INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600



Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/6055

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

OPTIMIZATION ISSUES IN DATA ANALYSIS:

AN ANALYTIC CENTER APPROACH TO KERNEL METHODS

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

ALEXANDER M. MALYSCHEFF

Norman, Oklahoma

2001

UMI Number: 9994069

UMI®

UMI Microform 9994069

Copyright 2001 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

OPTIMIZATION ISSUES IN DATA ANALYSIS:

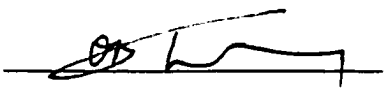
AN ANALYTIC CENTER APPROACH TO KERNEL METHODS

A Dissertation

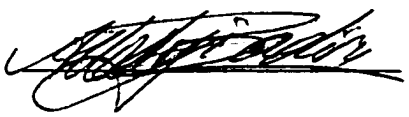
APPROVED FOR THE

SCHOOL OF INDUSTRIAL ENGINEERING

BY



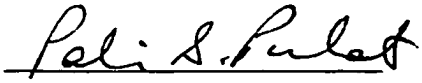
Dr. T.B. TRAFALIS



Dr. A.B. BADIRU



Dr. S. LAKSHMIVARAHAN



Dr. P.S. PULAT



Dr. S. RAMAN

©Copyright by Alexander M. Malyscheff 2001

All Rights Reserved.

Acknowledgements

I would like to thank the faculty and staff of the University of Oklahoma for having supported me throughout the course of my studies. In particular, I would like to express my gratitude to Dr. T.B. Trafalis, Dr. A.B. Badiru, Dr. S. Lakshmivarahan, Dr. P.S. Pulat, and Dr. S. Raman who have provided careful guidance and advice while serving on my committee during my doctoral studies. Special thanks to Dr. T.B. Trafalis for having served as my advisor. His expertise and guidance were of great importance for this research. Finally, I would also like to acknowledge the support of the National Science Foundation under NSF Grant ECS-9978813.

Danksagung

Es ist unmöglich, in Worten meine Dankbarkeit gegenüber meinen Eltern, Annegret Malyscheff und Rudolf Malyscheff, auszudrücken. Ohne ihre Zuneigung, Unterstützung und Grosszügigkeit hätte ich meine Studien niemals erfolgreich zum Abschluss bringen können.

A mes amis, en Europe et aux Etats-Unis, Ahmadali, Barbara, Béatrice, Carole et Carole, Erwan, Frank, Gwenn, Hélène, José, Katia, Mark, Michael, Milan, Nicolas, Olivier, Ralf, Séverine, Sy, Tanja, Volker, et Wolfgang qui ont rendu mon séjour à Norman une expérience inoubliable.

Contents

Acknowledgements iv

Danksagung v

vi

Abstract xiii

1 Introduction 1

1.1 Overview 1

1.2 Research Objectives 2

1.3 Scope of the Dissertation 3

2 Basic Issues in Machine Learning 4

2.1 Machine Learning 4

2.2 Classification and Regression 5

2.3 The Kernel Method 6

3 Support Vector Machines 9

3.1 Pattern Recognition 9

3.2 Regression 11

3.3 Machine Learning and the Concept of a Center 13

4 An Analytic Center Machine for Pattern Recognition 20

4.1 Statement of the problem 20

4.1.1 Geometry of Linear Learning Machines 21

4.1.2 Analytical Center of Version Space 23

4.2 Kernelization 26

4.3 A projected Newton descent method 31

4.4 Implementation Issues 35

4.4.1 How to process a "good" pattern 35

4.4.2 How to correct a "bad" pattern 38

5 An Analytic Center Machine for Regression Analysis 45

5.1 Statement of the problem 45

5.1.1 Geometry of Linear Learning Machines for Regression 45

5.1.2 Analytical Center of Version Space for Regression 48

5.2 Kernelization 49

5.3 Finding an initial feasible solution (Phase I) 55

5.4 A conjugate gradient method to minimize the logarithmic barrier (Phase II) 57

5.5 Regression Training: An Example 63

6 Computational Results **78**

6.1 Results for Pattern Recognition 78

6.1.1 Tests on Synthetic Data 78

6.1.2 Tests on Standard Benchmark Data 79

6.2 Results for Regression Analysis 81

6.2.1 The 'sinc' function 81

6.2.2 Benchmark tests 82

7 Summary, Conclusions and Recommendations **87**

7.1 Summary 87

7.2 Recommendations for Future Research 87

A MATLAB Code for Pattern Recognition **89**

B MATLAB Code for Regression Analysis **105**

C Standard Benchmark Data **121**

List of Tables

5.1 63

6.1 79

6.2 82

6.3 84

List of Figures

2.1 The Kernel Method for Classification 8

2.2 The Kernel Method for Regression 8

3.1 Classification in Pattern Space 14

3.2 Geometrical Interpretation of Optimization Problem *(P1)* 18

3.3 "Normal" Version Space 19

3.4 Elongated Version Space 19

4.1 Feasible Region in α-space 31

4.2 The previous solution α_{j-1} is always feasible with respect to the pattern k_j 38

4.3 The current solution α_1 is infeasible for the second pattern k_2 42

5.1 Analytic Center Machine for Regression, Linear Kernel, $\epsilon = 0.5$ 77

6.1 Analytic Center Solution 80

6.2 Support Vector Solution 81

6.3 Analytic Center Machine for Regression, 2nd Degree Polynomial Kernel, $\epsilon = 0.5$ 83

6.4 Analytic Center Machine for Regression, 6th Degree Polynomial Kernel, $\epsilon = 0.5$ 84

6.5 Analytic Center Machine for Regression, 10th Degree Polynomial Kernel, $\epsilon = 0.5$ 85

6.6 Analytic Center Machine for Regression, 15th Degree Polynomial Kernel, $\epsilon = 0.5$ 86

Abstract

Support vector machines have recently attracted much attention in the machine learning and optimization communities for their remarkable generalization ability. The support vector machine solution corresponds to the center of the largest hypersphere inscribed in the version space. Recently, however, alternative approaches [9] have suggested that the generalization performance can be further enhanced by considering other possible centers of the version space like the center of gravity. However, efficient methods for calculating the center of gravity of a polyhedron are lacking. A center that can be computed efficiently using Newton's method is the analytic center of a convex polytope [29]. We propose an algorithm, that finds the hypothesis that corresponds to the analytic center of the version space. We refer to this type of classifier as the analytic center machine (ACM). In this study ACMs have been employed to solve problems in pattern recognition and regression analysis. Preliminary experimental results are presented for which ACMs outperform support vector machines.

Chapter 1

Introduction

1.1 Overview

Support vector machines (SVMs) [31] have recently attracted much attention in the machine learning and optimization communities. As a new tool for solving problems in machine learning they are based on a simple quadratic programming approach. Hence, traditional convex optimization techniques suffice to compute the solution of the SVM learning problem. Moreover, Vapnik [32] demonstrates that using the so-called kernel method the same solution strategy applied to linear classification and regression problems can easily be extended to nonlinear instances. An overview on support vector machines can be found in [6, 27]. Currently, research in support vector machine learning focuses on methods to increase the efficiency of finding the support vector solution [10, 21] and on new implementations that allow for large-scale problems to be solved [14].

However, SVMs are not very effective in cases where the version space, i.e. the space

of hypotheses consistent with the training data, is elongated or asymmetric. Recently, an alternative approach has been proposed by Herbrich et al. [9] and Ruján [25], the so-called Bayes point machine, which has outperformed support vector machines in terms of generalization ability [5, 9].

1.2 Research Objectives

The solution of the support vector machines problem corresponds to the center of the largest inscribed hypersphere in version space. This center can be identified without difficulty, as its solution is directly found from a quadratic programming problem. Bayes point machines (BPMs) on the other hand are based on an approximation of the center of mass of the version space. BPMs outperform support vector machines, even though computing the center of gravity of a polyhedron in an n-dimensional space is hard, since it involves the computation of its volume, which is a #P-hard problem [11]. An easily computable center has been proposed by Sonnevend [29], the so-called "analytic center" of a convex polytope. This point depends on the data analytically (i.e. rather smoothly), is invariant with respect to affine transformations, and can be computed effectively by minimizing a strictly convex function over the convex polytope. Earlier, Bouten et al. [5] have investigated the generalization performance of perceptrons and found that the Bayes point solution in version space can be approximated by computing the minimum of a special class of potential functions. In this study we propose an algorithm that finds the hypothesis that corresponds to the analytic center of the version space. We employ the logarithmic barrier function as reference potential and implement analytic

center machines preserving the positive features of support vector machines such as the kernel mapping. Preliminary experimental results indicate that this new learning machine outperforms support vector machines.

1.3 Scope of the Dissertation

Chapter 2 provides a brief introduction to the basic concepts in machine learning. In the following chapter a review on support vector machines is presented spelling out the optimization models necessary to solve pattern recognition and regression analysis problems. Chapter 3 also explains the nature of a learning problem in the weight space and describes the solution of such a problem using the concept of a center. Chapter 4 introduces the reader to the analytic center machine for classification problems. Hereafter, the analytic center approach is extended in chapter 5 to problems dealing with regression analysis. Preliminary results are presented in chapter 6, while chapter 7 concludes this discussion and proposes possible directions for future research.

Chapter 2

Basic Issues in Machine Learning

2.1 Machine Learning

The fundamental concept in machine learning can be described as follows. Suppose, an experiment is conducted with a set of parameters describing the conditions, under which the experiment takes place. After having realized the experiment a particular result can be observed. The question that arises is, whether there exists a mathematical expression that could predict or at least approximate the outcome of the experiment. What type of approach should be taken, in order to answer this question? First, it seems obvious that one single experiment will not carry sufficient information to allow deduction of some mathematical expression. Hence, the experiment must be repeated, if possible under varying conditions. These conditions, under which the experiment is conducted, are generally described by a set of attributes. Each attribute in turn is represented mathematically by a variable, the input parameter. For example, if there are six attributes that characterize an experiment, we have six input parameters. For

each set of input parameters the corresponding result is observed and compared to the input parameters of the experiment. Call the variable indicating the result of the experiment the output parameter. Once a certain number of, say l, experiments has been conducted a rule must be found, which predicts with as few errors as possible the value of the output parameter for all l experiments drawing solely on information based on the input parameters. This process will be referred to as *training* , and the set of experiments used to find the mathematical function linking input and output data is called the training set. In other words, during training an unknown system is presented with a set of input data, for which the result of the experiment, the output, is known. The training experiments are used to "learn", how the system behaves, extracting what is called the discriminant function f. Once training has been concluded and a good discriminant function has been found, the *testing* phase can begin, where the outcome of a given set of input parameters is indeed unknown. Obviously, the goal is to commit as few errors as possible during testing. Hence, a discriminant function is needed, which will be able to adopt as many vital characteristics of the experiment as possible during training and translate these properties into a low rate of failure during testing. Mathematically, the training set is described by $(x_j, y_j), j = 1, ...l$, where the x_j correspond to the input parameters and the y_j refer to the output parameters.

2.2 Classification and Regression

Learning machines can be implemented in two different ways. In *pattern classification problems* the output parameter y_j is a categorical variable that indicates to which class

a given combination of input parameters x_j belongs. Here, the output parameter is also referred to as label. Most applications in classification problems deal with two different classes. Therefore, the labels $y_j = 1$ and $y_j = -1$ are selected to distinguish between classes A and B. More formally, let each of the l input vectors be represented by $x_j \in \Re^d$, where $j = 1, ..., l$ and observe the corresponding labels $y_j \in \{-1, +1\}$. Application examples for pattern classification comprise breast cancer diagnosis [15], optical character recognition [7], or computer vision [19].

Contrary to classification problems the objective of *regression problems* is to approximate a sequence of scalar values $y_j \in \Re$, $j = 1, ..., l$. For the input vector holds $x_j \in \Re^d$, however, in many applications the input parameters reduce to a set of one-dimensional scalars. A field of application can be found in the regression analysis and prediction of the value of financial securities. In [17] promising results for time-series analysis were reported.

2.3 The Kernel Method

Most machine learning algorithms are limited to linear discriminant functions. Hence, if for example a certain classification experiment displays a nonlinear separating surface, many algorithms (e.g. the perceptron) will not be able to account for this nonlinear behavior. For support vector machines the kernel method was first discussed in [4]. Using a function $\phi : x \rightarrow \phi(x)$ the problem can be transformed from the input or pattern space to some high-dimensional feature space. The mapping creates the corresponding classification problem in feature space allowing linear separation of the problem data.

Hence, linear classification methods can be employed in feature space to solve problems that are nonlinear in input space. The price we have to pay is to significantly increase the number of dimensions due to the transformation. A similar approach can be performed for regression problems.

Unfortunately, very often the function $\phi(x)$ is not available, can not be computed, or does not even exist. However, the inner product of two vectors can be computed, both, in pattern and in feature space. In other words, while $\phi(x)$ might not be available, the inner product $\phi(x_1)^T \phi(x_2)$ can still be computed in feature space. This inner product can be expressed by the kernel function

$$k : \Re^d \times \Re^d \to \Re : k(x_1, x_2) = \phi(x_1)^T \phi(x_2) .$$

In order to employ the kernel method, it is necessary to express the data vectors x_j in terms of inner products. Consequently, the constraints describing the classification or regression problem have to be reformulated, such that solely inner products are used. These inner products are then mapped yielding a linear regression problem or a linear classification problem in feature space. Figure 2.1 illustrates the process of transforming the problem data from input space to feature space in classification problems.

A similar transformation can be conducted for regression problems as shown in Figure 2.2. Possible kernel functions include polynomial kernels such as $k(x_1, x_2) = \left(x_1^T x_2 + 1\right)^p$ with p indicating the degree of the polynomial or exponential kernels as for example $k(x_1, x_2) = \exp^{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}}$. For more information about kernel functions see Cristianini and Shawe-Taylor [8].

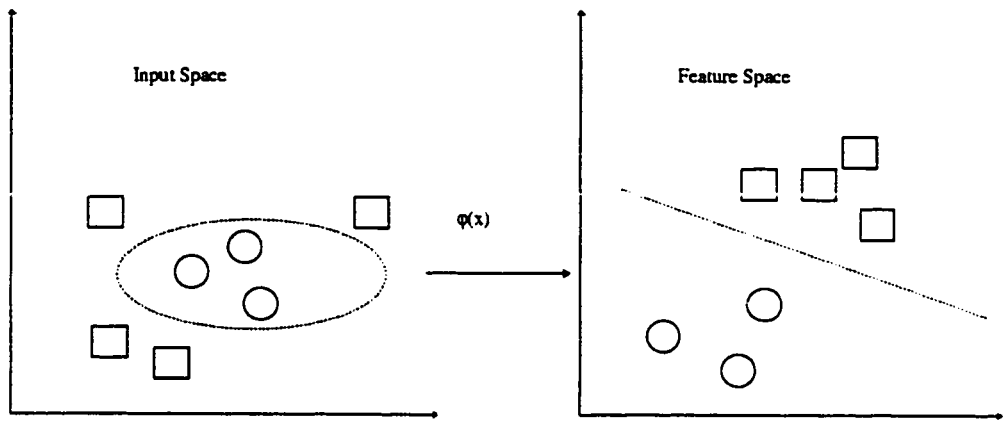


Figure 2.1: The Kernel Method for Classification

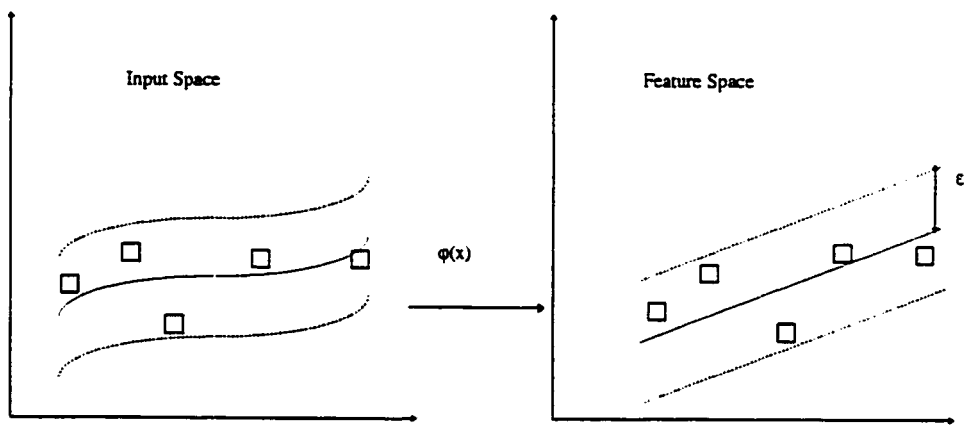


Figure 2.2: The Kernel Method for Regression

Chapter 3

Support Vector Machines

3.1 Pattern Recognition

Support vector machines can be implemented in two different environments. Specifically, regression analysis and pattern classification. In this section support vector machines in pattern recognition will be discussed, while the following paragraph will elaborate on support vector machines in regression. Both instances can be represented by a quadratic programming problem.

Consider first the classification scenario. Let the training data consist of l points, each of which is represented by a vector $x_j \in \Re^d$, where $j = 1, ..., l$, and assign to each of the points a label y_j with $y_j \in \{-1, +1\}$. Here, we only distinguish between two classes. Hence the training set can be written as $T = \{(x_j, y_j)_{j=1}^{l}\} \subset \Re^d \times \{-1, +1\}$. Constructing a learning machine that will separate the data of two training sets in pattern space finds its geometrical equivalent in identifying a hyperplane that separates the data points labeled by $y_j = -1$ from the data points labeled by $y_j = +1$. As there

are situations, where data sets can not be linearly separated, we will restrict ourselves for the moment to those problem sets that are linearly separable. This type of classifier is often also referred to as the hard margin classifier. Vapnik [31] has shown that the linear support vector machine algorithm requires a solution of the following optimization problem:

$$\begin{aligned} &(P) \qquad \min \frac{1}{2}\|w\|^2 \qquad (3.1)\\ &\textit{subject to}\\ &y_j\,(x_j \cdot w + b) \geq +1 \quad \forall j = 1, ..., l, \end{aligned}$$

where $w \in \Re^d$ is the vector normal to the separating hyperplane and $b \in \Re$ the offset with respect to the origin. By assigning a Lagrangian multiplier μ_j to each constraint and by introducing the variables $\Gamma^T = (\mu_1, \mu_2, ..., \mu_l)$, $D_{ij} = (y_i y_j x_i x_j)$, $e = (1, 1, ..., 1)$, and $y = (y_1, y_2, ..., y_l)$, the dual problem can be formulated in closed form:

$$\begin{aligned} &(D) \qquad \max +\Gamma^T e - \frac{1}{2}\Gamma^T D \Gamma \qquad (3.2)\\ &\textit{subject to}\\ &\Gamma^T y = 0\\ &\Gamma_j \geq 0 \quad \forall j = 1, ..., l. \end{aligned}$$

A nonlinear separable problem in pattern space becomes a linearly separable problem in feature space using the concept of a kernel function [31]. As can be seen, the dual problem (D) allows expressing all data pertinent to the problem, x_j and y_j, to be expressed

in terms of inner products. Hence, in order to allow nonlinear problems to be solved, the only change that has to be made is rewriting the matrix $D_{ij} = (y_i y_j \cdot k(x_i, x_j))$. For further information on support vector machines in classification the reader might also refer to [6].

Support vector machines result in a rather simple quadratic programming problem. Since the problem is convex, there are no local minima and various optimization algorithms will be able to identify the optimal solution. The name support vector is derived from those points in the input space, which touch ("support") the decision function. More formally, if the Lagrangian multiplier μ_j is nonzero, the corresponding data sample (x_j, y_j) fulfills the primal constraint in (3.1) as an equality. Thus, all samples that yield a strictly positive μ_j are support vectors. For this reason, the matrix D is rather sparse in nature for problems that do not yield many support vectors.

One of the setbacks of this implementation approach is that the matrix D will have the size $l \times l$, where l is the number of data points. Remember that many problems might include well above one million data samples [28, 14]. In that case we would have to deal with 10^{12} elements in D. Currently, a lot of research is being conducted to investigate ways to decompose problem (D) and solve subproblems, where the matrix D is of smaller size [12, 10, 21].

3.2 Regression

In this section the primal and dual optimization problems for support vector machine regression will be presented. Let the training data consist of l vectors $x_j \in \Re^d$, where

$j = 1, ..., l$. During training the input parameters are mapped to l real output parameters y_j with $y_j \in \Re$. Hence the training set can be written as $T = \{(x_j, y_j)_{j=1}^{l}\} \subset \Re^{d+1}$. Vapnik [31] has shown that the support vector machine regression translates into the following primal optimization problem:

$$\begin{aligned} &(PR) \qquad \min \frac{1}{2}\|w\|^2 \qquad (3.3) \\ &\textit{subject to} \\ &y_j - w^T x_j - b \leq \epsilon \quad \forall j = 1, ..., l, \\ &w^T x_j + b - y_j \leq \epsilon \quad \forall j = 1, ..., l, \end{aligned}$$

where $w \in \Re^d$ is the slope and $b \in \Re$ the offset with respect to the origin. Note that for d-dimensional input parameters $w \in \Re^d$. The variable ϵ can be interpreted as the precision that is required from the regression function. Geometrically, it creates a tube of width 2ϵ around the regression function, within which all measured data samples (x_j, y_j) must be contained. Note that for too small values for ϵ the optimization problem might turn infeasible.

Again, let μ_j be the Lagrangian multiplier corresponding to the first set of constraints and μ_j^* be the Lagrangian multiplier corresponding to the second set. Define $\Pi^T = (\mu_1 - \mu_1^*, \mu_2 - \mu_2^*, ..., \mu_l - \mu_l^*)$, $D_{ij} = (x_i x_j)$, $e = (1, 1, ..., 1)$, and $y = (y_1, y_2, ..., y_l)$. Then the dual problem can be formulated as follows:

$$(DR) \qquad \max -\frac{1}{2}\Pi^T D \Pi - \epsilon \sum_{j=1}^{l} \left(\mu_j + \mu_j^*\right) + y^T \Pi \qquad (3.4)$$

$$\textit{subject to}$$

$$\Pi^T e = 0$$

$$\mu_j \geq 0 \quad \forall j = 1, ..., l$$

$$\mu_j^* \geq 0 \quad \forall j = 1, ..., l.$$

As in the case of the classification approach the kernel method can be applied to solve nonlinear regression problems in feature space [31]. The matrix D has to be reformulated accordingly, $D_{ij} = (k(x_i, x_j))$. Additional information regarding support vector machine regression can be found in [27].

3.3 Machine Learning and the Concept of a Center

Previously, it was mentioned that the support vector machine solution corresponds to the center of the largest inscribed hypersphere in version space and that for certain types of problems this solution might not be satisfying. In this section we will elaborate on how to make the connection between the input space and the version space and explain the differences between the support vector machine and the approach taken here. We will provide an intuitive argument why alternative centers in version space have a strong potential of yielding a better generalization performance. In order to do so consider first the classification problem depicted in Figure 3.1.

Two classes are to be separated by a hyperplane with normal vector $\tilde{w}$. Both classes consist of two points (vectors) indicated by circles for class $+1$ and dots for class -1. The two classes are separated by a hyperplane, which we will assume for the moment passes through the origin of the input space. Furthermore, two additional hyperplanes

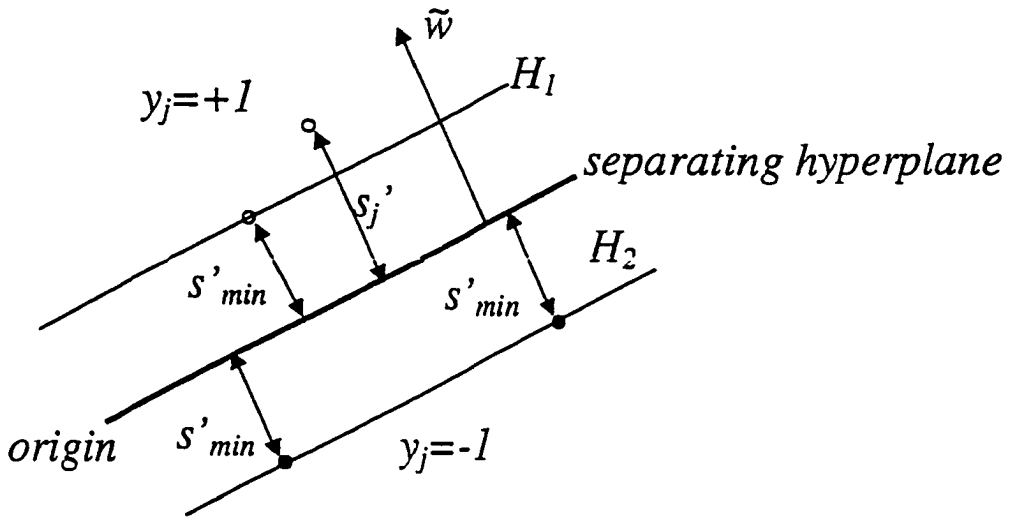


Figure 3.1: Classification in Pattern Space

H_1 and H_2 can be constructed, which are parallel to the separating hyperplane and which contain the vectors of each class that are closest to the separating hyperplane. In order to achieve a good separation, it is desirable to "push" the two hyperplanes H_1 and H_2 as far apart as possible to allow for a large "buffer zone" between the two classes. For each vector in both classes the distance from the separating hyperplane can be computed. Let the distance measure for the j-th vector be denoted by the expression $\tilde{s}_j'$. Since we require that the separating hyperplane go through the origin, the distance for any vector from that plane can be calculated from

$$\tilde{s}_j' = y_j \tilde{w}_N^T \tilde{x}_j, \tag{3.5}$$

where $\tilde{w}_N$ represents a normal vector of unit length and $\tilde{x}_j$ describes the position of the sample point. Moreover, for any separating hyperplane we can introduce a minimum

distance defined by those vectors which lie closest to the separating hyperplane:

$$\tilde{s}'_{min} = \min\{\tilde{s}'_j\} = \min\{y_j \tilde{w}_N^T \tilde{x}_j\}. \tag{3.6}$$

The distance between the two hyperplanes H_1 and H_2 equals $2\tilde{s}'_{min}$ and it is our objective to maximize this distance. Therefore, this classification problem can be formulated using the following optimization approach *(P1)* .

$$\begin{aligned} \textit{(P1)} \qquad & \max\ \tilde{s}'_{min} = \max\{\min\{\tilde{s}'_j\}\} = \max\{\min\{y_j \tilde{w}_N^T \tilde{x}_j\}\} \\ & \textit{subject to} \\ & y_j \tilde{w}_N^T \tilde{x}_j \geq \tilde{s}'_{min} > 0 \\ & \|\tilde{w}_N\| = 1, \end{aligned} \tag{3.7}$$

where $\tilde{w}_N = \frac{\tilde{w}}{\|\tilde{w}\|}$. Note that $\tilde{s}'_{min}$ must be strictly positive to ensure separation of the two classes. Since the unit length normal vector is used here, we automatically have the constraint $\|\tilde{w}_N\| = 1$. Next, let us replace $\tilde{w}_N$ by $\tilde{w}$, such that

$$\begin{aligned} \max\ \tilde{s}'_{min} = \max\{\min\{\tilde{s}'_j\}\} = \max\{\min\{y_j \frac{\tilde{w}^T}{\|\tilde{w}\|} \tilde{x}_j\}\} \\ \textit{subject to} \\ y_j \frac{\tilde{w}^T}{\|\tilde{w}\|} \tilde{x}_j \geq \tilde{s}'_{min} > 0. \end{aligned}$$

If we define $\tilde{s}_{min} = \min\{y_j \tilde{w}^T \tilde{x}_j\}$, we can further reformulate the optimization problem:

$$\max\ \tilde{s}'_{min} = \max\{\min\{y_j\tilde{w}^T\tilde{x}_j\} \cdot \frac{1}{\|\tilde{w}\|}\} = \max\frac{\tilde{s}_{min}}{\|\tilde{w}\|}$$

subject to

$$y_j\tilde{w}^T\tilde{x}_j \geq \tilde{s}'_{min}\|\tilde{w}\| > 0.$$

Next, normalize the minimum distance such that it is of unit length and realize (see the objective function) that $\tilde{s}'_{min}\|\tilde{w}\| = \tilde{s}_{min}$. Using $\tilde{s}_{min} = 1$ the optimization problem reduces to

$$\max\ \frac{1}{\|\tilde{w}\|}$$

subject to

$$y_j\tilde{w}^T\tilde{x}_j \geq 1.$$

Finally, substituting $\tilde{w}^T = \left[w^T, b\right]$ and $\tilde{x}_j^T = \left[x_j^T, 1\right]$ we can simplify the problem one more time. Introducing optimization problem *(P2)* we have:

$$\textit{(P2)} \qquad \min\ \frac{1}{2}\|w\|^2 \tag{3.8}$$

subject to

$$y_j\left(w^T x_j + b\right) \geq 1.$$

It can be seen that problem *(P2)* corresponds to the support vector machine problem for classification outlined in (3.1), thus, the original problem *(P1)* describes in fact also the support vector machine problem. The two approaches differ solely in terms of

scaling. Problem *(P1)* is solved on the unit sphere adjusting the minimum distance of the data points (slacks) $\tilde{s}'_{min}$, while problem *(P2)* requires $\tilde{s}_{min}$ to have unit length leaving the radius of the sphere as a variable. In order to further illustrate the nature of these optimization problems we will elaborate in the following on the geometrical interpretation of *(P1)* . This will be achieved by describing the classification problem in the space of all $\tilde{w}_N$'s. First, notice that all feasible $\tilde{w}_N$'s in *(P1)* can be found on the unit sphere. Moreover, the expression $y_j \tilde{w}_N^T \tilde{x}_j > 0$ can be interpreted as a halfspace defined by a hyperplane with normal vector $y_j \tilde{x}_j$. Notice that a hyperplane described in such a way passes through the origin in the space of $\tilde{w}_N$. Suppose next that there are l data points in the input space. These l data points will generate l hyperplanes, which in turn will define l halfspaces in the space of $\tilde{w}_N$. Each of these hyperplanes will pass through the origin reducing the set of feasible solutions to a subset of the unit sphere. This subset of the unit sphere defines the set of feasible solutions in the space of $\tilde{w}_N$. In the literature this set is often also referred to as the version space. Figure 3.2 displays a feasible region for four patterns.

Next, we will discuss the shape of the version space and compare the solutions of the support vector machine and the analytic center machine. Remember that it was shown that *(P1)* describes the support vector machine solution. It can be seen that problem *(P1)* extracts the center of the largest inscribed hypersphere in the version space as the solution vector. As we are attempting to maximize the minimum distance, we might move away from one separating hyperplane in Figure 3.2, however, in doing so we might approach another separating hyperplane, which will then define another $\tilde{s}'_{min}$. Obviously, the optimization problem will finally identify the center of the largest

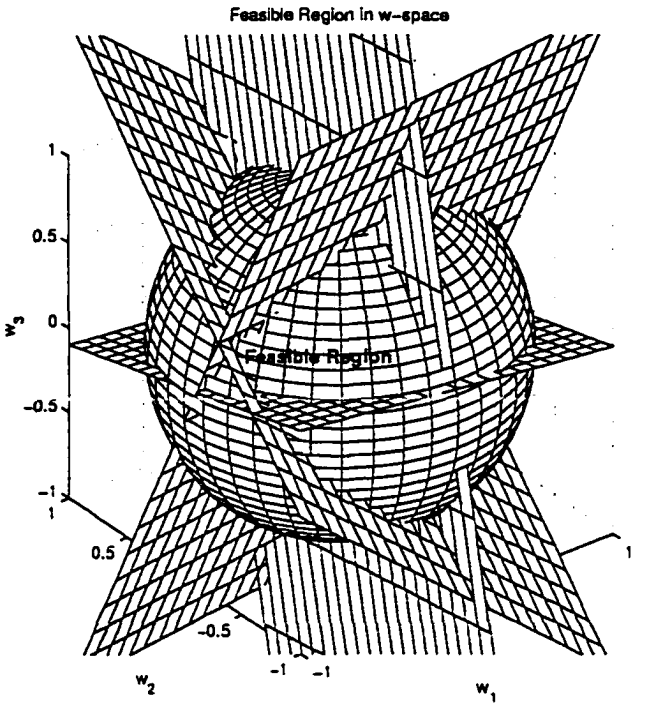


Figure 3.2: Geometrical Interpretation of Optimization Problem *(P1)*

inscribed hypersphere as the best possible solution. Moving in any direction from that point will result in a nonoptimal value for $\tilde{s}'_{min}$. This approach will yield satisfying results as long as the version space is "normal" and not elongated. Figure 3.3 illustrates a situation, where four hyperplanes define a feasible region, which is not elongated. Keeping in mind that the situation depicted in Figure 3.3 must be actually projected on the unit sphere it can be seen that the solutions for the support vector machine $\tilde{w}_{N,SVM}$ and the analytic center machine $\tilde{w}_{N,ACM}$ are almost identical.

This situation, however, changes when elongated version spaces are considered. Figure 3.4 might serve as an example for this case. As before, there are four hyperplanes defining the set of feasible solutions on the unit sphere. The support vector machine again finds the center of the largest inscribed hypersphere of the version space. Nonetheless, it can be seen that the point $\tilde{w}_{N,SVM}$ does not yield a satisfying solution for this

learning problem. On the other hand the analytic center of the version space $\tilde{w}_{N,ACM}$ will serve as a much better reference point to reflect the available problem information.

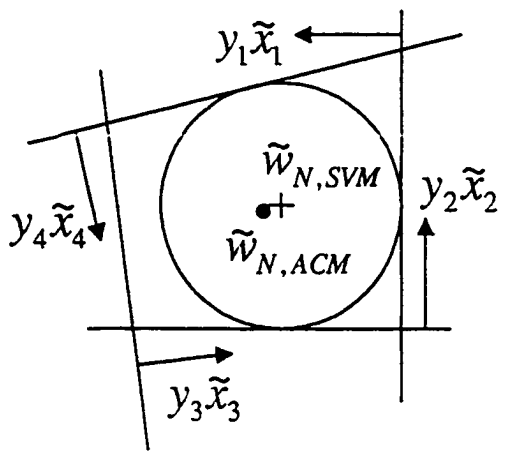


Figure 3.3: "Normal" Version Space

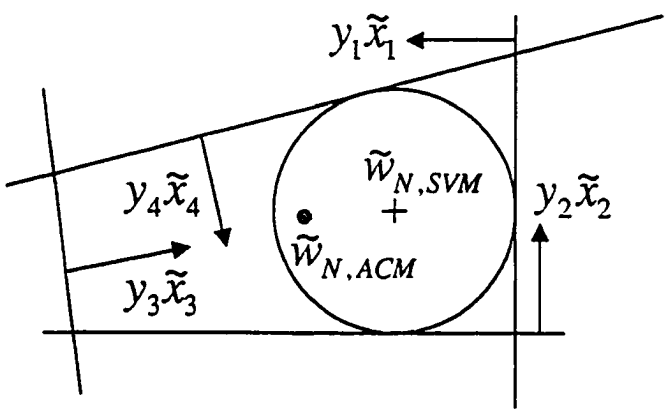


Figure 3.4: Elongated Version Space

Chapter 4

An Analytic Center Machine for Pattern Recognition

4.1 Statement of the problem

In the previous chapter the support vector machine approach was discussed and an intuitive argument was provided for considering as an alternative the solution that corresponds to the analytic center of the version space. In this chapter we will describe how the analytic center of the version space can be computed for pattern classification problems. Unfortunately, computing the analytic center often suffers from the difficulty of finding an initial feasible solution, when all constraints that define the feasible region are considered simultaneously. An alternative approach would be to deal with every sample point (constraint) at a time. Hence, the problem is solved for the first data point, then the next data point enters the system and another optimization problem is solved using the previous solution and so forth. This iterative way of finding the analytic center

of the problem can be interpreted as an online approach. Moreover, recent research has suggested to verify the generalization ability of support vector machines, as some results point to alternative learning machines with powerful generalization performance [9].

In order to implement this online approach, the support vector algorithm will be modified reducing the constraints of problem (P) essentially to what is known as the perceptron. Then, turning our attention to the weight space, it can be seen that each pattern will correspond to a hyperplane with normal vector $yx = (yx_1, yx_2, ..., yx_d)$, if we are in a d-dimensional space. Moreover, after a slight modification we will be able to let all hyperplanes corresponding to examples pass through the origin. Therefore, the collection of all patterns, i.e. l hyperplanes, will reduce the region containing acceptable results for w to a cone in a d-dimensional space. Some studies refer to this cone as the version space $V(T)$ [16]. While the perceptron allows any of these feasible w's to be selected as a solution, our objective will be to choose a w^* which remains far away from all the edges of the cone, since we do not know how future test patterns will scatter. This refers to as a large margin classifier [26].

4.1.1 Geometry of Linear Learning Machines

We would like to construct a learning machine based on the concept of the analytic center, which behaves also as a large margin classifier. Later, it will be shown that a similar learning machine can be derived for the feature space using the concept of kernel functions. In order to convey the basic ideas, we review the concepts that characterize the perceptron. In contrast to support vector machines no optimization process is invoked and the right-hand-side of the constraints becomes zero. Specifically in the

perceptron algorithm it is our objective to solve the feasibility problem:

$$y_j \cdot (x_j \cdot w + b) \geq 0 \quad \forall j = 1, ..., l, \tag{4.1}$$

where $x_j, w \in \Re^d$ and $y_j, b \in \Re$. Any feasible solution (w, b) of (4.1) will separate the training set T. Thus, for linearly separable problems the perceptron will arbitrarily identify one out of infinitely many linear classifiers and generalization will be rather poor in most instances. We will return to the issue of selecting a good hyperplane in section 4.1.2. For the moment let us slightly rewrite the set of constraints. By introducing the variables $\tilde{w} = (w, b) \in \Re^{d+1}$ and $\tilde{x}_j = (x_j, 1) \in \Re^{d+1}$ the perceptron problem simplifies to:

$$y_j \cdot (\tilde{x}_j \cdot \tilde{w}) \geq 0 \quad \forall j = 1, ..., l. \tag{4.2}$$

Essentially, b is now being considered as an additional weight, say $\tilde{w}_{d+1}$. Therefore, the problem dimension is expanded by one. As can be seen each pattern j is now described by the simple linear inequality

$$y_j \tilde{x}_j \cdot \tilde{w} \geq 0 \quad \forall j = 1, ..., l, \tag{4.3}$$

which corresponds in weight space to a hyperplane with normal vector $y_j \tilde{x}_j$. Note that all hyperplanes are passing now through the origin of the weight space.

When the first pattern (x_1, y_1) enters in the system of linear inequalities, the set of feasible solutions is reduced to a $(d+1)$-dimensional halfspace. The second pattern

(x_2, y_2) will reduce the halfspace to a cone with the apex of the cone at the origin. After that, each additional pattern will either leave the current version space unchanged or even further reduce it. Once all patterns have been learned, the cone of the training patterns has been identified. Note that some training points (x_j, y_j) might be redundant for the determination of the cone. The set of all $\tilde{w}$'s inside the cone, $V(T)$, constitutes the set of feasible solutions; hence infinitely many solutions exist. This situation corresponds to the perceptron in pattern space. Keep in mind that here we consider problems, which can be linearly separated (hard margin classifier). In this case the set of feasible $\tilde{w}$'s will never be empty. The advantage of this approach lies in the nice geometry of the problem, which allows a rather simple implementation. Once the final cone is identified, it is intuitively clear that we would like to identify a solution as close to the central trajectory of the cone as possible. In doing so, we guarantee that the selected optimal $\tilde{w}^*$ will correspond to a separating hyperplane in the space of patterns (x-space) that lies far away from the data samples of both classes. The next question to be addressed is, how to select the optimal $\tilde{w}^*$ from all feasible $\tilde{w}$'s in the version space.

4.1.2 Analytical Center of Version Space

Next, we will use an idea that stems from the concept of interior point methods, the so-called analytic center. In order to describe the analytic center, we first introduce the concept of logarithmic barrier functions. Define the slack $\tilde{s}_j \in \Re$ as a measure of how close or how far away the current solution is from constraint j. If the current solution violates constraint j, $\tilde{s}_j$ will be negative. If the current solution fulfills constraint j as an equality, $\tilde{s}_j$ will be zero. An *interior* point of the feasible region is defined as a point,

for which all slacks $\tilde{s}_j$ are positive. For the set of constraints, which currently describe our feasible region the slacks can be expressed as follows:

$$\tilde{s}_j = y_j \tilde{x}_j \cdot \tilde{w} \geq 0 \quad \forall j = 1, ..., l. \tag{4.4}$$

Furthermore, we need to identify a function, called potential function [18], which goes to infinity as we approach the boundary of the feasible region. A logarithmic function $\Phi : \Re^+ \rightarrow \Re$ fulfills our requirements. More specifically,

$$\Phi(\tilde{s}) = -ln(\tilde{s}). \tag{4.5}$$

In analogy to electrostatics the function $\Phi(\tilde{s})$ could describe the potential of some electrostatic field, if we consider the hyperplanes defining the feasible region as infinite plates of electric charge. For l constraints the logarithmic barrier function Φ becomes:

$$\Phi(\tilde{s}) = -\sum_{j=1}^{l} ln(\tilde{s}_j), \quad \tilde{s}^T = (\tilde{s}_1, ..., \tilde{s}_l). \tag{4.6}$$

Here, the vector $\tilde{s} \in \Re^l$ represents the collection of slacks for all l constraints, $\tilde{s} = (\tilde{s}_1, \tilde{s}_2, ..., \tilde{s}_l)$. The minimum of the logarithmic barrier function $\Phi(\tilde{s})$ constitutes an excellent means of computing a point that will be far away from all constraints. The point at which the barrier function attains its minimum is defined as the *analytic center* of the feasible region.

Now, let us return to the problem we would like to solve. We consider an augmented weight space (keep in mind that the offset b is treated as the $(d+1)$-th element of $\tilde{w}$),

in which the set of all patterns $(\tilde{x}_j, y_j)$ is represented by hyperplanes passing through the origin. After all hyperplanes have been defined, a cone in $\Re^{d+1}$ will describe the set of feasible solutions in the augmented weight space. It is our objective, to retrieve a weight vector $\tilde{w}^*$, which will be as far away from the "walls" of the cone as possible. The tool we are going to employ is the concept of the analytic center. Unfortunately, the minimum of the logarithmic barrier function can only be calculated, if the feasible region is compact, which is decidedly not the case for a cone in $\Re^{d+1}$. In order to compactify the feasible region, we add as a constraint a hypersphere in $\Re^{d+1}$ with radius R around the origin. In other words, the solution, which will lie without doubt very close to the central trajectory of the cone, will be projected onto a hypersphere of radius R.

Therefore, our goal is to calculate the minimum of $\Phi(\tilde{s})$ while making sure at the same time that the optimal weight vector $\tilde{w}^*$ satisfy the spherical constraint. For calculation purposes we choose $R = \sqrt{2}$. Taking into account the labels y_j as well, the slacks can be expressed through

$$\tilde{s}_j = y_j \cdot (\tilde{x}_j \cdot \tilde{w}) \geq 0 \quad \forall j = 1, ..., l \tag{4.7}$$

and the logarithmic barrier function becomes

$$\Phi(\tilde{s}) = -\sum_{j=1}^{l} ln(\tilde{s}_j). \tag{4.8}$$

Following the above, the optimal weight vector can be computed by solving the optimization problem:

$$\begin{aligned} \min\ \Phi(\tilde{w}) &= -\sum_{j=1}^{l} ln\left(y_j\left(\tilde{x}_j \cdot \tilde{w}\right)\right) \\ \text{s.t. } \frac{1}{2}\tilde{w}^T\tilde{w} &= 1. \end{aligned} \tag{4.9}$$

Since the slacks depend on $\tilde{w}$, we will write from now on $\Phi(\tilde{w})$ or in feature space $\Phi(\alpha)$, instead of $\Phi(\tilde{s})$. Remember that by definition $\tilde{w}$ also includes the offset b of the optimal separating hyperplane. Before we will continue our discussion and demonstrate, how the optimization problem in (4.9) can be solved, we will extend the current formulation using the concept of kernel functions to the case of nonlinear separation.

4.2 Kernelization

In the previous section the classification problem was reformulated, such that each pattern can be interpreted as a hyperplane in weight space. A first optimization model was presented allowing linear classification problems to be solved. We will now perform yet another transformation moving the problem to some high-dimensional feature space. The algorithm under consideration actually does not solve nonlinear problems. However, it is possible to map classification problems that can not be linearly separated to the feature space using a function $\phi: x \rightarrow \phi(x)$. The mapping creates the corresponding classification problem in feature space, which can be solved employing linear classifiers. The price we have to pay is to significantly increase the number of dimensions. Thus a nonlinear problem behaves in feature space as if it was linearly separable.

Unfortunately, very often the function $\phi(x)$ is not available, can not be computed, or does not even exist. However, the inner product of two vectors can be computed, both,

in pattern and in feature space. In other words, while $\phi(x)$ might not be available, we can still compute the inner product $\phi(x_1)^T\phi(x_2)$ in feature space. This inner product can be expressed by the kernel function

$$k : \Re^d \times \Re^d \to \Re : k(x_1, x_2) = \phi(x_1)^T\phi(x_2).$$

The reader might verify that besides the label y_j in equation (4.2) in section 4.1 every constraint can be formulated using solely inner products. Nonetheless, the constraints still contain the weight vector, while it would be desirable to express the inner products only through the data vectors x_j. However, if we express the weight vector by a linear combination of data vectors, we can successfully avoid this problem. Indeed, if we define $w = \sum_{i=1}^{l} \alpha_i x_i$ we have:

$$k(w, x_j) = k\left(\sum_{i=1}^{l} \alpha_i x_i, x_j\right) = \left(\sum_{i=1}^{l} \alpha_i \phi(x_i)\right)^T \phi(x_j) = \sum_{i=1}^{l} \alpha_i k(x_i, x_j) \qquad (4.10)$$
$$\forall j = 1, ..., l.$$

Therefore, the set of l equations in (4.1) defining hyperplanes in weight space can be reformulated using inner products. Specifically,

$$y_j\left(x_j \cdot \sum_{i=1}^{l} \alpha_i x_i + b\right) = y_j\left(\sum_{i=1}^{l} \alpha_i x_i^T x_j + b\right) \geq 0 \quad \forall j = 1, ..., l. \qquad (4.11)$$

After having preprocessed the set of constraints, it is now easy to replace the pure inner products by the kernel function $k(x_i, x_j)$. Therefore, the set of constraints representing the l patterns can be described in the dual space of α's as follows:

$$y_j \left(\sum_{i=1}^{l} \alpha_i k\left(x_i, x_j\right) + b \right) \geq 0 \quad \forall j = 1, ..., l. \tag{4.12}$$

Obviously, this new set of linear inequalities must be solved in the space of α's. Note, that the decision function to test for a new pattern x can be formulated and calculated based only on the information contained in the set of α's:

$$f\left(x\right) = sign\left(\sum_{i=1}^{l} \alpha_i k\left(x_i, x\right) + b \right). \tag{4.13}$$

We are now ready to define our learning problem in α-space. Define the matrix $K \in \Re^{(l+1)\times l}$ as:

$$K = \begin{bmatrix} y_1 k_{11} & y_2 k_{12} & \cdots & y_l k_{1l} \\ y_1 k_{21} & y_2 k_{22} & \cdots & y_l k_{2l} \\ \cdots & \cdots & & \cdots \\ y_1 k_{l1} & y_2 k_{l2} & \cdots & y_l k_{ll} \\ y_1 & y_2 & \cdots & y_l \end{bmatrix}. \tag{4.14}$$

We follow the convention that the k_{ij} represent the kernel evaluated at the points x_i and x_j. Specifically, $k_{ij} = k\left(x_i, x_j\right)$. Note that K is not symmetric. Moreover, introduce the vectors

$$k_j = \begin{bmatrix} y_j k_{1j} \\ y_j k_{2j} \\ \dots \\ y_j k_{lj} \\ y_j \end{bmatrix}, \forall j = 1, ..., l, \tag{4.15}$$

where $k_j \in \Re^{l+1}$. Therefore the matrix K can also be expressed as $K = (k_1, k_2, ..., k_l)$. Furthermore, if we let the vector $\alpha \in \Re^{l+1}$ to be

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_l \\ b \end{bmatrix}, \tag{4.16}$$

we can write the set of equations for all l patterns as $K^T\alpha \geq 0,\ \forall j = 1, ..., l$. Note that the offset b is included as the $(l+1)$-element and the corresponding adjustments were made for K and k_j. Alternatively, the j-th constraint in α-space can be defined by $k_j^T\alpha \geq 0,\ \forall j = 1, ..., l$, since

$$k_j^T\alpha = y_j\alpha_1 k(x_1, x_j) + y_j\alpha_2 k(x_2, x_j) + ... + y_j\alpha_l k(x_l, x_j) + y_j b \geq 0. \tag{4.17}$$

Replacing the kernel expressions by simple inner products, equivalence between equation (4.17) and the more commonly known perceptron formulation becomes more obvious as is described by the following inequality:

$$y_j\alpha_1 x_1^T x_j + y_j\alpha_2 x_2^T x_j + ... + y_j\alpha_l x_l^T x_j + y_j b = y_j \cdot \left(\sum_{i=1}^{l} \alpha_i x_i^T \right) x_j + y_j b = y_j \cdot \left(w^T x_j + b \right) \geq 0. \tag{4.18}$$

Since it is necessary to rewrite the optimization problem from equation (4.9) for the space of α's, computation of the slacks for each constraint is required. We have l slacks, which can be computed from $s_j = k_j^T \alpha, \ \forall j = 1, ..., l$. Previously, each pattern was characterized by its normal vector $y_j \tilde{x}_j$. This vector defined a hyperplane in the space of weights reducing the final feasible region to a cone in $\Re^{d+1}$. We have now moved the problem to the space of α's. Therefore, the normal vector defining each hyperplane in α-space is given by k_j. Note that the offset is included in k_j, consequently, each hyperplane passes again through the origin; this time, however, in the space of α's. The logarithmic barrier function evaluates the summation over all slack values, which are given by $s_j = k_j^T \alpha, \ \forall j = 1, ..., l$. Hence, the barrier is a function of α, i.e. $\Phi = \Phi(\alpha)$. Again, as in section 4.1, the feasible region has the shape of a cone, which implies that the analytic center without any projection or box limitations imposed on the variables, does not exist. The analytic center of the barrier function is therefore calculated with the additional requirement that the set of variables (the α's) be projected on a hypersphere of radius $R = \sqrt{2}$. Consequently, we have to solve the following optimization problem:

$$\begin{aligned} \min \ \Phi(\alpha) &= -\sum_{j=1}^{l} ln\left(k_j^T \alpha\right) \\ \text{s.t. } h(\alpha) &= \frac{1}{2}\alpha^T \alpha - 1 = 0. \end{aligned} \tag{4.19}$$

Figure 4.1 illustrates this situation graphically. We need to find the analytic center

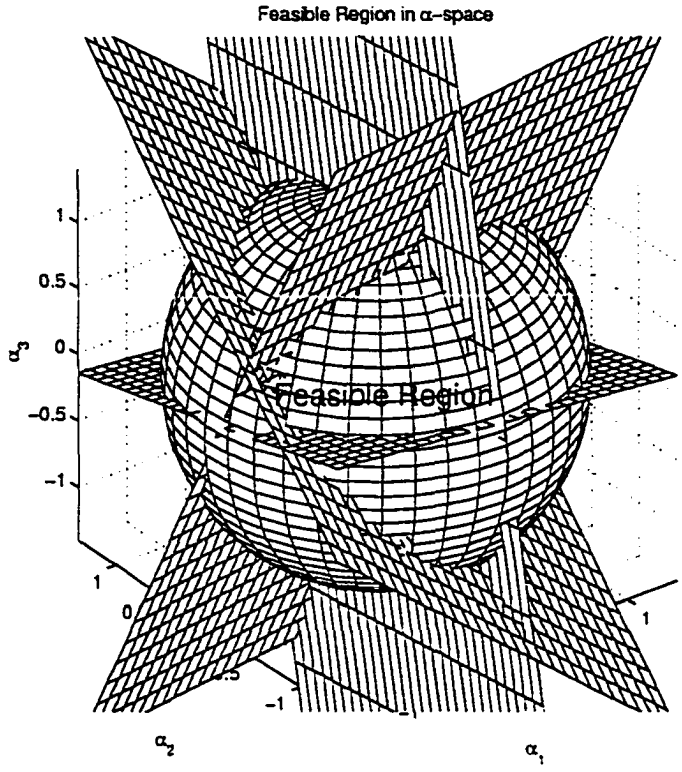


Figure 4.1: Feasible Region in α-space

of the feasible region, which is described by the intersection of several hyperplanes with the surface of the sphere. In the next section it will be shown, how the optimization problem can be solved using a projected Newton descent method.

4.3 A projected Newton descent method

In the previous section our discussion resulted in a constrained optimization problem, which must be solved by changing the set of α's. We started with the concept of a cone, which described all feasible w's in weight space that would qualify to separate the two classes. It was then shown that a similar approach could be used to solve nonlinear

instances resulting in roughly the same optimization problem. The two approaches differ in the space, in which the variables "live", as well as in the parameters. For the linear case information can be stored in $y_j \tilde{x}_j$, while for the nonlinear case the k_j's need to be computed. The vector k_j depends on the kernel function as well as on the input data (x_j, y_j). In the following we have to address the optimization problem in (4.19) and implement an algorithm that allows solving this type of problem.

Let us start the analysis by introducing the Lagrangian of (4.19). With u being the Lagrangian multiplier we have:

$$L(\alpha) = \Phi(\alpha) + u \cdot h(\alpha). \tag{4.20}$$

The Karush-Kuhn-Tucker (KKT) optimality conditions [1] require that at optimality the gradient of the Lagrangian vanish (feasibility of the corresponding dual optimization problem). Moreover, primal feasibility is required as well, hence, the constraint in (4.19) represents an additional optimality condition. The following equations account for dual and primal feasibility and represent the optimality conditions:

$$\nabla_\alpha L(\alpha) = \nabla_\alpha \Phi(\alpha) + u \cdot \nabla_\alpha h(\alpha) \tag{4.21}$$

$$h(\alpha) = \frac{1}{2}\alpha^T \alpha - 1 = 0. \tag{4.22}$$

Next, define a new variable $Z(\alpha, u)$ and let us rewrite the optimality conditions in matrix form. Thus,

$$Z(\alpha, u) = \begin{bmatrix} \nabla_\alpha L(\alpha) \\ h(\alpha) \end{bmatrix} = \begin{bmatrix} \nabla_\alpha \Phi(\alpha) + u \cdot \nabla_\alpha h(\alpha) \\ \frac{1}{2}\alpha^T \alpha - 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.23}$$

In a more compact form the optimality conditions require $Z(\alpha, u) = 0$. Employing the Newton-Raphson method to calculate α^* and u^*, the first-order approximation linear system

$$Z(\alpha_k, u_k) + \nabla_\alpha Z(\alpha_k, u_k) \begin{bmatrix} \alpha - \alpha_k \\ u - u_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.24}$$

must be solved, in order to calculate the next iterate $(\alpha, u) = (\alpha_{k+1}, u_{k+1})$. Equation (4.24) allows to iteratively approach the solution (α^*, u^*), with which we can compute the classifier. Here, ∇Z denotes the Jacobian of Z and (α_k, u_k) represents the current iterate. We find for the Jacobian of Z

$$\nabla Z(\alpha_k, u_k) = \begin{bmatrix} \nabla^2 L(\alpha_k, u_k) & \nabla h(\alpha_k, u_k) \\ \nabla h(\alpha_k, u_k)^T & 0 \end{bmatrix}. \tag{4.25}$$

Next, we need to compute the elements of ∇Z and Z in order to move to the next iterate (α_{k+1}, u_{k+1}). Let us first examine the derivatives of the logarithmic barrier function and introduce a matrix $S \in \Re^{l \times l}$ that carries in its diagonal the slack values $s_j = k_j^T \alpha_k$, where α_k represents the current iterate. Moreover, let $e = (1, 1, ..., 1)$ denote a vector of ones in an appropriate dimension, here $e \in \Re^l$. Then the gradient of $\Phi(\alpha)$ becomes $\nabla\Phi(\alpha) = -KS^{-1}e$. An alternative way of expressing the gradient of the barrier would be $\nabla\Phi(\alpha) = -\frac{k_1}{s_1} - \frac{k_2}{s_2} - ... - \frac{k_l}{s_l}$. The Hessian of the barrier can be computed similarly, resulting in $\nabla^2\Phi(\alpha) = KS^{-2}K^T$ or alternatively, $\nabla^2\Phi(\alpha) =$

$\frac{k_1 k_1^T}{s_1^2} + \frac{k_2 k_2^T}{s_2^2} + ... + \frac{k_l k_l^T}{s_l^2}$.

Considering next the gradient and the Hessian of h, we find for the gradient $\nabla h(\alpha) = \alpha$ and for the Hessian $\nabla^2 h(\alpha) = I$, with $I \in \Re^{(l+1)\times(l+1)}$ representing the unit matrix. Now it is easy to extend the discussion to the gradient and to the Hessian of the Lagrangian. In fact,

$$\nabla^2 L(\alpha, u) = \nabla^2 \Phi(\alpha) + u \cdot \nabla^2 h(\alpha) = K S^{-2} K^T + u \cdot I \tag{4.26}$$

$$\nabla L(\alpha, u) = \nabla \Phi(\alpha) + u \cdot \nabla h(\alpha) = -K S^{-1} e + u \cdot \alpha \tag{4.27}$$

Putting it all together we find for Z:

$$Z(\alpha_k, u_k) = \begin{bmatrix} -K S^{-1} e + u_k \cdot \alpha_k \\ \frac{1}{2}\alpha_k^T \alpha_k - 1 \end{bmatrix} \tag{4.28}$$

and for the Jacobian of Z:

$$\nabla Z(\alpha_k, u_k) = \begin{bmatrix} K S^{-2} K^T + u_k \cdot I & \alpha_k \\ \alpha_k^T & 0 \end{bmatrix}. \tag{4.29}$$

We have now presented a modified Newton procedure, which allows finding the minimum value of the barrier function subject to the constraint that all α's be elements of a hypersphere in $\Re^{l+1}$. Note the dimensions of $Z \in \Re^{l+2}$ and $\nabla Z \in \Re^{(l+2)\times(l+2)}$, since the $(l+1)$-th element corresponds to the offset b, while the $(l+2)$-th element refers to the Lagrangian multiplier u.

4.4 Implementation Issues

Until now, we have proposed a new method of selecting the optimal weight w^* for learning linear classifiers. Hereafter, the idea of finding a value close to the central trajectory of a cone was extended to the nonlinear case. We have studied ways to obtain the optimal value (α^* or w^*) from an open cone using concepts based on the analytic center. However, several remarks should be made about the current version of the algorithm. First, it is necessary to identify an interior point of the feasible region. Keep in mind that the barrier function is logarithmic; hence, it is not defined for α's yielding negative slacks. It might not be trivial to find an initial feasible solution, for which the Newton algorithm will work, in particular in the nonlinear case. Instead, it might be wiser to solve the problem in a sequential way. What do we mean by that? So far, the algorithm is confronted with all available data right from the beginning. We might call this the batch approach. If we initiate on the other hand the Newton procedure pattern by pattern, we will be able to circumvent the problem of finding an initial feasible solution. Therefore, we will next turn our attention to the issue of implementation of this new approach in machine learning.

4.4.1 How to process a "good" pattern

We propose an online approach in computing the analytic center in α-space by evaluating an approximation of the center for some current set of constraints. Once a pattern is classified correctly, which means that the current iterate is inside the current cone, we let a new pattern (a new hyperplane) enter the system of linear inequalities. Hence, for

each new pattern feasibility of the current solution is verified and adjusted. In the next few paragraphs we will outline the mechanics behind this way of solving the problem.

We initiate the algorithm with arbitrary starting values, α_0 and u_0. Next, the normal vector for the hyperplane in α-space is calculated:

$$k_1^T = (y_1 k_{11}, y_1 k_{21}, ..., y_1 k_{l1}, y_1) = (y_1 k(x_1, x_1), y_1 k(x_2, x_1), ..., y_1 k(x_l, x_1), y_1). \tag{4.30}$$

Upon evaluating $k_1^T \alpha_0$ two alternative outcomes are possible. If the value for $k_1^T \alpha_0$ is negative, which by definition also means that the slack $s_1 = k_1^T \alpha_0$ is negative, we have to correct our current solution α_0. Let us postpone for the moment the discussion of this case by assuming that α_0 is feasible, hence the slack is strictly positive. Remember that even for slack values equal to zero, the Hessian might be ill-conditioned as the logarithmic barrier approaches $+\infty$ in those cases. The gradient and the Hessian of the barrier function can be found respectively from

$$\nabla\Phi(\alpha_0) = -\frac{k_1}{s_1} \tag{4.31}$$

$$\nabla^2\Phi(\alpha_0) = \frac{k_1 k_1^T}{s_1^2}. \tag{4.32}$$

Next, evaluate $h(\alpha_0) = \frac{1}{2}\alpha_0^T \alpha_0 - 1$ as well as the gradient $\nabla h(\alpha_0) = \alpha_0$. With this in mind we find for Z:

$$Z(\alpha_0, u_0) = \begin{bmatrix} -\frac{k_1}{s_1} + u_0 \cdot \alpha_0 \\ \frac{1}{2}\alpha_0^T \alpha_0 - 1 \end{bmatrix}. \tag{4.33}$$

Since the Hessian of $h(\alpha)$ equals the unit matrix with dimension $(l+1) \times (l+1)$, the Jacobian of Z is easily computed using

$$\nabla Z(\alpha_0, u_0) = \begin{bmatrix} \frac{k_1 k_1^T}{s_1^2} + u_0 \cdot I & \alpha_0 \\ \alpha_0^T & 0 \end{bmatrix}. \tag{4.34}$$

Based on the calculations of Z and ∇Z, the next iterate can be computed by solving the linear Newton approximation in equation (4.24). However, we would like to point out that this subproblem will not be solved to optimality, as we are not interested in the solution after having processed only one pattern.

Consequently, after having successfully computed what amounts to α_1 and u_1 the normal vector corresponding to the second pattern is evaluated:

$$k_2^T = (y_2 k_{12}, y_2 k_{22}, ..., y_2 k_{l2}, y_2) = (y_2 k(x_1, x_2), y_2 k(x_2, x_2), ..., y_2 k(x_l, x_2), y_2). \tag{4.35}$$

Again, one needs to verify, whether the slack $s_2 = k_2^T \alpha_1$ is positive and if so, the new values for Z and ∇Z are computed:

$$Z(\alpha_1, u_1) = \begin{bmatrix} -\frac{k_1}{s_1} - \frac{k_2}{s_2} + u_1 \cdot \alpha_1 \\ \frac{1}{2}\alpha_1^T \alpha_1 - 1 \end{bmatrix} \tag{4.36}$$

$$\nabla Z(\alpha_1, u_1) = \begin{bmatrix} \frac{k_1 k_1^T}{s_1^2} + \frac{k_2 k_2^T}{s_2^2} + u_1 \cdot I & \alpha_1 \\ \alpha_1^T & 0 \end{bmatrix}. \tag{4.37}$$

Hereafter, we conduct an additional Newton search and update the current solution (α_2, u_2) using equation (4.24) before repeating this procedure for the remaining patterns.

Starting from an initial solution at $\alpha_0 = (0.5, 0.5)$ Figure 4.2 illustrates, how the current solution is adjusted, when we process successively the patterns k_1, k_2, and k_3. The updated solution is given by α_3.

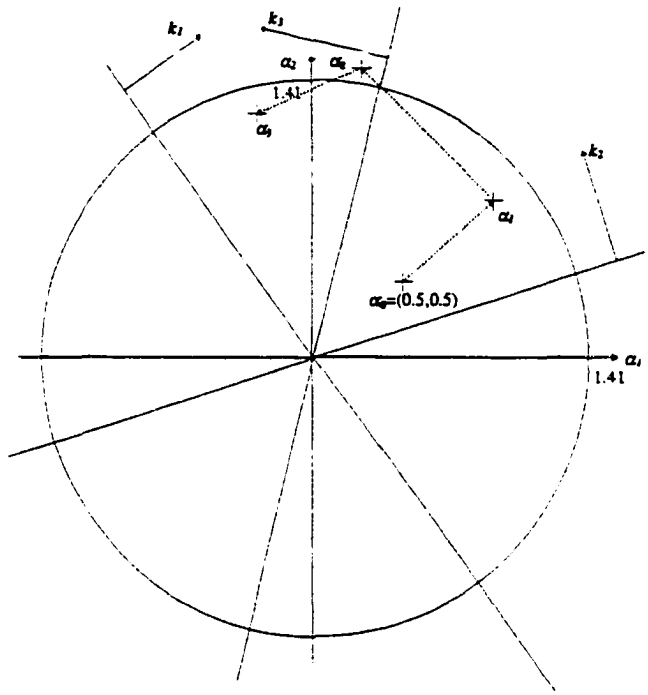


Figure 4.2: The previous solution α_{j-1} is always feasible with respect to the pattern k_j

Next, we will explain, how to adapt the algorithm for the case where the cone is significantly reduced by a new pattern such that the previous solution lies outside the new cone.

4.4.2 How to correct a "bad" pattern

If patterns are not correctly classified, the current iterate represents an infeasible starting point for the Newton optimization process, which excludes using this type of optimiza-

tion technique. To be more precise, suppose that we are at iteration j and that the j-th pattern reduces the feasible region so significantly that for the $(j-1)$-th solution (α_{j-1}, u_{j-1}), α_{j-1} lies outside of the cone. Looking at the set of slacks, this implies that the j-th slack s_j will be negative. However, all remaining slacks $s_1, ..., s_{j-1}$ are still positive. Hence, we either need to move the current iterate into the new cone or alternatively relax the shape of the feasible region, such that the new feasible set still comprises the α_{j-1} of the previous iterate (α_{j-1}, u_{j-1}). Since we have only very limited information about the current position of (α_{j-1}, u_{j-1}) we decided to pursue the latter approach of relaxing the j-th constraint of the feasible region.

Since there is only one constraint (constraint j, the last one that entered the system of patterns), which is violated by the current iterate, we only need to focus on the j-th slack value. Keep in mind that s_j is a measure of distance between α_{j-1} and the hyperplane $k_j^T \alpha = 0$. Modification of the feasible region will be realized as follows. Since s_j is the sole parameter defining the position of the hyperplane, we can shift the hyperplane parallel to itself by changing the value for s_j. For instance, the hyperplane $k_j^T \alpha = s_j$ passes exactly through the current iterate α_{j-1}. Obviously, the parallel shift is not yet sufficient, as the barrier function approaches $+\infty$ for this situation. For this reason, we need to shift the last hyperplane a little farther, say, about ten percent of the slack value to guarantee that the current iterate α_{j-1} lies inside this "extended" cone. Consequently, the j-th shifted hyperplane is defined by the set of all α's for which $k_j^T \alpha = 1.1 s_j$. This hyperplane will change the entry for the j-th slack to $-0.1 s_j$. Remember that s_j is negative, therefore $-0.1 s_j$ is positive. We have now adjusted the feasible region such that α_{j-1} is an element of the feasible perturbed set.

Next, we need to invoke Newton's algorithm to virtually "push" α_{j-1} to the analytic center of the feasible region. This will be realized by conducting five consecutive Newton steps. Let $i = 1, ..., 5$ count the Newton steps on the relaxed problem. At each step, we will examine, whether the updated solution $(\alpha_{j-1,i}, u_{j-1,i})$ still violates the original constraint $k_j^T \alpha_{j-1,i} \geq 0$. If so, the remaining Newton steps are executed. Of course, one must be very careful about updating the slack s_j during this process.

Store the overall shift of the j-th hyperplane in the variable $total = -1.1 s_j$. That way, once Newton's algorithm offers a new $\alpha_{j-1,i}$, the j-th slack value is calculated with respect to the original (unrelaxed) problem, $s = K^T \cdot \alpha$. However, the Newton procedure requires a positive slack value $s_{j,i}$, which means that the j-th slack must be separately updated with respect to the relaxed problem. Therefore, s_j is replaced by an intermediate slack valid only for the i-th Newton iteration on the relaxed problem $s_{j,i} = total + s_{j,i-1}$ in order to yield the slacks for the i-th Newton iteration on the relaxed problem (define $s_{j,0} = s_j$). This approach essentially measures the distance between $\alpha_{j-1,i}$ from the relaxed hyperplane $k_j^T \alpha = -total$.

Nonetheless, it is possible that even after five Newton steps the analytic center of the relaxed feasible region does not yet satisfy the j-th constraint. Keeping in mind that $total = -1.1 s_j$ defined the position of the relaxed j-th hyperplane we can calculate a new relaxed problem with a relaxed hyperplane that will lie between the first relaxed hyperplane and the original unrelaxed hyperplane (all parallel, we did not modify k_j). Obviously, our situation will have improved after five Newton iterations, however, it is possible that the analytic center of the relaxed problem still violates the original problem. The new offset for the second relaxed problem can be computed as

$total = -1.1 s_{j-1,5}$ yielding the hyperplane $k_j^T \alpha = -total$. In other words the first relaxed hyperplane is moved to a new position reducing the relaxed cone. Hereafter, we invoke once more Newton's descent algorithm. Either one of the intermediate iterates $(\alpha_{j-1,i}, u_{j-1,i})$ finally lies within the original cone, or we have to construct a third intermediate hyperplane after five Newton steps. Recapitulating we have to deal with two different counters. Specifically, j indicates the pattern and i indicates the Newton step executed. To increase readability we have refrained from introducing a third index denoting the number of relaxed hyperplanes needed until we have moved the current iterate $(\alpha_{j-1,i}, u_{j-1,i})$ inside the original cone.

Note that for each newly computed solution $(\alpha_{j-1,i}, u_{j-1,i})$ ones needs to verify *all* slacks s_j, since it is possible that a pure Newton step leaves the feasible region. In our case the new α could violate a constraint other than the one (s_j), which has just been relaxed. This case arises particularly, if α_{j-1} of the current solution $(\alpha_{j-1,i}, u_{j-1,i})$ is located already rather close to one or more constraints [33]. We decided to halve the step size of a pure Newton step and verify whether feasibility is given. If not, the step size is cut in half again and constraint violation is checked again.

Figure 4.3 displays the procedure of adjusting infeasible α_j's. As can be seen α_0 is feasible for the first pattern k_1 moving the current solution to α_1. Hereafter, the second training pattern k_2 reduces the feasible region significantly so that α_1 is no longer feasible. Thus the second constraint is relaxed by $total' = -1.1 s_2' = -1.1 s_{2,0}'$. Conducting five successive Newton steps results in the new iterate $\alpha_{1,5}' = \alpha_{1,0}''$. Obviously, the current solution is not yet feasible with respect to the original problem. Therefore, a second relaxation (indicated by a second prime) has to be conducted. Hence, $s_{2,5}' = s_{2,0}''$ and

the Newton method is invoked again. Feasibility is achieved at $i = 2$ and the training pattern can now be considered as a "good pattern". Therefore, one more Newton step is conducted with respect to the original feasible region to classify k_2.

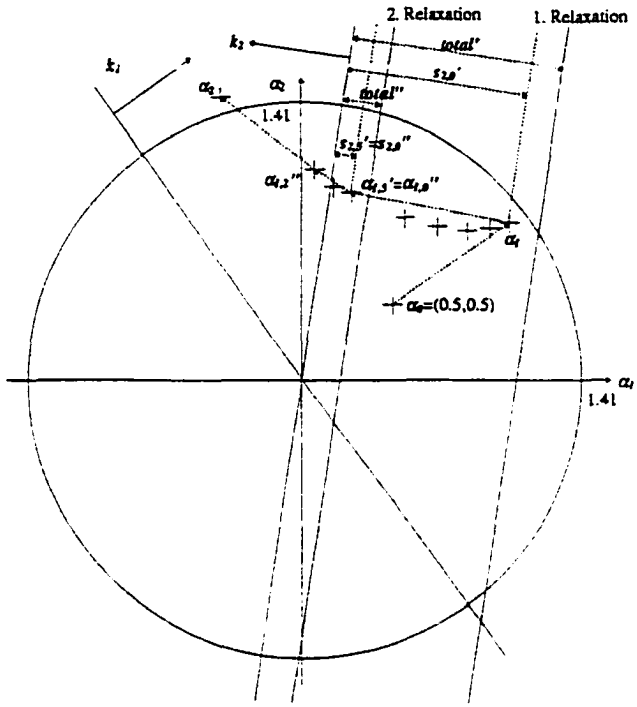


Figure 4.3: The current solution α_1 is infeasible for the second pattern k_2

We will now recapitulate our discussion until this point. The following generic pseudo code provides an overview of all steps necessary for the implementation of an online analytic center machine.

Step 1: Initialize

L - Number of patterns

j - Current pattern, $j = 1, ..., L$

i - Index to identify all patterns from $1, ..., j$

α_0 - Initial starting point, vector of ones

u - Initial Lagrangian multiplier for projection on sphere

Step 2a: Center and Add

Set $j = 1$

Compute the newest k_j according to $k_j^T = (y_j k_{1j}, y_j k_{2j}, ..., y_j k_{lj}, y_j)$,

the slacks $s_i = k_i^T \alpha, \ \forall i = 1, ..., j$, and let $S = diag\,(s_i)\,, \forall i = 1, ..., j$

Let $K = (k_1, ..., k_j)$

if $s_j \leq 0$

go to step 2b: Correct

else

go to step 2c: Update

Step 2b: Correct

Set $total = -1.1 \cdot s_j$ and $s_{relax} = -0.1 \cdot s_j$ (Problem relaxation)

Let $S_{relax} = diag\,(s_1, s_2, ..., s_{j-1}, s_{relax})$

while $(s_j = k_j \alpha \leq 0)$

for $newtoncounter = 1$ to 5

compute Z and ∇Z using equations (4.28) and (4.29)

calculate new (α, u) using Newton's approximation (4.24)

if $(s = K^T \cdot \alpha \geq 0)$

go to step 2c: Update

else

update the j-th slack, $s_j = total + s_j$ (s_j is still negative!)

check feasibility of all other slacks $(s_1, ..., s_{j-1})$

while $min\,(s_1, ..., s_{j-1}) \leq 0$

halve the pure Newton step

compute $s = K^T \cdot \alpha$ and replace s_j, $s_j = total + s_j$

endwhile

endfor

Set $total = -1.1 \cdot s_j$ (Construct new relaxation)

Replace s_j, $s_j = -0.1 \cdot s_j$

endwhile

Step 2c: Update

compute Z and ∇Z using equations (4.28) and (4.29)

calculate new (α, u) using the newton approximation (4.24)

$j = j + 1$, return to step 1

Chapter 5

An Analytic Center Machine for Regression Analysis

5.1 Statement of the problem

In the previous chapter the concept of the analytic center machine was presented for classification problems. We now extend this idea to regression analysis. Again, we will first address the geometrical aspects and branch out from there to the optimization model that extracts a suitable solution.

5.1.1 Geometry of Linear Learning Machines for Regression

Considering the problem of linear regression the variable definition must be changed slightly. Let $x_j \in \Re^d$ and $y_j \in \Re$ describe a data set consisting of l samples (x_j, y_j). We would like to approximate this data set by a linear discriminant function $f(x) = w^T x + b$, where $w \in \Re^d$ and $b \in \Re$. Next, a learning machine based on the concept of the analytic

center is developed. In section 5.2 we will demonstrate that the following approach can easily be extended to the case of nonlinear regression. Spelling out the feasibility constraints we find

$$\begin{aligned} y_j - f(x_j) &\leq \epsilon \qquad \forall j = 1, ..., l \\ f(x_j) - y_j &\leq \epsilon \qquad \forall j = 1, ..., l. \end{aligned} \tag{5.1}$$

Substituting for the discriminant function the following set of constraints is found:

$$\begin{aligned} y_j - w^T x_j - b &\leq \epsilon \qquad \forall j = 1, ..., l \\ w^T x_j + b - y_j &\leq \epsilon \qquad \forall j = 1, ..., l. \end{aligned} \tag{5.2}$$

Any feasible solution (w, b) of (5.2) will provide a possible solution for the regression problem. In the standard SVM approach the length $\|w\|^2$ is minimized. In this chapter, however, a solution will be computed that corresponds to the analytic center in the space of hypotheses. This means that w and b will be considered as variables while the data set of the problem (x_j, y_j) is fixed. Before we will discuss implications on the geometry of the feasible region (version space), let us simplify the equations. Defining $\tilde{w} = (w, 1) \in \Re^{d+1}$ and $\tilde{x}_j = (x_j, b) \in \Re^{d+1}$ reduces the set of constraints to

$$\begin{aligned} y_j - \tilde{w}^T \tilde{x}_j &\leq \epsilon \qquad \forall j = 1, ..., l \\ \tilde{w}^T \tilde{x}_j - y_j &\leq \epsilon \qquad \forall j = 1, ..., l. \end{aligned} \tag{5.3}$$

Notice that the dimension of the input space has been extended by one, where the variable b essentially becomes $\tilde{w}_{d+1}$. Rewriting the set of equations yields:

$$\begin{aligned} \tilde{w}^T \tilde{x}_j &\geq y_j - \epsilon \qquad \forall j = 1, ..., l \\ \tilde{w}^T \tilde{x}_j &\leq y_j + \epsilon \qquad \forall j = 1, ..., l. \end{aligned} \tag{5.4}$$

Consider the jth pair of inequalities. The first inequality describes a halfspace with a normal vector $\tilde{x}_j$ in $\Re^{d+1}$ and an offset $y_j - \epsilon$. The second inequality corresponds to a halfspace whose normal vector is parallel to the one of the first halfspace, but whose offset is $y_j + \epsilon$. Here however, the direction of $\tilde{x}_j$ is reversed, thus, these two halfspaces combine into a slab of infinite width and length.

Hence, for each j we find a slab with normal vector $\tilde{x}_j$ and two offsets of $y_j - \epsilon$ and $y_j + \epsilon$. The version space $V(T_R)$ (or feasible region) becomes therefore the intersection of l slabs of infinite length in $\Re^{d+1}$. Any vector $\tilde{w}$ that satisfies all constraints lies inside all l slabs and solves the regression problem. Of course the solution provided by most of these $\tilde{w}$'s will be rather disappointing. For support vector machine regression the objective function serves as a tool to extract a particular $\tilde{w}$. For the analytic center machine, the one $\tilde{w}$ is selected that minimizes the sum over all logarithms of the slacks of the constraints. Intuitively, this solution attempts to extract a point inside the feasible region that is as far away from the constraints as possible. Usually, solutions close to the boundary of the version space have a larger generalization error [5]. The following paragraph will illustrate this approach mathematically.

5.1.2 Analytical Center of Version Space for Regression

Similar to the pattern classification instance, we will approach this problem by first computing the slacks that correspond to the constraints in (5.3). For the set of constraints, which currently describe the feasible region, a set of l lower slacks and l upper slacks can be defined:

$$\begin{aligned} \tilde{s}_j^l &= \tilde{w}^T \tilde{x}_j - y_j + \epsilon \geq 0 \quad \forall j = 1, ..., l \\ \tilde{s}_j^u &= -\tilde{w}^T \tilde{x}_j + y_j + \epsilon \geq 0 \quad \forall j = 1, ..., l. \end{aligned} \tag{5.5}$$

Obviously, each slab accounts for one upper and one lower slack value. Furthermore, using once more the concept of the barrier function as outlined earlier in (4.5), we compute the following expression for the logarithmic barrier function Φ:

$$\Phi\left(\tilde{s}_j^l, \tilde{s}_j^u\right) = -\sum_{j=1}^{l} ln\left(\tilde{s}_j^l\right) - \sum_{j=1}^{l} ln\left(\tilde{s}_j^u\right). \tag{5.6}$$

As for the pattern classification scenario the minimum of the logarithmic barrier function $\Phi\left(\tilde{s}_j^l, \tilde{s}_j^u\right)$ constitutes an excellent means of computing a point that will be far away from all constraints. At this point we will refrain from a further analysis of the linear regression instance but rather continue the discussion from here, once the nonlinear and therefore the more general approach has been laid out. Section 5.2 will describe, how the problem can be transferred to the feature space using the concept of kernelization.

5.2 Kernelization

In the previous section 5.1 the regression problem for linear instances was introduced. The space of hypotheses was presented providing a geometric interpretation of the constraint set in the space of $\tilde{w}$'s. Next, the problem will be reformulated to solve nonlinear regression problems. Expressing again the weight vector by a linear combination of data vectors, $w = \sum_{i=1}^{l} \alpha_i x_i$, the set of $2l$ equations in (5.2) defining l slabs in weight space can be reformulated as follows:

$$\begin{aligned} y_j - w^T x_j - b &= y_j - \left(\sum_{i=1}^{l} \alpha_i x_i\right)^T x_j - b = y_j - \sum_{i=1}^{l} \alpha_i x_i^T x_j - b \leq \epsilon \qquad (5.7) \\ \forall j &= 1, ..., l \\ w^T x_j + b - y_j &= \left(\sum_{i=1}^{l} \alpha_i x_i\right)^T x_j + b - y_j = \sum_{i=1}^{l} \alpha_i x_i^T x_j + b - y_j \leq \epsilon \\ \forall j &= 1, ..., l, \end{aligned}$$

where $x_j, w \in \Re^d$, $y_j, b \in \Re$, and $\epsilon \in \Re^+$. Recall that ϵ can be interpreted as the precision of the regression function. Nonnegative values for ϵ can be arbitrarily selected by the decision maker as long as the set of constraints remains feasible. Moreover, keep in mind that in many applications the input data reduces to a real scalar (time-series-analysis), thus we have $d = 1$ in these cases.

After this preprocessing step the regression problem can now be transferred to feature space by replacing the inner products by the kernel function $k(x_i, x_j)$. Therefore, the set of constraints representing the l patterns that are available for regression training can be described in the dual space of α's as follows:

$$y_j - \sum_{i=1}^{l} \alpha_i k(x_i, x_j) - b \leq \epsilon \quad \forall j = 1, ..., l \tag{5.8}$$
$$\sum_{i=1}^{l} \alpha_i k(x_i, x_j) + b - y_j \leq \epsilon \quad \forall j = 1, ..., l.$$

The set of linear inequalities defines a convex set in the space of α's. Any α that solves the system in (5.8) is an element of the version space in the space of α's. It will be shown shortly that similar to linear regression the version space for the nonlinear variable α corresponds geometrically to an intersection of l slabs. Once an acceptable feasible solution α has been found, the regression function can be computed as

$$f(x) = \sum_{i=1}^{l} \alpha_i k(x_i, x) + b. \tag{5.9}$$

We are now ready to spell out our learning problem in a more compact form in α-space. Define the matrix $K \in \Re^{(l+1)\times l}$ as:

$$K = \begin{bmatrix} k_{11} & k_{12} & ... & k_{1l} \\ k_{21} & k_{22} & ... & k_{2l} \\ ... & ... & & ... \\ k_{l1} & k_{l2} & ... & k_{ll} \\ 1 & 1 & ... & 1 \end{bmatrix}. \tag{5.10}$$

Again, we follow the convention that the k_{ij} represent the kernel evaluated at the points x_i and x_j. Specifically, $k_{ij} = k(x_i, x_j)$. Note that K differs slightly from the kernel matrix derived for pattern classification as the variable y_j is omitted. By defining the vectors

$$k_j = \begin{bmatrix} k_{1j} \\ k_{2j} \\ \dots \\ k_{lj} \\ 1 \end{bmatrix}, \forall j = 1, ..., l, \tag{5.11}$$

where $k_j \in \Re^{l+1}$, K can be expressed as $K = (k_1, k_2, ..., k_l)$. Furthermore, if we let the vector $\alpha \in \Re^{l+1}$ to be

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_l \\ b \end{bmatrix}, \tag{5.12}$$

we can write the set of inequalities for all l patterns as

$$\begin{aligned} y_j - k_j^T \alpha &\leq \epsilon \quad \forall j = 1, ..., l \\ k_j^T \alpha - y_j &\leq \epsilon \quad \forall j = 1, ..., l. \end{aligned} \tag{5.13}$$

Note that the offset b is included as the $(l+1)$-element and the corresponding adjustments were made for K and k_j. Verifying the current formulation for a linear kernel $k(x_i, x_j) = x_i^T x_j$ one computes for the halfspace corresponding to the lower slack the same expression as in (5.2):

$$
\begin{aligned}
y_j - k_j^T \alpha &= y_j - \alpha_1 k(x_1, x_j) - \alpha_2 k(x_2, x_j) - ... - \alpha_l k(x_l, x_j) - b \qquad (5.14) \\
&= y_j - \alpha_1 x_1^T x_j - \alpha_2 x_2^T x_j - ... - \alpha_l x_l^T x_j - b = \\
&= y_j - \sum_{i=1}^{l} \alpha_i x_i^T x_j - b = y_j - w^T x_j - b \leq \epsilon.
\end{aligned}
$$

What type of geometry is described by the set of inequalities in (5.13)? Rewriting the constraints yields:

$$
\begin{aligned}
k_j^T \alpha &\geq y_j - \epsilon \quad \forall j = 1, ..., l \qquad (5.15) \\
k_j^T \alpha &\leq y_j + \epsilon \quad \forall j = 1, ..., l.
\end{aligned}
$$

Hence, for a given j there are two constraints each defining two hyperplanes parallel to each other with normal vector $k_j \in \Re^{l+1}$. It can be seen that all feasible α's must lie in between the two hyperplanes. In other words, the two hyperplanes represent a lower and an upper bound for a slab in $\Re^{l+1}$, between which all feasible solutions for the j-th point of the regression data are contained. Note that here the data "lives" in a space whose dimension is $(l+1)$, while in the linear model the slabs were constructed in $\Re^{d+1}$. Next, a lower slack s_j^l and an upper slack s_j^u can be defined for each data sample:

$$
\begin{aligned}
s_j^l &= k_j^T \alpha - y_j + \epsilon \geq 0 \quad \forall j = 1, ..., l \qquad (5.16) \\
s_j^u &= -k_j^T \alpha + y_j + \epsilon \geq 0 \quad \forall j = 1, ..., l.
\end{aligned}
$$

For example, if a vector α is considered, which lies exactly in the geometrical center of

the j-th slab, then the lower slack and the upper slack should both be equal, specifically $s_j^l = s_j^u$.

Before the general case of l slabs is discussed, let us first elaborate on the situation, where $l = 2$. This means for the regression training problem that a regression function for two points must be derived. Then the feasible region in the space of α's is an intersection of two slabs. Keeping in mind that the dimensionality in the space of α's exceeds the number of training points by one, the two slabs will "live" in this case in a three-dimensional space, $\Re^3$. The intersection of two slabs in $\Re^3$ results in a feasible region which is unbounded and therefore not compact. This important finding indicates that the logarithmic barrier function will attain its minimum at infinitely many points, if no additional constraints are imposed.

Similar conclusions hold true for the case of l points. Since $\alpha \in \Re^{l+1}$, the intersection of l slabs will not yield a bounded feasible region. Suppose for a moment that additional slabs were available to compactify the feasible region (version space). Then, minimization of the logarithmic barrier $\Phi(\alpha)$ would result in a unique global minimum, α^*, in other words the solution would be represented by a single point. Since the collection of constraints comprises only l slabs, the feasible region of the current model is unbounded and the barrier function attains its minimum on points lying on a line, if the l vectors k_j are linearly independent. If some of the k_j are linearly dependent, the set of α^*'s yielding the same minimum barrier value are found on an affine space of the dimension of the nullspace of K, $N = null(K)$.

The question is, which one of the infinitely many solutions for $\Phi(\alpha^*)$ we should choose. Since the only point on a line or a hyperplane with a distinct characteristic is

the point closest to the origin, it seems most appropriate to select this *minimum norm solution.*

Next, it is necessary to formulate the additional constraints that allow us to isolate this point. Let q_j indicate the vectors, which are perpendicular to all l vectors k_j. In other words, the vectors q_j form a basis for the nullspace of K and let v be the dimension of this nullspace $v = dim\,(N)$. We know that there exists at least one q_j. However, depending on how many vectors k_j are linearly dependent up to l vectors could form the nullspace leaving a one dimensional rangespace of K, $range\,(K) \in \Re$. Finally, introduce the matrix Q, which carries all q_j, $Q = (q_1, ..., q_v) \in \Re^{(l+1)\times v}$.

Since the q_j are perpendicular to all k_j, they provide an excellent tool to "close" the feasible region, as they can be used to define pairs of halfspaces (slabs) that will compactify the set of constraints. Thus, two parallel hyperplanes with normal vector q_j will be sufficient to exclude all but one solution from the set of vectors minimizing the logarithmic barrier function along a particular direction q_j. Intuitively, we provide two "lids" to close the feasible region with respect to the direction q_j. Moreover, if we use the same ϵ as in expression (5.16) and if we set $y_j = 0$, we will extract the solution along the direction q_j, which is closest to the origin. Repeating this procedure for all q_j, we extract an α that minimizes the logarithmic barrier function and that is closest to the origin (minimum norm solution). We thus have to deal with the following set of constraints describing v additional slabs:

$$q_j^T \alpha \;\geq\; 0 - \epsilon \quad \forall j = 1, ..., v \tag{5.17}$$

$$q_j^T \alpha \leq 0 + \epsilon \quad \forall j = 1, ..., v.$$

These inequalities translate into the following set of slack values:

$$\begin{aligned} s_{l+j}^l &= q_j^T \alpha - 0 + \epsilon \geq 0 \quad \forall j = 1, ..., v \\ s_{l+j}^u &= -q_j^T \alpha + 0 + \epsilon \geq 0 \quad \forall j = 1, ..., v. \end{aligned} \tag{5.18}$$

Consequently, we have to solve the following unconstrained convex optimization problem:

$$\begin{aligned} \min \ \Phi(\alpha) = &- \sum_{j=1}^{l} ln\left(k_j^T \alpha - y_j + \epsilon\right) - \sum_{j=1}^{l} ln\left(-k_j^T \alpha + y_j + \epsilon\right) \\ &- \sum_{j=1}^{v} ln\left(q_j^T \alpha + \epsilon\right) - \sum_{j=1}^{v} ln\left(-q_j^T \alpha + \epsilon\right) \end{aligned} \tag{5.19}$$

The first sum in the objective function refers to the lower slacks s_j^l, the second sum refers to the upper slacks s_j^u, while v denotes the dimension of the nullspace of K. The third and fourth sum indicate the additional set of v slabs necessary to compactify the feasible region.

The following section 5.3 will outline, how an initial feasible solution can be found by solving a linear programming problem. Thereafter, a conjugate gradient method will be presented that solves the optimization problem in (5.19).

5.3 Finding an initial feasible solution (Phase I)

In the previous section 5.2 an optimization problem was developed that will identify the analytic center solution for the regression analysis problem. In order to find the

minimum value of the logarithmic barrier function Φ, availability of an initial feasible solution would significantly facilitate the algorithm. For the optimization problem at hand an initial feasible solution can be obtained as follows. Introduce a new function $h : \Re^{l+1} \rightarrow \Re$, where h is defined on the set of constraints described in (5.13). It can be seen that the epigraph of h defines an unbounded, yet convex set. Consequently, for two different values for ϵ with the restriction $\epsilon_2 \geq \epsilon_1$ the following observation can be made. Suppose, the set L_1 comprises all feasible α_1 for ϵ_1, thus $L_1 = \{\alpha_1 \in \Re^{l+1} : \quad y_j - k_j^T \alpha_1 \leq \epsilon_1, k_j^T \alpha_1 - y_j \leq \epsilon_1, \quad \forall j = 1, ..., l\}$. Similarly, $L_2 = \{\alpha_2 \in \Re^{l+1} : y_j - k_j^T \alpha_2 \leq \epsilon_2, k_j^T \alpha_2 - y_j \leq \epsilon_2, \quad \forall j = 1, ..., l\}$. Since $\epsilon_2 \geq \epsilon_1$ and since h is a convex function, we can conclude that $L_2 \supset L_1$.

With this in mind consider the following remark: if we can compute the smallest possible ϵ (ϵ_{min}) and simultaneously the corresponding solution vector, then this particular vector will be an initial feasible solution for all other ϵ, since we always have $\epsilon \geq \epsilon_{min}$. Thus, minimizing ϵ subject to the feasibility constraints in (5.13) results in a linear programming problem yielding an initial feasible solution and at the same time a lower bound for the value of ϵ. Finally, in order to extract the minimum norm solution, additional constraints have to be imposed, similar to the ones introduced in expression (5.17). Thus, the linear programming model becomes:

$$\begin{aligned} & \min \ \epsilon & (5.20) \\ \text{s.t. } & y_j - k_j^T \alpha \ \leq \ \epsilon \quad \forall j = 1, ..., l \\ & k_j^T \alpha - y_j \ \leq \ \epsilon \quad \forall j = 1, ..., l. \end{aligned}$$

$$-q_j^T \alpha \leq \epsilon \quad \forall j = 1, ..., v$$

$$q_j^T \alpha \leq \epsilon \quad \forall j = 1, ..., v.$$

Call the minimum value of the objective function of this linear program ϵ_{min} and the corresponding solution vector α_{init}. We will call this part of the algorithm phase I, since it directly influences phase II, the minimization of the barrier for a given ϵ selected by the decision maker. Phase I provides the decision maker with the information about the smallest value of ϵ that he can select without the set of constraints in (5.20) and (5.13) being infeasible. Note that $\epsilon_{min} = 0$ means that we have perfect curve fitting, $k_j^T \alpha_{init} = y_j$, $\forall j = 1, ..., l$.

5.4 A conjugate gradient method to minimize the logarithmic barrier (Phase II)

In section 5.3 it was demonstrated that a simple linear programming problem can be used to identify an initial feasible solution α_{init} for the optimization problem in (5.19). Note however that if the decision maker selects $\epsilon = \epsilon_{min}$ the logarithmic barrier function is not defined, in fact, the feasible region degenerates to a point. In that case α_{init} is the solution. This part of the regression training algorithm was referred to as phase I. In this section a conjugate gradient method will be presented that allows computation of the minimum value of the logarithmic barrier function $\Phi(\alpha)$. This will be phase II of the regression training algorithm, which identifies the optimal set of α's for a given $\epsilon \geq \epsilon_{min}$. Recall that the optimization problem that needs to be solved is

$$\begin{aligned}\min\ \Phi(\alpha) = &-\sum_{j=1}^{l} ln\left(k_j^T\alpha - y_j + \epsilon\right) - \sum_{j=1}^{l} ln\left(-k_j^T\alpha + y_j + \epsilon\right) \\ &-\sum_{j=1}^{v} ln\left(q_j^T\alpha + \epsilon\right) - \sum_{j=1}^{v} ln\left(-q_j^T\alpha + \epsilon\right) \\ = &-\sum_{j=1}^{l} ln\left(s_j^l\right) - \sum_{j=1}^{l} ln\left(s_j^u\right) - \sum_{j=l+1}^{l+v} ln\left(s_j^l\right) - \sum_{j=l+1}^{l+v} ln\left(s_j^u\right)\end{aligned} \tag{5.21}$$

keeping in mind that the third and fourth summation refer to the additional constraints necessary to guarantee a compact feasible region. Conjugate gradient methods appear rather appealing for this type of unconstrained optimization problem due to their limited storage requirements. Depending on the exact type of conjugate gradient method we require only three or four n-vectors to be stored in memory for the algorithm [1]. In the following the somewhat general procedure for conjugate gradient methods will be briefly laid out.

We wish to minimize the unconstrained function $\Phi : \Re^{l+1} \rightarrow \Re$ with α^j representing the argument to be minimized at iteration j. The next iterate is computed from

$$\alpha^{j+1} = \alpha^j + \lambda^j z^j, \tag{5.22}$$

where z^j constitutes the search direction and λ^j the optimal step length minimizing the barrier function Φ along z^j. The algorithm is initialized by defining the first search direction as the negative gradient of Φ at α^0:

$$z^0 = -\nabla\Phi\left(\alpha^0\right). \tag{5.23}$$

Hereafter, all new search directions are computed as convex combinations of the

previous search direction and the current gradient

$$z^{j+1} = -\nabla\Phi\left(\alpha^{j+1}\right) + t \cdot z^{j}, \tag{5.24}$$

with t representing the so-called deflection parameter. Various approaches exist to compute t [1, 2]. In this discussion we follow Fletcher and Reeves' approach by employing

$$t = \frac{\nabla\Phi\left(\alpha^{j+1}\right)^{T}\nabla\Phi\left(\alpha^{j+1}\right)}{\nabla\Phi\left(\alpha^{j}\right)^{T}\nabla\Phi\left(\alpha^{j}\right)}. \tag{5.25}$$

As can be seen one essential ingredient in conjugate gradient methods is the computation of $\nabla\Phi(\alpha)$. Consequently, we will next devote our attention to finding a compact way of computing the gradient of the logarithmic barrier function. Recall that overall there are $(l+v)$ slabs resulting in $(l+v)$ upper slacks and $(l+v)$ lower slacks, respectively. For simplicity, we will include all inequalities at once in our discussion, thus, we will introduce variables with the subscript "ext" indicating the extended vector or matrix, which includes also the additional constraints that extract the minimum norm solution. Recall that $Q = (q_1, ..., q_v) \in \Re^{(l+1)\times v}$ Thus, let

$$K_{ext} = [K, Q] \in \Re^{(l+1)\times(l+v)} \tag{5.26}$$

and

$$y_{ext} = (y_1, ..., y_l, 0, ..., 0) \in \Re^{l+v}, \tag{5.27}$$

which means that we add exactly v zeros to the vector containing the y_j. Next,

define a vector s^l carrying $(l+v)$ lower slacks as well as a vector s^u with $(l+v)$ upper slacks. Moreover, with $e^T = (1, 1, ..., 1) \in \Re^{l+v}$ the lower and the upper slacks for all $(l+v)$ slabs can be calculated in closed form as follows:

$$\begin{aligned} s^l &= K_{ext}^T \alpha - y_{ext} + \epsilon \cdot e \geq 0 \\ s^u &= -K_{ext}^T \alpha + y_{ext} + \epsilon \cdot e \geq 0. \end{aligned} \tag{5.28}$$

Let S_l and S_u be the corresponding diagonal matrices of the vectors s^l and s^u. Hence,

$$\begin{aligned} S_l &= diag\left(s^l\right) = diag\left(K_{ext}^T \alpha - y_{ext} + \epsilon \cdot e\right) \\ S_u &= diag\left(s^u\right) = diag\left(-K_{ext}^T \alpha + y_{ext} + \epsilon \cdot e\right). \end{aligned} \tag{5.29}$$

Using the above definitions the gradient of $\Phi(\alpha)$ becomes:

$$\nabla\Phi(\alpha) = K_{ext} S_u^{-1} e - K_{ext} S_l^{-1} e. \tag{5.30}$$

In open form $\nabla\Phi(\alpha)$ corresponds to

$$\nabla\Phi(\alpha) = \left(\frac{k_1}{s_1^u} - \frac{k_1}{s_1^l}\right) + ... + \left(\frac{k_l}{s_l^u} - \frac{k_l}{s_l^l}\right) + \left(\frac{q_1}{s_{l+1}^u} - \frac{q_1}{s_{l+1}^l}\right) + ... + \left(\frac{q_v}{s_{l+v}^u} - \frac{q_v}{s_{l+v}^l}\right), \tag{5.31}$$

where the variable α is embedded in the various slack values. Notice that for vectors that are located exactly in the center of a given slab the gradient components of the lower and the upper hyperplane cancel out resulting in a zero gradient for this particular slab.

Once the deflection parameter t and the new search direction z^{j+1} have been calculated, a line search must be performed minimizing $\Phi(\alpha)$ along the direction z^{j+1}.

Consider the first iteration and let $\alpha^1 = \alpha^0 + \lambda z^0$. Plugging this expression into equation (5.21) yields:

$$\Phi(\lambda) = -\sum_{j=1}^{l} ln\left(k_j^T\alpha^0 - y_j + \epsilon + \lambda k_j^T z^0\right) - \sum_{j=1}^{l} ln\left(-k_j^T\alpha^0 + y_j + \epsilon - \lambda k_j^T z^0\right) \quad (5.32)$$
$$- \sum_{j=1}^{u} ln\left(q_j^T\alpha^0 + \epsilon + \lambda q_j^T z^0\right) - \sum_{j=1}^{u} ln\left(-q_j^T\alpha^0 + \epsilon - \lambda q_j^T z^0\right).$$

In order to compute the minimizer λ^* of $\Phi(\lambda)$, the following equation

$$\frac{\partial\Phi}{\partial\lambda} = 0 \quad (5.33)$$

must be solved. Employing Newton-Raphson's method this translates into the following first-order linear approximation:

$$\frac{\partial\Phi}{\partial\lambda} + \frac{\partial^2\Phi}{\partial\lambda^2}(\lambda - \lambda_k) = 0. \quad (5.34)$$

where k denotes the number of iterations with respect to Newton's algorithm. In order to solve equation (5.34), the first and the second derivative of $\Phi(\lambda)$ must be computed. The following notation will simplify these expressions. First, let $m = K_{ext}^T z^0$, where the superscript zero refers to the fact that at this point we wish to compute α^1 from the line search. Recall the vectors of the lower and the upper slack, $s^l = K_{ext}^T\alpha - y_{ext} + \epsilon\cdot e$ and $s^u = -K_{ext}^T\alpha + y_{ext} + \epsilon\cdot e$, which can be used to define new slack vectors for the line search, more specifically, $c^l = s^l + \lambda m$, and $c^u = s^u - \lambda m$. Note that

$c^l, c^u \in \Re^{l+v}$, indicating that the additional set of slabs that compactifies the feasible region is included here. Next, let

$$C_l = diag\left(c^l\right) = S_l + \lambda \cdot diag\left(m\right) \tag{5.35}$$

$$C_u = diag\left(c^u\right) = S_u - \lambda \cdot diag\left(m\right).$$

Using the above variables the first and second derivatives for the line search result in

$$\frac{\partial \Phi}{\partial \lambda} = m^T C_u^{-1} e - m^T C_l^{-1} e \tag{5.36}$$

and

$$\frac{\partial^2 \Phi}{\partial \lambda^2} = m^T C_u^{-2} m + m^T C_l^{-2} m, \tag{5.37}$$

which allows us to compute λ^* that solves equation (5.34) and minimizes $\Phi\left(\lambda\right)$ along z^0. After that, α^1 can be calculated from equation (5.22). Next, the slack values are updated and the gradient $\nabla\Phi\left(\alpha^1\right)$ is computed before a new descent direction is determined from (5.25) and (5.24). Let us review all the steps involved in the two-phase minimization process:

1. Phase I: Set $j = 0$ and find an initial feasible solution $\alpha_{init} = \alpha^j$ as outlined in section 5.3. Select $\epsilon > \epsilon_{min}$.

2. Phase II: Compute the first descent direction $z^j = -\nabla\Phi\left(\alpha^j\right)$ using equation (5.30).

3. Phase II: Perform a line search solving equation (5.34) and update α according to $\alpha^{j+1} = \alpha^j + \lambda^j z^j$. Set $j = 1$.

4. Phase II: Compute the new slacks and the new gradient $\nabla\Phi(\alpha^j)$ from equation (5.28) and (5.30).

5. Phase II: Compute t as described in (5.25).

6. Phase II: Find the new descent direction $z^j = -\nabla\Phi(\alpha^j) + t \cdot z^{j-1}$ (5.24).

7. Phase II: Perform a line search solving equation (5.34) and update α according to $\alpha^{j+1} = \alpha^j + \lambda^j z^j$. Increase j by one and go to step 4.

Next, a simple example further illustrates the mechanics behind an analytic center machine for regression.

5.5 Regression Training: An Example

We wish to approximate a sequence of three points ($l = 3$) as shown in Table 5.1.

Table 5.1: Example Data for Regression

Data for x	1	2	3
Data for y	1	1	2

Before the actual algorithm can be executed, all necessary variables must be calculated. Let y denote the set of target values $y^T = (1, 1, 2)$ and introduce the kernel function that will be used, $k(x_i, x_j) = x_i x_j + 1$. Since a linear kernel is used, we perform

a linear regression. By simply changing the degree of the kernel function, higher polynomial regression functions can be analyzed. However, for a problem consisting of only three data samples even a quadratic kernel would already result in perfect curve fitting. Consequently, the linear programming problem in (5.20) would identify already the optimal coefficients, rendering the phase II search for the analytic center unnecessary. Define the coefficient vector $\alpha = (\alpha_1, \alpha_2, \alpha_3, b) \in \Re^4$.

Next, let us consider the kernel matrix K. For the first element $k_{11} = k(x_1, x_1) = x_1 \cdot x_1 + 1$ one finds $k_{11} = 2$. Repeating this procedure results in $k_{21} = 3$ and $k_{31} = 4$. Hence, k_1^T becomes $k_1^T = (k_{11}, k_{21}, k_{31}, 1) = (2, 3, 4, 1)$ and the kernel matrix can be expressed as follows:

$$K = [k_1, k_2, k_3] = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 5 & 7 \\ 4 & 7 & 10 \\ 1 & 1 & 1 \end{bmatrix}.$$

For the vectors spanning the null space one finds $q_1^T = (1, -2, 1, 0)$ and $q_2^T = (-2, 1, 0, 1)$. Recall that the null space has at least the dimension one, here however $v = 2$, which means that k_1, k_2, k_3 are not linearly independent. Consequently,

$$Q = [q_1, q_2] = \begin{bmatrix} 1 & -2 \\ -2 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Finally, let us spell out K_{ext} and y_{ext}

$$K_{ext} = [k_1, k_2, k_3, q_1, q_2] = \begin{bmatrix} 2 & 3 & 4 & 1 & -2 \\ 3 & 5 & 7 & -2 & 1 \\ 4 & 7 & 10 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad y_{ext} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \\ 0 \end{bmatrix},$$

where y_{ext} has been extended by exactly v zeros. We are now ready to formulate the linear programming problem, which will provide the initial feasible solution as well as information about the smallest possible ϵ. Spelling out the equations yields

$$\min \epsilon$$

$$\text{s.t.} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 5 & 7 & 1 \\ 4 & 7 & 10 & 1 \\ 1 & -2 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ b \end{bmatrix} \leq \epsilon \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 5 & 7 & 1 \\ 4 & 7 & 10 & 1 \\ 1 & -2 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ b \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \leq \epsilon \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

While the objective function attains its minimum at $\epsilon_{min} = 0.25$, the corresponding

optimal coefficients are $\alpha_{init}^T = (0.025, 0.075, 0.100, 0.025)$. In other words, for $\epsilon = \epsilon_{min} = 0.25$ the feasible region degenerates to a simple point. Therefore, the decision maker must choose $\epsilon \geq 0.25$. In this example $\epsilon = 0.5$ is selected for the analytic center analysis. Now, we are ready to proceed to phase II. Let

$$
\begin{aligned}
s_1^l &= k_1^T\alpha - y_1 + \epsilon \geq 0 \\
s_2^l &= k_2^T\alpha - y_2 + \epsilon \geq 0 \\
s_3^l &= k_3^T\alpha - y_3 + \epsilon \geq 0 \\
s_4^l &= q_1^T\alpha - 0 + \epsilon \geq 0 \\
s_5^l &= q_2^T\alpha - 0 + \epsilon \geq 0 \\
s_1^u &= -k_1^T\alpha + y_1 + \epsilon \geq 0 \\
s_2^u &= -k_2^T\alpha + y_2 + \epsilon \geq 0 \\
s_3^u &= -k_3^T\alpha + y_3 + \epsilon \geq 0 \\
s_4^u &= -q_1^T\alpha + 0 + \epsilon \geq 0 \\
s_5^u &= -q_2^T\alpha + 0 + \epsilon \geq 0
\end{aligned}
$$

be the lower and the upper slacks. Then the optimization problem

$$\min \Phi(\alpha) = -lns_1^l - lns_2^l - lns_3^l - lns_4^l - lns_5^l - lns_1^u - lns_2^u - lns_3^u - lns_4^u - lns_5^u$$

will identify the analytic center closest to the origin.

Iteration 1:

We start with $(\alpha^0)^T = (0.050, 0.075, 0.100, 0.025)$ from Phase I and compute the slacks from equations (5.16) and (5.18) or equivalently from (5.28) and write them in matrix form:

$$S_u^0 = \begin{bmatrix} 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \quad S_l^0 = \begin{bmatrix} 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

Here, the superscript "zero" refers to the initial situation before completion of the first iteration. From the slack matrices the gradient of the barrier function can be determined. Using equation (5.30) $\nabla\Phi(\alpha)$ becomes

$$g^0 = \nabla\Phi(\alpha) = K_{ext}\left(S_u^0\right)^{-1} e - K_{ext}\left(S_l^0\right)^{-1} e = \begin{pmatrix} -8.0000 \\ -13.3333 \\ -16.6667 \\ -2.6667 \end{pmatrix},$$

while in the absence of a previous descent direction we can write for z^0 from (5.23):

$$z^0 = -g^0 = \begin{pmatrix} 8.0000 \\ 13.3333 \\ 16.6667 \\ 2.6667 \end{pmatrix},$$

Before we minimize $\Phi(\alpha)$ along z^0 define:

$$m = K_{ext}^T z^0 = \begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 5 & 7 & 1 \\ 4 & 7 & 10 & 1 \\ 1 & -2 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} 8.0000 \\ 13.3333 \\ 16.6667 \\ 2.6667 \end{pmatrix} = \begin{pmatrix} 133.3333 \\ 224.0000 \\ 314.6667 \\ 0 \\ 0 \end{pmatrix},$$

Next, the new iterate $\alpha^1 = \alpha^0 + \lambda^* z^0$ is computed using Newton's approximation solving equation (5.34) which is:

$$\frac{\partial \Phi}{\partial \lambda} + \frac{\partial^2 \Phi}{\partial \lambda^2}(\lambda - \lambda_k) = 0.$$

Let the initial solution be $\lambda_0 = 0$. Then the slack matrices for the line search C_u and C_l are identical to S_u^0 and S_l^0 as described in equation (5.35). Note that no indices are assigned to C_u and C_l to increase readability. Nonetheless, the reader should be aware that j refers to the iteration of the outer loop updating the α^j, while an inner loop consists of the line search finding an optimal λ^* for each α^j. For $\lambda_0 = 0$ one finds:

$$C_u = S_u - \lambda \cdot diag\,(m) = S_u = \begin{bmatrix} 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix},$$

$$C_l = S_l + \lambda \cdot diag\,(m) = S_l = \begin{bmatrix} 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

Now calculate the first and second derivative of $\Phi(\lambda)$ using expressions (5.36) and (5.37).

$$\begin{aligned} \frac{\partial \Phi}{\partial \lambda} &= m^T C_u^{-1} e - m^T C_l^{-1} e = -597.3333 \\ \frac{\partial^2 \Phi}{\partial \lambda^2} &= m^T C_u^{-2} m + m^T C_l^{-2} m = 2.9683 \cdot 10^6 \end{aligned}$$

Using a rather weak stopping criterion of $\left|\frac{\partial \Phi}{\partial \lambda}\right| \leq 0.01$ Newton's algorithm terminates after one iteration with $\lambda^* = 2.0124 \cdot 10^{-4}$ (experimentation appears to indicate that a careful line search does not increase the performance of the algorithm). We conclude the first iteration by updating the vector α from (5.22):

$$\alpha^1 = \alpha^0 + \lambda^* z^0 = \begin{pmatrix} 0.050 \\ 0.075 \\ 0.100 \\ 0.025 \end{pmatrix} + 2.0124 \cdot 10^{-4} \cdot \begin{pmatrix} 8.0000 \\ 13.3333 \\ 18.6667 \\ 2.6667 \end{pmatrix} = \begin{pmatrix} 0.0516 \\ 0.0777 \\ 0.1038 \\ 0.0255 \end{pmatrix}.$$

Iteration 2:

The new iterate is $(\alpha^1)^T = (0.0516, 0.0777, 0.1038, 0.0255)$ and the slack matrices become:

$$S_u^1 = \begin{bmatrix} 0.72 & 0 & 0 & 0 & 0 \\ 0 & 0.20 & 0 & 0 & 0 \\ 0 & 0 & 0.69 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \quad S_l^1 = \begin{bmatrix} 0.28 & 0 & 0 & 0 & 0 \\ 0 & 0.80 & 0 & 0 & 0 \\ 0 & 0 & 0.31 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

With these expression the gradient is updated to:

$$g^1 = K_{ext}\left(S_u^1\right)^{-1} e - K_{ext}\left(S_l^1\right)^{-1} e = \begin{pmatrix} -0.5339 \\ -0.7251 \\ -0.9163 \\ -0.3427 \end{pmatrix}.$$

Contrary to iteration 1 a previous descent direction is now available, thus, t must be

computed to determine a convex combination of the current gradient and the previous descent direction:

$$t = \frac{\nabla\Phi\left(\alpha^1\right)^T \nabla\Phi\left(\alpha^1\right)}{\nabla\Phi\left(\alpha^0\right)^T \nabla\Phi\left(\alpha^0\right)} = 0.003.$$

Next, the new descent direction z^1 can be found from

$$z^1 = -g^1 + t \cdot z^0 = \begin{pmatrix} 0.5339 \\ 0.7251 \\ 0.9163 \\ 0.3427 \end{pmatrix} + 0.003 \cdot \begin{pmatrix} 8.0000 \\ 13.3333 \\ 18.6667 \\ 2.6667 \end{pmatrix} = \begin{pmatrix} 0.5576 \\ 0.7645 \\ 0.9715 \\ 0.3506 \end{pmatrix}.$$

For the line search we have to recompute m:

$$m = K_{ext}^T z^1 = \begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 5 & 7 & 1 \\ 4 & 7 & 10 & 1 \\ 1 & -2 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} 0.5576 \\ 0.7645 \\ 0.9715 \\ 0.3506 \end{pmatrix} = \begin{pmatrix} 7.6453 \\ 13.6465 \\ 17.6477 \\ 0 \\ 0 \end{pmatrix}.$$

Again, λ is set initially to $\lambda_0 = 0$ which implies that $C_u = S_u^1$ and $C_l = S_l^1$ for the first Newton iteration (otherwise subtract or add λ times m according to equation (5.35)):

$$C_u = \begin{bmatrix} 0.72 & 0 & 0 & 0 & 0 \\ 0 & 0.20 & 0 & 0 & 0 \\ 0 & 0 & 0.69 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \quad C_l = \begin{bmatrix} 0.28 & 0 & 0 & 0 & 0 \\ 0 & 0.80 & 0 & 0 & 0 \\ 0 & 0 & 0.31 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

With m, C_u, and C_l we can solve the first-order approximation

$$\frac{\partial \Phi}{\partial \lambda} + \frac{\partial^2 \Phi}{\partial \lambda^2}(\lambda - \lambda_k) = 0.$$

Note that for other stopping criteria several Newton steps might have to be performed until a solution has been reached. Accordingly, C_u and C_l will differ from S_u^1 and S_l^1, as λ_k will no longer be zero. The first and second derivative of $\Phi(\lambda)$ become:

$$\begin{aligned} \frac{\partial \Phi}{\partial \lambda} &= m^T C_u^{-1} e - m^T C_l^{-1} e = -1.8623 \\ \frac{\partial^2 \Phi}{\partial \lambda^2} &= m^T C_u^{-2} m + m^T C_l^{-2} m = 8.7690 \cdot 10^3 \end{aligned}$$

After one iteration the optimal parameter is found to be $\lambda^* = 2.1238 \cdot 10^{-4}$. Computing α^2 yields

$$\alpha^2 = \alpha^1 + \lambda^* z^1 = \begin{pmatrix} 0.0516 \\ 0.0777 \\ 0.1038 \\ 0.0255 \end{pmatrix} + 2.1238 \cdot 10^{-4} \cdot \begin{pmatrix} 0.5576 \\ 0.7645 \\ 0.9715 \\ 0.3506 \end{pmatrix} = \begin{pmatrix} 0.0517 \\ 0.0778 \\ 0.1040 \\ 0.0256 \end{pmatrix}.$$

Iteration 3:

The new iterate is $(\alpha^2)^T = (0.0517, 0.0778, 0.1040, 0.0256)$ and the slack matrices become:

$$S_u^2 = \begin{bmatrix} 0.7215 & 0 & 0 & 0 & 0 \\ 0 & 0.2022 & 0 & 0 & 0 \\ 0 & 0 & 0.6829 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \quad S_l^2 = \begin{bmatrix} 0.2785 & 0 & 0 & 0 & 0 \\ 0 & 0.7978 & 0 & 0 & 0 \\ 0 & 0 & 0.3171 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

Calculating the gradient yields:

$$g^2 = K_{ext}\left(S_u^2\right)^{-1} e - K_{ext}\left(S_l^2\right)^{-1} e = \begin{pmatrix} -0.0955 \\ 0.0127 \\ 0.1209 \\ -0.2038 \end{pmatrix}.$$

The parameter t is derived as before from

$$t = \frac{\nabla\Phi\left(\alpha^2\right)^T \nabla\Phi\left(\alpha^2\right)}{\nabla\Phi\left(\alpha^1\right)^T \nabla\Phi\left(\alpha^1\right)} = 0.037$$

and the new descent direction z^2 results in:

$$z^2 = -g^2 + t \cdot z^1 = \begin{pmatrix} 0.0955 \\ -0.0127 \\ -0.1209 \\ 0.2038 \end{pmatrix} + 0.037 \cdot \begin{pmatrix} 0.5576 \\ 0.7645 \\ 0.9715 \\ 0.3506 \end{pmatrix} = \begin{pmatrix} 0.1162 \\ 0.0156 \\ -0.0850 \\ 0.2167 \end{pmatrix}.$$

For the line search we will need

$$m = K_{ext}^T z^2 = \begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 5 & 7 & 1 \\ 4 & 7 & 10 & 1 \\ 1 & -2 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} 0.1162 \\ 0.0156 \\ -0.0850 \\ 0.2167 \end{pmatrix} = \begin{pmatrix} 0.1559 \\ 0.0483 \\ -0.0593 \\ 0 \\ 0 \end{pmatrix}.$$

For $\lambda_0 = 0$, which implies that $C_u = S_u^1$ and $C_l = S_l^1$, the first Newton iteration gives

$$\begin{aligned} \frac{\partial \Phi}{\partial \lambda} &= m^T C_u^{-1} e - m^T C_l^{-1} e = -0.0653 \\ \frac{\partial^2 \Phi}{\partial \lambda^2} &= m^T C_u^{-2} m + m^T C_l^{-2} m = 0.4641 \end{aligned}$$

resulting in a new $\lambda_1 = 0 - \frac{-0.0653}{0.4641} = 0.1407$. For the second Newton step the matrices C_u and C_l are updated to

$$C_u = S_u^2 - \lambda \cdot diag(m) = \begin{bmatrix} 0.6996 & 0 & 0 & 0 & 0 \\ 0 & 0.1954 & 0 & 0 & 0 \\ 0 & 0 & 0.6912 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

$$C_l = S_l^2 + \lambda \cdot diag(m) = \begin{bmatrix} 0.3004 & 0 & 0 & 0 & 0 \\ 0 & 0.8046 & 0 & 0 & 0 \\ 0 & 0 & 0.3088 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

and the values for the derivatives become

$$\begin{aligned} \frac{\partial \Phi}{\partial \lambda} &= m^T C_u^{-1} e - m^T C_l^{-1} e = -0.0026 \\ \frac{\partial^2 \Phi}{\partial \lambda^2} &= m^T C_u^{-2} m + m^T C_l^{-2} m = 0.4286. \end{aligned}$$

Thus, the optimal step length for the line search is found to be $\lambda^* = 0.1407 - \frac{-0.0026}{0.4286} = 0.1469$. Now compute α^3 as before:

$$\alpha^3 = \alpha^2 + \lambda^* z^2 = \begin{pmatrix} 0.0517 \\ 0.0778 \\ 0.1040 \\ 0.0256 \end{pmatrix} + 0.1469 \cdot \begin{pmatrix} 0.1162 \\ 0.0156 \\ -0.0850 \\ 0.2167 \end{pmatrix} = \begin{pmatrix} 0.0688 \\ 0.0801 \\ 0.0915 \\ 0.0574 \end{pmatrix}.$$

Continuing this process for two more iterations one finds for $(\alpha^4)^T = (0.0704, 0.0801, 0.0897, 0.0610)$ and for $(\alpha^5)^T = (0.0705, 0.0801, 0.0897, 0.0610)$. It becomes apparent that the updated solution vector for α does not significantly change any more.

Therefore, we have found the optimal solution $\alpha^* = (0.0704, 0.0801, 0.0898, 0.0610)$. The logarithmic barrier function attains its minimum value of $\Phi(\alpha^*) = 4.9538$. Moreover, α^* represents the minimum norm solution, as $q_1^T\alpha^* = 0$ and $q_2^T\alpha^* = 0$ show that α^* is perpendicular to q_1 and q_2.

Next, it will be demonstrated, how to compute the functional values for the regression function. For example, for $x = 0$, we have $f(0) = \alpha_1 k(x_1, 0) + \alpha_2 k(x_2, 0) + \alpha_3 k(x_3, 0) + b \cdot 1 = 0.0704 \cdot (1 \cdot 0 + 1) + 0.0801 \cdot (2 \cdot 0 + 1) + 0.0898 \cdot (3 \cdot 0 + 1) + 0.0610 = 0.3013$. Similarly, $f(3) = 0.0704 \cdot (1 \cdot 3 + 1) + 0.0801 \cdot (2 \cdot 3 + 1) + 0.0898 \cdot (3 \cdot 3 + 1) + 0.0610 = 1.8013$. Figure 5.1 shows the graph of the regression training as well as the available data samples.

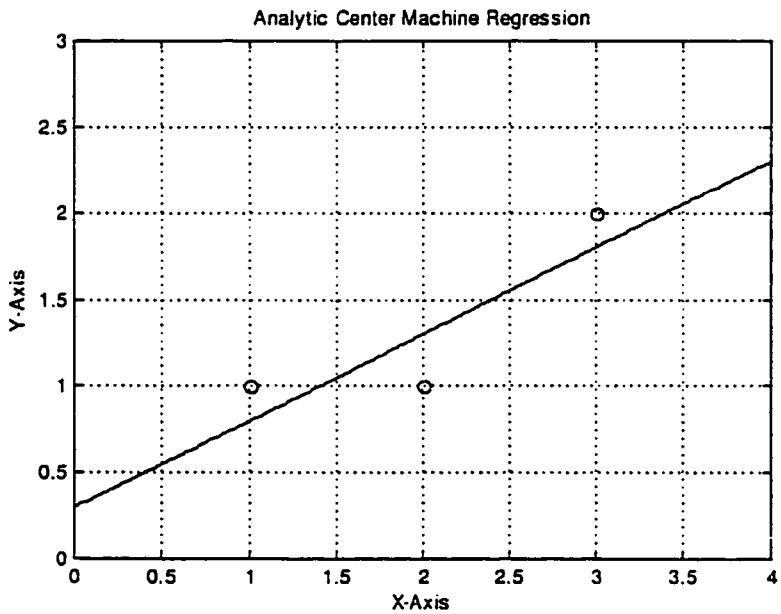


Figure 5.1: Analytic Center Machine for Regression, Linear Kernel, $\epsilon = 0.5$

Chapter 6

Computational Results

6.1 Results for Pattern Recognition

6.1.1 Tests on Synthetic Data

Preliminary experiments were conducted comparing the performance of the analytic center machine to support vector machines. Both approaches were implemented in MATLAB and compared on two datasets, which have been previously used by Herbrich et al. in [9]. Ten thousand points were randomly generated in $[-1,1]^3 \subset \Re^3$ and assigned a random label of $+1$ or -1. Hereafter, a sample of ten points was selected for training. The size of the test set comprised all 10,000 points for each one of the experiments. An exponential kernel $K(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$ with $\sigma = 1$ was used for the analysis. Table 6.1 displays the percentage values for the generalization error. As can be seen analytic center machines clearly outperform support vector machines for both experiments. Moreover, analytic center machines demonstrate a similar generalization behavior as Bayes point machines [9].

Table 6.1: Generalization Error

Dataset	SVM [9]	BPM [9]	ACM
Dataset 1	6.50	6.10	6.22
Dataset 2	15.10	8.00	8.68

Furthermore, we solved some problems graphically indicating the change of the decision surface. Figures 6.1 and 6.2 display the solutions for the analytic center machine and a support vector machine, respectively. Two classes were separated using again an exponential kernel $k(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$ with $\sigma = 1$. It can be seen that the separation achieved by the analytic center machine is much smoother than for the support vector machine. This is due to the fact that the analytic center solution is influenced by all patterns, while the support vector machine includes only information from the support vectors.

6.1.2 Tests on Standard Benchmark Data

In addition, performance of the analytic center machine was examined using three real-world datasets. The datasets `heart` and `thyroid` originate from the UCI Repository [3], while the dataset `banana` was previously used as a benchmark in [9, 23]. Each of the datasets consists of 100 training and test sets, which are randomly partitioned as follows. The datasets `heart` and `thyroid` exhibit a training to test set ratio of 60%:40%, while the dataset `banana` is partitioned based on a ratio of 10%:90% [1]. Table 6.2 shows the means and standard deviations for the support vector machine, the

[1]These datasets are publically available at http://www.first.gmd.de/ raetsch/

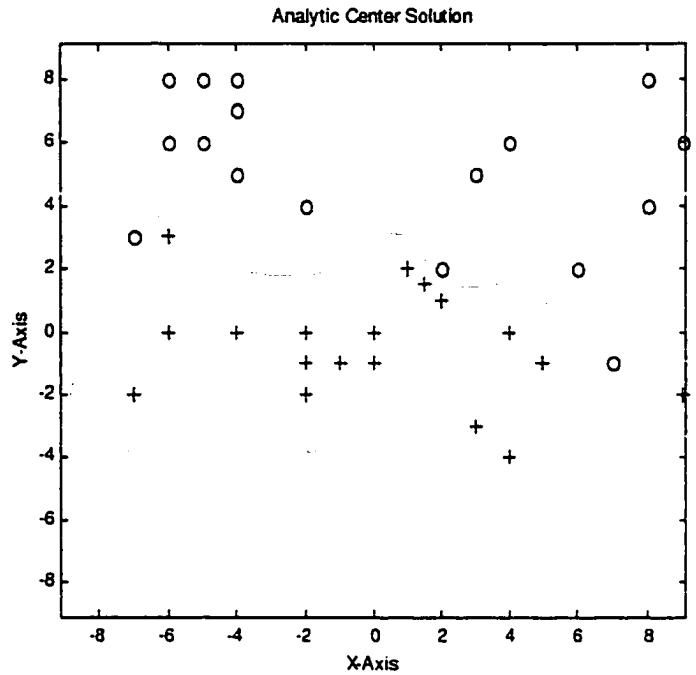


Figure 6.1: Analytic Center Solution

Bayes point machine, as well as for the analytic center machine, with the analytic center machine experiments being conducted using ACCPM [20], an analytic center optimization software. The fourth column indicates σ as it was used in the kernel function $k(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$. Finally, the fifth column represents the p-values for a pooled-variance t-test using a level of significance of 0.05 and verifying the hypothesis *SVM is greater than ACM*. The research on the analytic center machine for classification is also reported in [30].

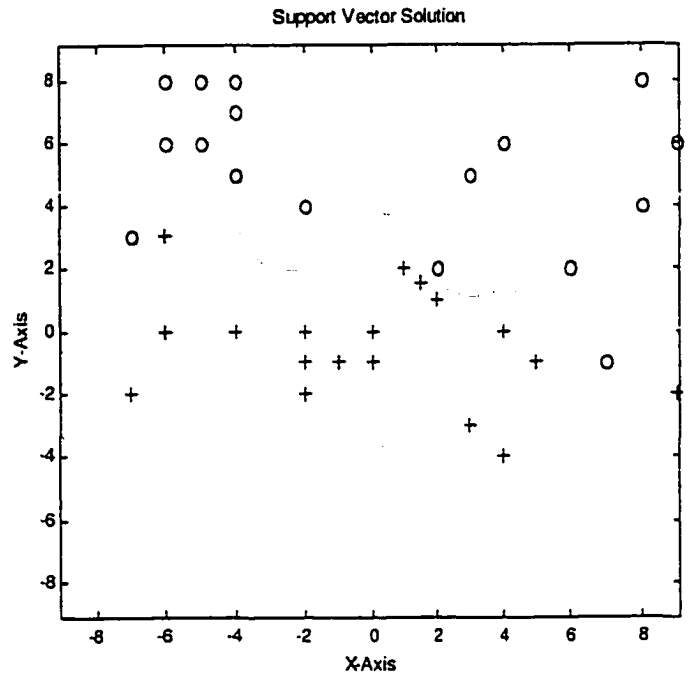


Figure 6.2: Support Vector Solution

6.2 Results for Regression Analysis

6.2.1 The 'sinc' function

Various synthetic functions were examined for experimentation. In this subsection the 'sinc' function

$$f(x) = 0.9 \cdot \frac{\sin(10x)}{10x} \tag{6.1}$$

is approximated. Figures 6.3 to 6.6 depict the behavior of $f(x)$ as well as the respective analytic center regression estimator using a polynomial kernel of the type $k(x_i, x_j) = (x_i \cdot x_j + 1)^p$.

Table 6.2: Generalization Error on Standard Benchmark Data

Dataset	SVM [9]	BPM [9]	ACM	σ	p-value
heart	25.40 ± 0.40	22.80 ± 0.34	21.87 ± 3.51	10.00	1.0000
thyroid	5.30 ± 0.24	4.40 ± 0.21	4.91 ± 2.28	3.00	0.9548
banana	16.20 ± 0.15	15.10 ± 0.14	14.73 ± 1.69	0.50	1.0000

The degree of the kernel p was varied beginning with a second degree polynomial yielding a rather mediocre approximation. For a polynomial kernel of degree six approximation of the 'sinc' function is significantly improved compared to the quadratic kernel.

The tenth degree polynomial kernel results in performance improvements around the points $x = 0.8$ and $x = -0.8$. For a degree of $p = 15$ the analytic center regression approach results in almost perfect curve-fitting on the training set. For all experiments the precision was set to $\epsilon = 0.5$.

6.2.2 Benchmark tests

Hereafter, regression performance was tested on the three Friedmann-functions, which are popular benchmark datasets in regression analysis [32]. A polynomial kernel of degree $p = 2$ was adopted and ϵ was selected as indicated in Table 6.3. For each of the three functions the training set consisted of 240 samples. The mean-square error of the regression estimator was then computed based on an average of ten runs on the training set. Performance of the analytic center machine for regression was compared to the support vector machine algorithm as outlined in equation (3.4). In addition, the

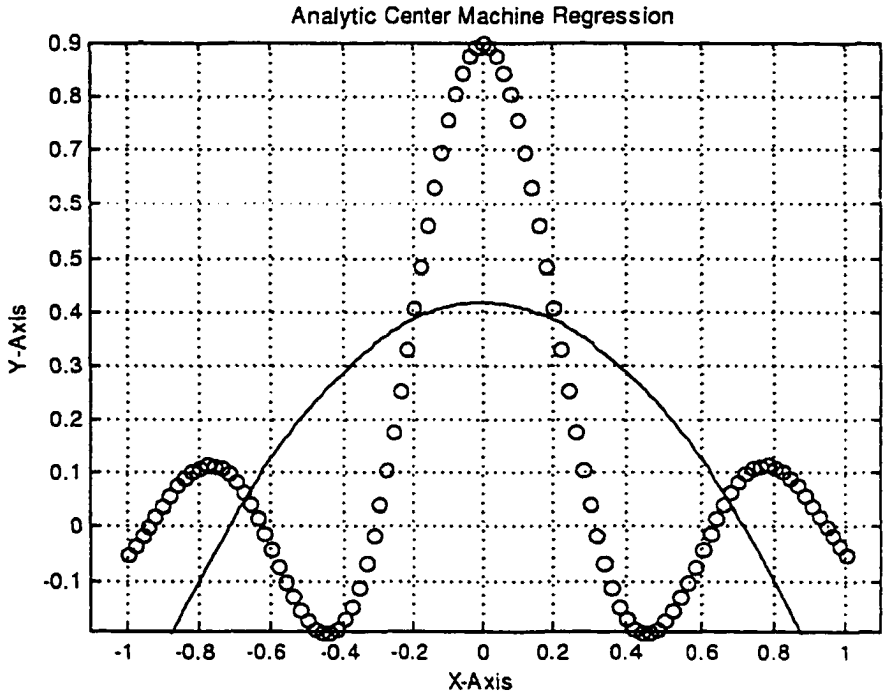


Figure 6.3: Analytic Center Machine for Regression, 2nd Degree Polynomial Kernel, $\epsilon = 0.5$

set of estimators corresponding to the initial feasible solution (α_{init}) was included as well in the column indicated MSE(LP). These results can also be found in [13].

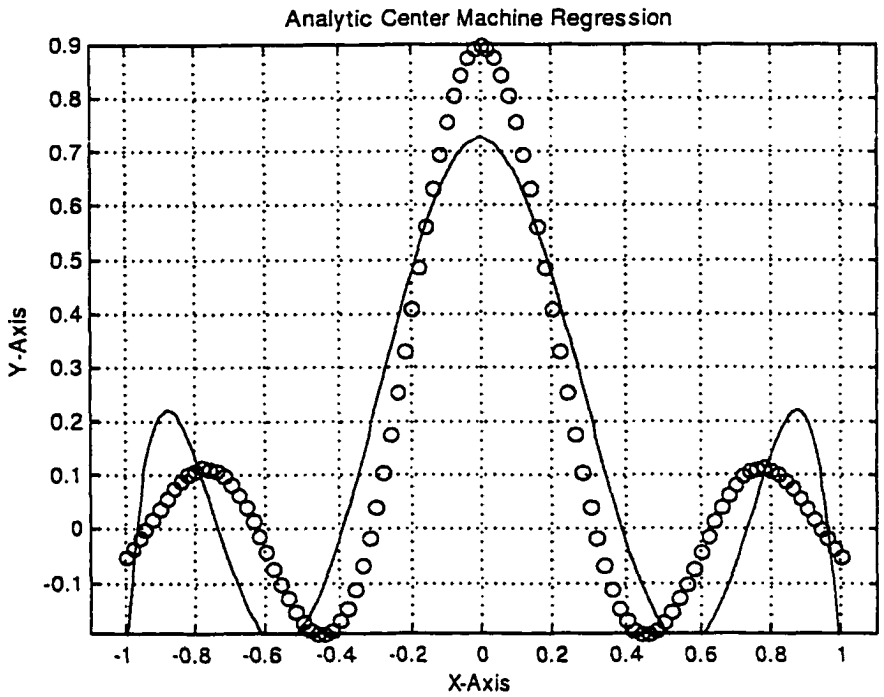


Figure 6.4: Analytic Center Machine for Regression, 6th Degree Polynomial Kernel, $\epsilon = 0.5$

Table 6.3: Experimental Results

Dataset	MSE(ACM)	MSE(LP)	MSE(SVM)	ϵ
Friedmann 1	2.41 ± 0.27	5.27 ± 3.80	18.47 ± 2.66	10
Friedmann 2	$(1.79 \pm 0.37) \cdot 10^5$	$(2.02 \pm 0.52) \cdot 10^5$	$(2.31 \pm 0.41) \cdot 10^5$	1200
Friedmann 3	0.79 ± 0.09	0.95 ± 0.16	1.32 ± 0.27	3

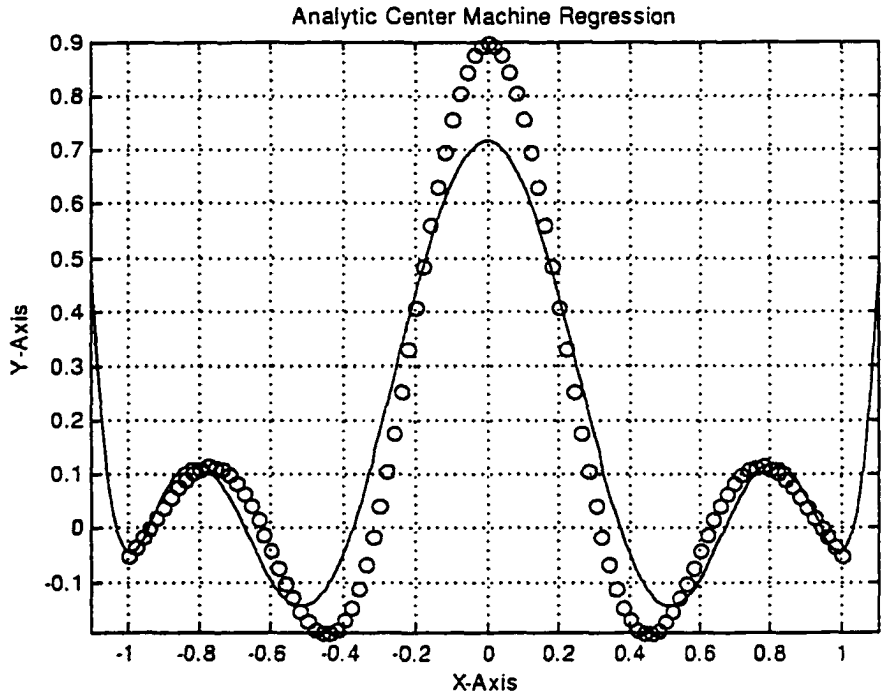


Figure 6.5: Analytic Center Machine for Regression, 10th Degree Polynomial Kernel, $\epsilon = 0.5$

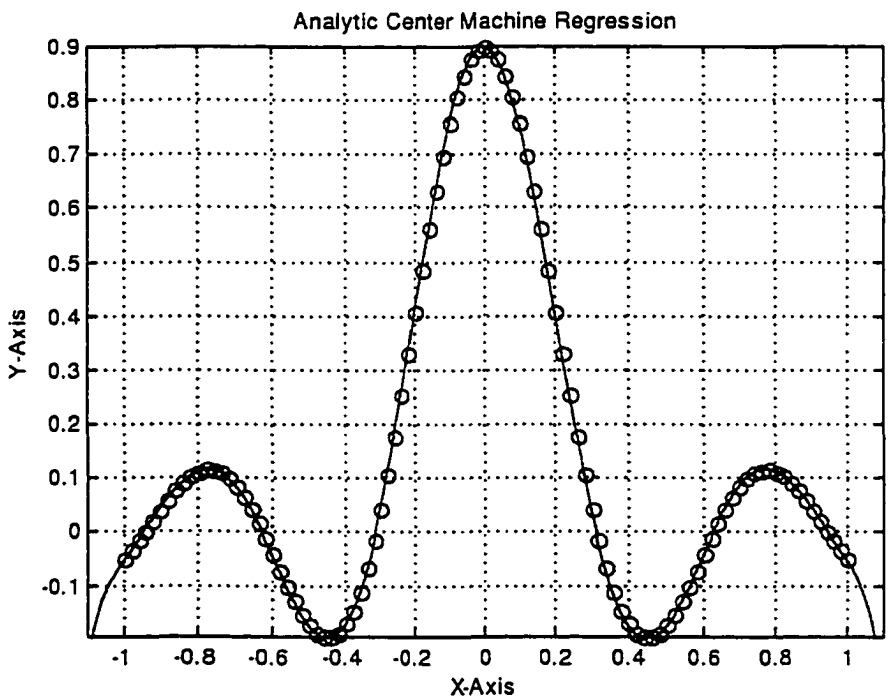


Figure 6.6: Analytic Center Machine for Regression, 15th Degree Polynomial Kernel, $\epsilon = 0.5$

Chapter 7

Summary, Conclusions and Recommendations

7.1 Summary

In this work, motivated by the support vector machine learning algorithm, we have developed new methods for solving problems in pattern classification and regression analysis. The version space of both problems was examined and the analytic center of these spaces was computed. Using the analytic center discriminant functions were computed that outperformed traditional learning machines.

7.2 Recommendations for Future Research

Preliminary experimentation has demonstrated that analytic center machines possess a promising potential for generalization. Future research might focus on an algorithm

which deletes some or all of the nonbinding constraints. This could lead to a considerable improvement in performance, when large-scale problems are discussed. In our current implementation the analytic center of the version space is computed using all patterns. Reducing the set of characteristic patterns to those that are essential in the definition of the version space would increase the sparsity of the matrices and could lead to a significant speed-up of the algorithm. We are currently investigating heuristics to delete redundant constraints that are not necessary for the description of the version space. Efficient factorization techniques can be used for solving Newton's system of equations for large-scale implementations. Alternative barrier methods are currently being investigated in [24, 22] suggesting an exponential barrier approach.

Appendix A

MATLAB Code for Pattern Recognition

```
% MAIN PROGRAM for ANALYTIC CENTER MACHINE algorithm
% Computes the center of a cone which is projected on a
% sphere of radius sqrt(2).

%                Implemented in January-March 2000
%                (c) 2000 Alexander M. Malyscheff,
%     Laboratory for Optimization and Artificial Intelligence
%                 School of Industrial Engineering
%                    The University of Oklahoma

% Global data-----------------------------------------------
```

```
clear;

global L D

global OFFSET   % polynomial kernel

global DEGREE   % polynomial kernel

global SIGMA    % exponential kernel

OFFSET = 0;

DEGREE = 1;

SIGMA = 10;


% Load input-----------------------------------------------

dummy = 1;

[patt,lab] = read(dummy);        % patterns, labels

K = [];                          % Kernel matrix, increases

C = [];                          % Patterns that form cone


alpha = ones(L+1,1);             % Initial guess

u = 1;

time = cputime;

for i = 1:L

   [K,alpha,u,C] = centeradd(K,alpha,u,D,patt,lab,i,L,C);

end

time = cputime-time
```

```
% And the result is...
for i = 1:L
   for j = 1:L
      k(j) = kernel(patt(j,:),patt(i,:));
   end
   kk = [k,1];
   result(i) = kk*alpha;
end
comparison = [lab result']
alpha

% Draw for the 2-dimensional case
if D == 3  % 2-D case
   [xvar,yvar] = meshgrid(-max(max(patt))-0.1:0.1:max(max(patt))+0.1);
   % adaptive picture size
   zvar = draw(alpha,patt,xvar,yvar,L);
   V = [0 0];
   contour(xvar,yvar,zvar,V)  % Contour line at level zero
   hold on   % Continue plotting in this graph
   for i = 1:L
      if lab(i) == 1
         aa = patt(i,1); bb = patt(i,2);
         plot(aa,bb,'+');
```

```
      elseif lab(i) == -1
         aa = patt(i,1); bb = patt(i,2);
         plot(aa,bb,'o');
      end
   end
   title('Analytic Center Solution');
   xlabel('X-Axis');
   ylabel('Y-Axis');
   hold off  % Finish plotting in this graph
end   % End plot

% Final check using the first data point
check = 0;   % We don't use the function KERNEL to check
for i = 1:L
   % Polynomial kernel
   check = check + alpha(i)*[patt(i,:)*patt(1,:)' + OFFSET]^DEGREE;
   % Exponential kernel
   %check = check + alpha(i)*exp(-((patt(i,:) -
   % patt(1,:))*(patt(i,:) - patt(1,:))')/(2*SIGMA*SIGMA));
end
check = check + alpha(L+1)

% Generalization performance
```

```
test = load('bad.ts');

[testsize,dim] = size(test);

testpatt = test(:,1:dim-1);

testlab = test(:,dim);


mistakes = 0;

for i = 1:testsize

   for j = 1:L

      testk(j) = kernel(patt(j,:),testpatt(i,:));

   end

   testkk = [testk,1];

   testresult(i) = testkk*alpha;

   if sign(testresult(i)) ~= testlab(i)

      mistakes = mistakes + 1;

   end

end

mistakes


% ***************************************************************


% M-file for reading the input data

% Reads the data and splits it in patterns and labels

% READ is called in ONLINE (MAIN PROGRAM)
```

```
function [patt,lab] = read(dummy)

global L D  % L is number of patterns, D-1 dimension (no labels)

train = load('bad.tr');

[L,D] = size(train);

patt = train(:,1:D-1);

lab = train(:,D);


% ************************************************************


% Computes dot products in feature space

% K(x_i,x_j) = Phi(x_i)'*Phi(x_j)

% KERNEL is called in ONLINE and GETK


function [out] = kernel(xone,xtwo)

global OFFSET

global DEGREE

global SIGMA


% Polynomial

% out = (xone*xtwo' + OFFSET)^DEGREE;


% Exponential
```

```
out = exp(-((xone-xtwo)*(xone-xtwo)')/(2*SIGMA*SIGMA));

% ************************************************************

% Function for the main program ONLINE
% Computes the new cone and the analytic
% center on the a sphere of radius sqrt(2)
% CENTERADD is called in ONLINE

function [K,alpha,u,C] = centeradd(K,alpha,u,D,patt,lab,i,L,C);
% C cone with all constraints, D-1 is dimension, L # of patterns
% i current pattern

k = getk(patt,lab,L,i)';

if k'*alpha < 0
   % this is the first shifting of constraint k
   % rest is done in function SHIFT
   % need to do this here to compute the total shift, total
   K = [K,k];
   [m,n] = size(K);
   s = K'*alpha;
   total = -1.1*s(n);
```

```
   s = [s(1:(n-1));-0.1*s(n)];  % now all slacks are positive
   while k'*alpha < 0
      flag = 1;
      [alpha,u,s,total] = correct(k,K,n,alpha,u,D,C,i,L,total,s);
   end
   K = K(:,1:(n-1));   % Need to remove the last column
   % column is added again in UPDATE
end

% alpha, u are now on the correct side!!!

[K,alpha,u,C] = update(k,K,alpha,u,D,C,i,L);

% ************************************************************

% GETK computes the entries of k(xi,xj) = yi*xj'*xi
% GETK is called in CENTERADD

function [k] = getk(patt,lab,L,i)

for j = 1:L        % L patterns, L+1 is b
   k(j) = kernel(patt(j,:),patt(i,:))*lab(i);
end
```

```
k = [k,lab(i)];

% ************************************************************

% CORRECT is one of the key functions
% Here a wrongly classified new sample adjusts
% the current alpha until the sample is correctly
% classified.  The constraint x_i is relaxed
% a little beyond the current alpha, then
% the projected logarithmic barrier is minimized
% until the first (new) alpha satisfies the original
% constraint.
% CORRECT is called in CENTERADD

function [alpha,u,s,total]=correct(k,K,n,alpha,u,D,C,i,L,total,s)

% Now we can start (up to) five Newton iterations as in UPDATE
sadjust = s;      % store s in sadjust, NEWTON works with sadjust
for index = 1:5
   [alpha,u,oldalpha,oldu,invjacobZ,Z]=newton(K,sadjust,n,alpha,u,L);
   s = K'*alpha;   % new slacks for alpha are computed in any case
   if min(K'*alpha) > 0 %ALL slacks need to be positive
```

```
      return
   end
   sadjust = [s(1:(n-1));total+s(n)];    % one of the key lines,
   % slack update relative to current shifted hyperplane, s(n)<0
   % need sadjust only for the FOR LOOP (NEWTON)

   % Need to check whether ALL SLACKS ARE POSITIVE? (shorter step)
   % Newton sometimes moves out of feasible region :(
   % Use oldalpha, oldu, invjacobZ, Z from NEWTON to update 'alpha'
   t = 1;
   while min(sadjust) <= 0
      t = 0.5*t;  % Stay far away from infeasibility, only half a step
      new = [oldalpha;oldu] - t*invjacobZ*Z;
      alpha = new(1:L+1); u = new(L+2);
      s = K'*alpha;
      sadjust = [s(1:(n-1));total+s(n)];
   end
end
% computes new slacks and new totalshift for new hyperplane
% in case that new alpha (after 5 newton iterations) is still bad
[s,total,n] = shift(K,alpha,n,s);  % total and s are updated

% ************************************************************
```

```
% UPDATE is one of the key functions
% Here the newton update on the projected sphere
% is performed.
% UPDATE is called in CENTERADD

function [K,alpha,u,C] = update(k,K,alpha,u,D,C,i,L)

K = [K,k];              % k is calculated in getk
s = K'*alpha; S = diag(s);
[m,n] = size(K);        % need the dimension for 'ones(n,1)'
% m = L + 1, # of patterns plus one (b)
% n changes at each iteration, goes from 1 to L

% We need to solve min phi = -log(k1'*alpha)-log(k2'*alpha)-...
% s.t. h(alpha) = 0.5*alpha'*alpha - 1 = 0

% resulting in the update system of equations:
% alphanew = alphaold  *  inv ( hessL  gradh ) * (  gradL  )
% unew       uold             ( gradh'    0  )   ( h(alpha))
%
%                             jacobZ                  Z
% L is Lagrangian
```

```
gradphi = -K*inv(S)*ones(n,1);

gradL = gradphi + u*alpha;

h = 0.5*alpha'*alpha - 1;       % Note that gradh = alpha

Z = [gradL;h];


hessphi = K*inv(S)^2*K';

jacobZ=[hessphi+u*eye(L+1),alpha;alpha',0];

new=[alpha;u]-inv(jacobZ)*Z;

alpha=new(1:L+1);

u=new(L+2);

C = [C;i]                  % New set of constraints, 'new cone'

% This is the only time

% that the set of cone constraints is updated

% even after invoking correction, update is run once!!!


% ************************************************************


% This is subroutine newton, performing up to

% five iterations to correct alpha for wrongly

% classified patterns.

% NEWTON is called in CORRECT
```

```
function [alpha,u,oldalpha,oldu,invjacobZ,Z] =
newton(K,s,n,alpha,u,L)

% oldalpha, oldu, invjacobZ, Z are needed in correct
% for a possible reduced step length (in the case,
% that two or more constraints are violated)

% We need to solve min phi = -log(k1'*alpha)-log(k2'*alpha)-...
% s.t. h(alpha) = 0.5*alpha'*alpha - 1 = 0

% resulting in the update system of equations:
% alphanew = alphaold  *  inv ( hessL  gradh ) * (  gradL  )
% unew       uold             ( gradh'    0  )   ( h(alpha))
%
%                           jacobZ                  Z
% L is Lagrangian

S = diag(s);
gradphi=-K*inv(S)*ones(n,1);
gradL=gradphi+u*alpha;
h = 0.5*alpha'*alpha - 1;      % Note that gradh = alpha
Z = [gradL;h];
```

```
hessphi = K*inv(S)^2*K';

%jacobZ in newton

[hessphi + u*eye(L+1),alpha;alpha',0];


invjacobZ = inv([hessphi + u*eye(L+1),alpha;alpha',0]);


oldalpha = alpha; % Need to store these values, in case

oldu = u;         % several slacks are negative, LINE SEARCH IN CORRECT

new = [oldalpha;oldu] - invjacobZ*Z;


alpha = new(1:L+1); % tentative alpha, u

u = new(L+2);


% ************************************************************


% Shift moves a wrongly classified pattern

% towards the current value of alpha,

% until the slack becomes positive

% with a value of 10% of the initial (neg)

% slack.

% SHIFT is called in CORRECT


function [s,total,n] = shift(K,alpha,n,s)
```

```
% newslack = -0.1*oldslack

% total is the overall shift, = -1.1*oldslack


total = -1.1*s(n);

s = [s(1:(n-1));-0.1*s(n)];


% ************************************************************


% Function DRAW displays the contour lines

% in a two dimensional case

% DRAW is called in ONLINE


function [out] = draw(alpha,patt,xvar,yvar,L)

global OFFSET

global DEGREE

global SIGMA

out = 0;

for i = 1:L

   % polynomial kernel

   out = out + alpha(i)*[patt(i,1).*xvar + patt(i,2).*yvar

   + OFFSET].^DEGREE;

   % operations are componentwise!!!

   % exponential kernel
```

```
   % out = out + alpha(i)*exp(-[(patt(i,1)-xvar).*(patt(i,1)-xvar) +
   % (patt(i,2)-yvar).*(patt(i,2)-yvar)]/(2*SIGMA*SIGMA));
end
out = out + alpha(L+1);
```

Appendix B

MATLAB Code for Regression Analysis

```
% MAIN PROGRAM for ANALYTIC CENTER MACHINE FOR REGRESSION
% algorithm, use conjugate gradient approach,
% compute linear combination of former and current
% gradient (gradphi, oldgrad)

%                    Implemented in May-July 2000
%                  (c) 2000 Alexander M. Malyscheff,
%      Laboratory for Optimization and Artificial Intelligence
%                  School of Industrial Engineering
%                     The University of Oklahoma
```

```
% CONGRAD

% Global data-------------------------------------------------

clear;

global L D

global OFFSET   % polynomial kernel

global DEGREE   % polynomial kernel

global SIGMA    % exponential kernel

global LAMBDA   % maximum step length is 80%

OFFSET = 1;

DEGREE = 1;

SIGMA = 10;


time = cputime;


% Read problem data ------------------------------------------

dummy = 1;

[K_ext,K,E,xdata,ydata,Ecol] = read(dummy);

% K is kernel matrix plus one row of ones

% E is null space extension

% Ecol is number of vectors describing nullspace
```

```
% Solve LP and find epsmin (and the initial feasible solution)
[lpminnormsol]=mineps(K,E,ydata,Ecol);
lpalpha=lpminnormsol(1:L+1)
epsmin=lpminnormsol(L+2)

% If epsmin is zero, then perfect fit ----------------------

% lpalpha is the initial starting point
alpha = lpalpha;                                        % alpha_0

% Define epsilon (depends on decision maker)  ---------------
%epsilon = 0.5;
epsilon = 2*epsmin;

% Barrier value at initial point
lpphi = barrier(alpha,epsilon,K,ydata)

% The first step of conj. grad., d0 = -grad ----------------------
[gradphi,s_u,s_l] = gradacr(alpha,K_ext,ydata,epsilon,Ecol);%g_0
oldgrad = gradphi;
d_alpha = -gradphi;                                     %d_alpha0
lambdastar = linesearch(K_ext,d_alpha,s_u,s_l,Ecol);%lambdastar0
```

```
alpha = alpha + lambdastar*d_alpha;                    %alpha_1
phi = barrier(alpha,epsilon,K,ydata)

stop = lpphi - phi  % stopping criterion
oldphi = phi;

% Conjugate gradient method ------------------------------------
while norm(stop) > 0.00001
   [gradphi,s_u,s_l] = gradacr(alpha,K_ext,ydata,epsilon,Ecol);%g_1
   norm(gradphi);
   t = (gradphi'*gradphi)/(oldgrad'*oldgrad);
   oldgrad = gradphi;          % update oldgrad for next iteration

   d_alpha = -gradphi + t*d_alpha;
   %d_alpha1 (convex combin. of gradient and d)

   % Linesearch: can not use Matlab function,
   % Nelder and Mead leaves feasible region in expansion step
   lambdastar = linesearch(K_ext,d_alpha,s_u,s_l,Ecol);%lambdastar1

   alpha = lambdastar*d_alpha + alpha;                 % alpha2
   phi = barrier(alpha,epsilon,K,ydata)
   stop = oldphi - phi
```

```
   oldphi = phi;
end
alpha
time = cputime-time

epsilon
% Compare the barrier values...
phi = barrier(alpha,epsilon,K,ydata)
lpphi=barrier(lpalpha,epsilon,K,ydata)

% Draw for the 1-dimensional input
if D == 2  % 1-D case
   [dummy2] = drawreg(alpha,xdata,ydata);
end

%------- Test on a separate test set ------------------------

%test = load('friedtest1.txt');
%[LL,D] = size(test);
%xtest = test(:,1:D-1);  % Call this xdata, need x for drawing
%ytest = test(:,D);

% ANALYTIC CENTER SOLUTION
```

```
%for i = 1:LL
%   sol(i) = 0;
%   for k = 1:L                                          % L patterns
%      sol(i) = sol(i) + kernel(xtest(i,:),xdata(k,:))*alpha(k);
%   end
%   sol(i) = sol(i) + alpha(L+1);  % Don't forget the offset b...
%end
%[sol',ytest];
%RMSE = norm(sol'- ytest) % 'sol' is a rowvector, ydata columnvector



% LINEAR PROGRAMMING SOLUTION
%for i = 1:LL
%   lpsol(i) = 0;
%   for k = 1:L                                          % L patterns
%      lpsol(i)=lpsol(i)+kernel(xtest(i,:),xdata(k,:))*lpalpha(k);
%   end
%   lpsol(i) = lpsol(i) + lpalpha(L+1);
% Don't forget the offset b...
%end
%[lpsol',ytest];
%LPRMSE = norm(lpsol'- ytest)
% 'lpsol' is a rowvector, ydata columnvectorfs
```

```
%--------- Test on the training set itself -----------------
% ANALYTIC CENTER SOLUTION
for i = 1:L
   sol(i) = 0;
   for k = 1:L                                        % L patterns
      sol(i) = sol(i) + kernel(xdata(i,:),xdata(k,:))*alpha(k);
   end
   sol(i) = sol(i) + alpha(L+1); % Don't forget the offset b...
end
%[sol',ydata];
MSE = norm(sol'- ydata)^2/240
% 'sol' is a rowvector, ydata columnvector

% LINEAR PROGRAMMING SOLUTION
for i = 1:L
   lpsol(i) = 0;
   for k = 1:L                                        % L patterns
      lpsol(i)=lpsol(i)+kernel(xdata(i,:),xdata(k,:))*lpalpha(k);
   end
   lpsol(i) = lpsol(i) + lpalpha(L+1);%Don't forget the offset b
end
%[lpsol',ydata];
```

```
LPMSE = norm(lpsol'- ydata)^2/240
% 'lpsol' is a rowvector, ydata columnvector

% ***************************************************************

% M-file for reading the input data
% Reads the data and splits it in input value
% and functional value
% READ is called in CONGRAD and MINEPS

function [K_ext,K,E,xdata,ydata, Ecol] = read(dummy)
global L D  % L is number of points, D-1 dimension (no labels)
global OFFSET
global DEGREE
global SIGMA
train=load('test5.txt');
[L,D] = size(train);
xdata = train(:,1:D-1);  % Call this xdata, need x for drawing
ydata = train(:,D);

for i = 1:L
   for j = 1:L        % L patterns
      k(i,j) = kernel(xdata(i,:),xdata(j,:));
```

```
   end
end
% K matrix, nonsymmetric, and E, the set of all vectors
% perpendicular to all vectors in K
K = [k; ones(L,1)'];
E = null(K','r');
[Erow, Ecol] = size(E);
K_ext = [K,E];

% ************************************************************

% Computes inner products in feature space
% K(xi,xj) = Phi(xi)'*Phi(xj)
% KERNEL is called in READ and CONGRAD

function [out] = kernel(xone,xtwo)
global OFFSET
global DEGREE
global SIGMA

% Polynomial
out = (xone*xtwo' + OFFSET).^DEGREE;  % DON'T FORGET THE DOT !!!
```

```
% Exponential (probably not of much use in 1-D?)
% out = exp((-((xone-xtwo)*(xone-xtwo)'))/(2*SIGMA*SIGMA));

% ************************************************************

% Subroutine to minimize epsilon s.t. the svm-regression
% constraints. This corresponds to perfect
% curve fitting. The problem
% can be stated as an LP-problem in the space of (alpha,b,eps).
% and provides guaranteed an initial starting point (alpha,b)
% Additional constraints are included to find the minnorm sol.
% MINEPS is called in CONGRAD

function [lpminnormsol] = mineps(K,E,ydata,Ecol)
global L
e=ones(L,1);

lpmatrix=[-K' -e;K' -e];
minnormmatrix=[lpmatrix;E'zeros(Ecol,1);-E' zeros(Ecol,1)];
% (include constraints that extract the minnorm solution)

lprhs = [-ydata;ydata];
minnormrhs = [lprhs;zeros(2*Ecol,1)];
```

```
% (include constraints that extract the minnorm solution)

lpobjfun = [zeros(L+1,1);1];  % No change for minnorm necessary

% Compute LP-solution:
lpminnormsol = lp(lpobjfun,minnormmatrix,minnormrhs);

% ***************************************************************

% Computes the logarithmic barrier function
% as a function of k, coeff=(alpha,bias),y, and epsilon.
% BARRIER is used in CONGRAD

function [phi] = barrier(alpha,epsilon,K,ydata)
global L
% We use K and not K_ext computing the barrier using
% the original UNALTERED problem
e = ones(L,1);
s_l =  K'*alpha - ydata + epsilon*e;
s_u=-K'*alpha+ydata + epsilon*e;

phi = -e'*log(s_l) - e'*log(s_u);
```

```
% ************************************************************

% Function GRADACR computes the gradient
% for the conjugate gradient method
% First used as 'gradphi', then passed
% on to 'oldgrad' to compute 't'
% GRADACR is called in CONGRAD

function [gradphi,s_u,s_l]=
gradacr(alpha,K_ext,ydata,epsilon,Ecol)
global L
% We need to account for "Ecol" add'l constraints
e = ones(L+Ecol,1);
ydata_ext = [ydata;zeros(Ecol,1)];

% Lower and upper slacks:
s_l =  K_ext'*alpha - ydata_ext + epsilon*e;
s_u=-K_ext'*alpha+ydata_ext + epsilon*e;

% Matrix notation:
S_l=diag(s_l);
S_u=diag(s_u);
invS_l=inv(S_l);
```

```
invS_u=inv(S_u);


factor = (invS_u - invS_l)*e;
% Compute the gradient for all patterns using
% Grad = f1*k1 + f2*k2 + ... + fl*kl
% which can be written using matrix notation:
gradphi = K_ext*factor;


% ***************************************************************


% LINESEARCH computes lambdastar, for
% which the linesearch achieves the
% minimum value.  In order to be on the
% safe side, we use the compact feasible region,
% i.e. K_ext instead of K.
% LINESEARCH is used in CONGRAD


function [lambdastar] = linesearch(K_ext,d_alpha,s_u,s_l,Ecol)
global L


e = ones(L+Ecol,1);
lambda = 0;
% The problem for the upper slacks can be formulated as:
```

```
% min Phi = -ln(-K_ext*alpha+ydata_ext+epsilon - lambda*K_ext*d_alpha)
% which can be written as:
% min Phi = -ln(s_1^u - lambda*v_1)

v = K_ext'*d_alpha;
stop = 2;

while norm(stop) > 0.01
   lambdagrad = [v./(s_u - lambda*v)]'*e - [v./(s_l + lambda*v)]'*e;
   lambdahess = [v./(s_u - lambda*v)]'*[v./(s_u - lambda*v)]
                + [v./(s_l + lambda*v)]'*[v./(s_l + lambda*v)];
   lambda = -lambdagrad/lambdahess + lambda;
   stop = -lambdagrad/lambdahess;
end lambdastar = lambda;

% ************************************************************

% Function DRAWREG displays
% the regression for a one
% dimensional case
% DRAWREG is called in CONGRAD

function [dummy2] = drawreg(alpha,xdata,ydata)
```

```
global L

dummy2=1;

x = (min(xdata)-0.1:0.01:max(xdata)+0.1)';  %+- 0.1

lx = length(x);


% Compute output matrix OUTK, like K, but for lx (many more) values

for i = 1:lx          % lx data points

   for j = 1:L        % L patterns

      outk(i,j) = kernel(x(i,:),xdata(j,:));

   end

end

outk;

OUTK = [outk'; ones(lx,1)'];    % Just to be conform to K and K'

y = OUTK'*alpha;


%figure(1)

%subplot(2,2,1)

%h = plot(x,coeff);

plot(x,y)

axis([min(xdata)-0.1 max(xdata)+0.1 min(ydata) max(ydata)])

grid on

xlabel('X-Axis')

ylabel('Y-Axis')
```

```
%legend(h,'first')

title('Analytic Center Machine Regression')

hold on   % Continue plotting in this graph

for i = 1:L

   aa = xdata(i); bb = ydata(i);

   plot(aa,bb,'o');

end

hold off  % Finish plotting in this graph


% ************************************************************
```

Appendix C

Standard Benchmark Data

The dataset `heart` contains information on heart disease diagnoses which were compiled by several hospitals in the United States and Europe. The dimension of the input vector is 13, carrying mostly medical information, but also for example the patient's age and sex. The output value has dimension one indicating the diagnosis for each patient. For the experiments in this study 170 patterns (patients) were considered for training and 100 patterns were used for testing. The `thyroid` disease records were supplied by the Garavan Institute, Sydney, Australia. Similarly to the `heart` dataset `thyroid` contains medical information on patients having thyroid problems. The input vector consists of 5 attributes while the output value is again of dimension one. Training was conducted on 140 patterns with the resulting discriminant function being tested against 75 patterns. The dataset `banana` consists of artificial data derived from several Gaussian blobs in two dimensions. The training set comprised 400 patterns while the test set contained 4900 patterns. For more information on the datasets see also the UCI Repository [3].

Bibliography

[1] M.Z. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & Sons, New York, 1993.

[2] D.P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Massachusetts, 1999.

[3] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[4] B.E. Boser, I.M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[5] M. Bouten, J. Schietse, and C. van den Broeck. Gradient descent learning in perceptrons: A review of its possibilities. *Physical Review E*, 52(2):1958–1967, 1995.

[6] C.J.C. Burges. A tutorial on support vector machines for pattern classification. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[7] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

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

[9] R. Herbrich, T. Graepel, and C. Campbell. Robust bayes point machines. *Proceedings of ESANN 2000*, 2000. In press.

[10] T. Joachims. Making large-scale svm learning practical. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 169–184. MIT Press, 1999.

[11] M.J. Kaiser, T.L. Morin, and T.B. Trafalis. Centers and invariant points of convex bodies. In P. Gritzmann and B. Sturmfels, editors, *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 367–385. American Mathematical Society, Providence, RI, 1991.

[12] L. Kaufmann. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 147–168. MIT Press, 1999.

[13] A.M. Malyscheff and T.B. Trafalis. An analytic center machine for regression. Technical report, Laboratory for Optimization and Intelligent Systems, School of

Industrial Engineering, University of Oklahoma, 2000. Submitted to *European Journal of Operational Research* .

[14] O.L. Mangasarian and D.R. Musicant. Massive support vector regression. Technical report, Data Mining Institute Technical Report 99-02, Computer Sciences Department, University of Wisconsin, 1999.

[15] O.L. Mangasarian, W.N. Street, and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.

[16] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[17] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In *Proceedings of IEEE NNSP '97*, 1997.

[18] Y.E. Nesterov and A.S. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM Publications, Philadelphia, 1994.

[19] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. *Proc. Computer Vision and Pattern Recognition '97*, pages 130–136, 1997.

[20] O. Péton and J.-Ph. Vial. A tutorial on ACCPM. Technical report, HEC/Logilab, University of Geneva, 2000. http://ecolu-info.unige.ch/logilab/software/accpm/index.html.

[21] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 185–208. MIT Press, 1999.

[22] G. Rätsch, A. Demiriz, and K. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. Technical report, Royal Holloway College, London, 2000.

[23] G. Rätsch, T. Onoda, and K.-R. Müller. Soft Margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001. In press.

[24] G. Rätsch, B. Schölkopf, S. Mika, and K.-R. Müller. SVM and Boosting: One Class. Technical report, GMD FIRST, Berlin, 2000.

[25] P. Ruján. Playing billard in version space. *Neural Computation*, 9:99–122, 1997.

[26] A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers*. The MIT Press, Cambridge, Massachusetts, 2000.

[27] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 1998. Invited paper, in press.

[28] A.J. Smola. *Learning with Kernels*. PhD thesis, Technical University Berlin, 1998.

[29] G. Sonnevend. An analytical center for polyhedrons and then new classes of global algorithms for linear (smooth, convex) programming. In *Lectures Notes in Control Information Sciences*, pages 866–876. Springer-Verlag, New-York, 1985.

[30] T.B. Trafalis and A.M. Malyscheff. An analytic center machine. *Machine Learning*, 2001. Accepted for publication.

[31] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[32] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[33] M.H. Wright. Why a pure primal newton barrier step may be infeasible. *SIAM Journal on Optimization*, 5(1):1–12, 1995.