a-priori knowledge is, that pears are higher than broad and apples are broader than high. So one feature would be the height-width-ratio. Another feature that could be chosen is the weight. So the picture of figure 3.3 will be gained after measurement of some examples. As it can be seen, the classifier can nearly be approximated to a linear one (the horizontal line). Other problems could consist of more than only two classes, the classes could overlap and therefore there is need of some error-tolerating scheme and the usage of nonlinearity.

There are two ways for training a classifier:

- Supervised learning
- Unsupervised learning

The technique of supervised learning uses a representatively sample, meaning it describes the classes very good. The sample leads to a classification, which should approximate the real classes in feature space. There the separation boundaries are computed.
In contrast to this, unsupervised learning uses algorithms, which analyze the grouping tendency of the feature vectors into point clouds (clustering).

Simple algorithms are e.g. the minimum distance classification, the maximum likelihood classifier or classifiers based on the Bayesian theory.

**Figure 3.3:** Training a classifier on the two-class problem of distinguishing between apples and pears by the usage of two features (weight and height-to-width ratio).
Chapter 4

Optimization Theory

As we have seen in the first two chapters, the learning task may be formulated as an optimization problem. The searched hypothesis function should therefore be chosen in a way, so the risk function is minimized. Typically this optimization problem will be subject to some constraints. Later we will see that in the support vector theory we are only concerned with the case, in which the function to be minimized/maximized, called the cost function, is a convex quadratic function, while the constraints are all linear. The known methods for solving such problems are called convex quadratic programming.

In this chapter we will take a closer look at the Lagrangian theory, which is the most adapted way to solve such optimization problems with many variables. Furthermore the concept of duality will be introduced, which plays a major role in the concept of Support Vector Machines.

The Lagrangian theory was first introduced in 1797 and it only was able to deal with functions constrained by equalities. Later in 1951 this theory was extended by Kuhn and Tucker to be adapted to the case of inequality constraints. Nowadays this extension is known as the Kuhn-Tucker theory.

4.1 THE PROBLEM

The general optimization problem can be written as a minimization problem, since reversing the sign of the function to be optimized turns it in the equal maximization problem.

**Definition 4.1 (Primal Optimization Problem)**
Given functions $f$, $g_i$ and $h_j$ defined on a domain $\Omega \subseteq \mathbb{R}^n$, the problem can be formulated:

<table>
<thead>
<tr>
<th>Minimize</th>
<th>$f(x)$ ; $x \in \Omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject to</td>
<td>$g_i(x) \leq 0$ ; $i = 1,..., k$</td>
</tr>
<tr>
<td></td>
<td>$h_j(x) = 0$ ; $j = 1,..., m$</td>
</tr>
</tbody>
</table>
where \( f(x) \) is called the \textit{objective} function, \( g_i \) the inequality and \( h_j \) the equality constraints. The optimal value of the function \( f \) is known as the \textit{value of the optimization problem}.

An optimization problem is called a \textit{linear program}, if the objective function and all constraints are linear, and a \textit{quadratic program}, if the objective function is quadratic, while the constraints remain linear.

**Definition 4.2 (Standard Linear Optimization Problem)**

\[
\begin{align*}
\text{Minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

or reformulated this means:

\[
\begin{align*}
\text{Minimize} & \quad c_1x_1 + \ldots + c_nx_n \\
\text{subject to} & \quad a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\
& \quad \ldots \ldots \\
& \quad a_{n1}x_1 + \ldots + a_{nn}x_n = b_n \\
& \quad x \geq 0
\end{align*}
\]

and another representation is:

\[
\begin{align*}
\text{Minimize} & \quad \sum c_ix_i \\
\text{subject to} & \quad \sum a_{ij}x_j = b_i, \text{ mit } i = 1\ldots n \\
& \quad x \geq 0
\end{align*}
\]

It is possible to rewrite each common linear optimization problem in this standard form, even if the constraints are given as inequalities. For further readings on this topic one can refer to many good textbooks about optimization available.

There are many ways to get the solution(s) of linear problems, e.g. Gaussian Reduction, Simplex with the Hessian Matrix, \ldots, but we will not have such problems and therefore do not discuss these techniques here.
Definition 4.3 *(Standard Quadratic Optimization Problem)*

\[
\begin{align*}
\text{Minimize} & \quad c^T x + x^T D x \\
\text{subject to} & \quad A x \leq b \\
& \quad x \geq 0
\end{align*}
\]

with Matrix $D$ overall positive (semi-) definite, so the objective function is convex. *Semi definite* means, that for each $x$, $x^T D x \geq 0$ (in other words, $D$ has non-negative eigenvalues). Non-convex functions and domains are not discussed here, because they will not play any role in the algorithms for Support Vector Machines. For further readings on nonlinear optimization, refer to [Jah96]. So in this problem you have variables $x$ in the form $x$ and $x^2$, which does not lead to a linear system, where only the form $x$ is found.

Definition 4.4 *(Convex domains)*

A subdomain $D$ of the $\mathbb{R}^n$ is convex, if for any two points $x, y \in D$ the connection between them is also an element of $D$. Mathematically this means:

\[(1-h)x + hy \in D\]

for all $h \in [0,1]$, and $x, y \in D$

For example the $\mathbb{R}^2$ is a convex domain. In figure 4.1 only the three upper domains are convex.

*Figure 4.1: 3 convex and 2 non-convex domains*
Definition 4.5 (Convex functions)

A function $f$ is said to be convex in $D \subseteq \mathbb{R}^n$, if the domain $D$ is convex and for all $x, y \in D$ and $h \in [0,1]$ this applies:

$$f(hx + (1-h)y) \leq hf(x) + (1-h)f(y)$$

In words this means, that the graph of the function always lies under the secant (or chord).

Another criterion for convexity of twice differentiable functions is the positive semi definiteness of the Hessian Matrix [Jah96].

The problem of minimizing a convex function on a convex domain (set) is known as a convex programming problem. The main advantage of such problems is the fact, that every local solution to the convex problem is also a global solution and that a global solution is always unique there. In nonlinear, non-convex problems, the main problem are the local minimums. For example, algorithms implementing the Gradient-Descent (-Ascent) method to find a minimum (maximum) of the objective function cannot guarantee, that the found minimum is a global one, and so the solution would not be optimal.

In the rest of this diploma and in the support vector theory, the optimization problem can be restricted to the case of a convex quadratic function with linear constraints on the domain $\Omega \subseteq \mathbb{R}^n$.
4.2 LAGRANGIAN THEORY

The intention of the Lagrangian theory is to characterize the solution of an optimization problem initially, when there are no inequality constraints. Later the method was extended to the presence of inequality constraints, known as the Kuhn-Tucker theory.

To ease the understanding we first introduce the simplest case of optimization in absence of any constraints.

Theorem 4.6 (Fermat)

A necessary condition for $w^*$ to be a minimum of $f(w)$ $f \in C^1$, is that the first derivation $\frac{\partial f(w^*)}{\partial w} = 0$. This condition is also sufficient if $f$ is a convex function.

Addition: A point $x_0=(x_1...x_n) \in \mathbb{R}^n$ realizing this condition is called a stationary point of the function $f: \mathbb{R}^n \to \mathbb{R}$.
To use this on constrained problems, a function, known as the Lagrangian, is defined, that unites information about both the objective function and its’ constraints. Then the stationarity of this can be used to find solutions.

In appendix C you can find a graphical solution to such a problem in two variables and the calculated Lagrangian solution to the same problem. Also an example for the general case is formulated there.

**Definition 4.7 (Lagrangian)**

Given an optimization problem with objective function \( f(w) \) and the equality constraints \( h_i(w) = c_i \), \( i = 1, \ldots, n \), the Lagrangian function is defined as

\[
L(w, \alpha) = f(w) + \sum_{i=1}^{n} \alpha_i (c_i - h_i(w))
\]

And as every equality can be transformed to \( h_i(w) = 0 \), the Lagrangian is

\[
L(w, \alpha) = f(w) + \sum_{i=1}^{n} \alpha_i h_i(w)
\]

The coefficients \( \alpha_i \) are called the **Lagrange multipliers**.

**Theorem 4.8 (Lagrange)**

A **necessary** condition for a point \( w^* \in \mathbb{R}^n \) to be a minimum (solution) of the objective function \( f(w) \) subject to \( h_i(w) = 0 \), \( i = 1 \ldots n \), with \( f, h_i \in C^1 \) is

\[
\frac{\partial L(w^*, \alpha^*)}{\partial w} = 0 \quad \text{(Derivation subject to } w) \]

\[
\frac{\partial L(w^*, \alpha^*)}{\partial \alpha} = 0 \quad \text{(Derivation subject to } \alpha) \]

These conditions are also **sufficient** in the case that \( L(w, \alpha) \) is a convex function. This means the solution is a global optimum.
The conditions provide a linear system of \( n+m \) equations, with the last \( m \) the equality constraints (See appendix C for examples). By solving this system one obtains the solution.

**Note:** At the optimal point the constraints equal zero and so the value of the Lagrangian is equal to the objective function:

\[
L(w, \alpha) = f(w^*)
\]

As an interpretation of the Lagrange multiplier \( \alpha \) of the function \( f(w) + \alpha(c - h(w)) \), we assume it as a function of \( c \) and differentiate it with respect to \( c \):

\[
\frac{\partial L}{\partial c} = \alpha
\]

But in the optimum \( L(w, \alpha) = f(w^*) \). So we can interpret that the Lagrange multiplier \( \alpha \) gives a hint on how the optimum is changing if the constant \( c \) of the constraint \( g(w) = c \) is changed.

Now to the most general case, where the optimization problem both contains equality and inequality constraints.

**Definition 4.9 (Generalized Lagrangian Function)**

The general optimization problem can be stated as

\[
\begin{align*}
\text{Minimize} & \quad f(w) \\
\text{subject to} & \quad g_i(w) \leq 0; \quad i = 1 \ldots k \quad (\text{inequalities}) \\
& \quad h_j(w) = 0; \quad j = 1 \ldots m \quad (\text{equalities})
\end{align*}
\]

Then the generalized Lagrangian is defined as:

\[
L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{m} \beta_i h_i(w)
\]

\[
= f(w) + \alpha^T g(w) + \beta^T h(w)
\]
4.3 DUALITY

The introduction of dual variables is a powerful tool, because using this alternative - the dual reformulation of an optimization problem often turns out to be easier to solve in contrast to its’ so called primal problem because the handling of inequality constraints in the primal (which are often found) is very difficult. The dual problem to a primal problem is obtained by introducing the Lagrange multipliers, also called the dual variables. So the dual function does not depend on the primal variables anymore and solving this problem is the same as solving the primal one. The new dual variables are then considered to be the fundamental unknowns of the problem. Duality is also a common procedure in linear optimization problems. For further readings look at [Mar00]. In general the primal minimization problem is then turned in the dual maximization one. So at the optimal solution point the primal and the dual function both meet with having an extreme there (convex functions would only have one global extreme).

Here we only look at the duality method important for Support Vector Machines. To transform the primal problem in its’ dual one two steps are necessary. First, the derivatives of the set up primal Lagrangian are set to zero with respect to the primal variables. Second, substitute the so gained relations back into the Lagrangian. This removes the dependency on the primal variables and corresponds to explicitly computing the new function

\[ \theta(\alpha, \beta) = \inf_{w \in \Omega} L(w, \alpha, \beta) \]

For proof of this see [Nel00].

So overall the primal minimization problem of definition 4.1 can be transformed in the dual problem as:

**Definition 4.10 (Lagrangian Dual Problem)**

Maximize \[ \theta(\alpha, \beta) = \inf_{w \in \Omega} L(w, \alpha, \beta) \]

subject to \[ \alpha > 0 \]

---

\(^5\) \(\text{Inf} = \text{infimum} \) The infimum of any subset of a linear order (linearly ordered set) is the greatest lower bound of the subset. In particular, the infimum of any set of numbers is the largest number in the set which is less than or equal to every other number in the set.

Rewritten this means: \[ \theta(\alpha, \beta) = \inf_{w \in \Omega} L(w, \alpha, \beta) = \max_{\alpha > 0} \min_{w} \min(L(w, \alpha, \beta)) \]
This strategy is a standard technique in the theory of Support Vector Machines. As seen later, the dual representation allows us to work in high dimensional spaces using so-called Kernels without “falling prey to the curse of dimensionality”. The Kuhn-Tucker complementary conditions, introduced in the following subchapter, lead to a significant reduction of the data involved in the training process. These conditions imply that only the active constraints have non-zero dual variables and therefore are necessary to determine the searched hyperplane. This observation will later lead to the term support vectors, as seen in chapter 5.

4.4 KUHN-TUCKER THEORY

Theorem 4.11 (Kuhn-Tucker)

Given an optimization problem with convex domain $\Omega \subseteq \mathbb{R}^n$:

Minimize $f(w)$  \quad $w \in \Omega$

subject to $g_i(w) \leq 0$  \quad $i = 1 \ldots k$
$h_j(w) = 0$  \quad $j = 1 \ldots m$

with $f \in C^1$ convex and $g_i, h_i$ affine, necessary and sufficient conditions for a point $w^*$ to be a optimum, are the existence of $\alpha^*, \beta^*$ such that

$$
\frac{\partial}{\partial w} L(w^*, \alpha^*, \beta^*) = 0
$$

$$
\frac{\partial}{\partial \beta} L(w^*, \alpha^*, \beta^*) = 0
$$

$$
\alpha^*_i g_i(w^*) = 0  \quad ; \quad i = 1 \ldots k
$$

$$
g_i(w^*) \leq 0
$$

$$
\alpha^*_i \geq 0
$$

---

6 Explained in chapter 6
The third relation is also known as the KT complementary condition. It implies that for active constraints, \( \alpha_i^* \geq 0 \), whereas for inactive ones \( \alpha_i^* = 0 \).

As interpretation of the complementary condition one can say, that 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 that constraint with the constraint active. So the KT conditions say that either a constraint is active, meaning \( g_i(w^*) = 0 \) and \( \alpha_i^* \geq 0 \), or the corresponding multiplier \( \alpha_i^* = 0 \).

So the KT conditions give a hint on how the solution looks like and how the Lagrange multipliers behave. And a point is only an optimal solution if and only if these KT conditions are fulfilled.

Summarizing this chapter it can be said that all the theorems and definitions above give some useful techniques for solving convex optimization problems with inequality and equality constraints both acting at the same time. The goal of the techniques is to “simplify” the primal given problem by formulizing the dual one, in which the constraints are mostly equalities which are easier to handle. The KT conditions describe the optimal solution and its’ important behaviour and will be the stopping criterion for the later implemented numerical solutions.

Later in the chapters about implementation of the solving algorithms to such optimization problems we will see that the main problem will be the size of the training set, which therefore defines the size of the kernel matrix as a solution. With the use of standard techniques for calculating the solution, the kernel matrix will fast exceed hundreds of megabytes in the memory even when the sample size is just a few thousand points (which is not much in real-world applications).