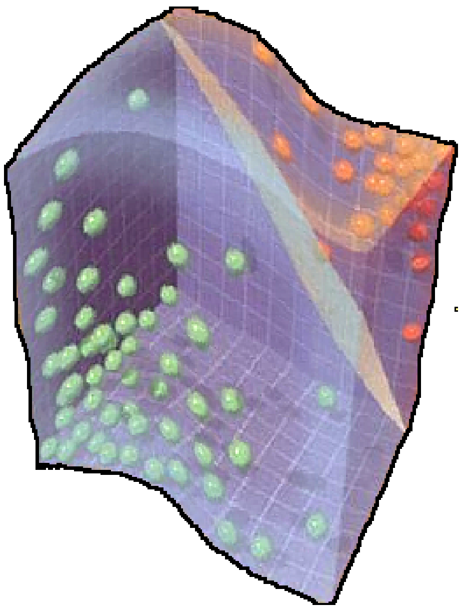
Feature space learning in Support Vector Machines through Dual Objective optimization

Auke-Dirk Pietersma
August 2010

Master Thesis
Artificial Intelligence
Department of Artificial Intelligence,
University of Groningen, The Netherlands



Supervisors:
Dr. Marco Wiering (Artificial Intelligence, University of Groningen)
Prof. dr. Lambert Schomaker (Artificial Intelligence, University of Groningen)

rijksuniversiteit
groningen

Abstract

In this study we address the problem on how to more accurately learn underlying functions describing our data, in a Support Vector Machine setting. We do this through Support Vector Machine learning in conjunction with a Weighted-Radial-basis function. The Weighted-Radial-Basis function is similar to the Radial-Basis function, in addition it has the ability to perform feature space weighing. By weighing each feature differently we overcome the problem that every feature supposedly has equal importance for learning ourk target function. In order to learn the best feature space we designed a feature-variance filter. This filter scales the feature space dimensions according to the relevance each dimension has for the target function, and was derived from the Support Vector Machine's dual objective -definition of the maximum-margin hyperplane- with the Weighted-Radial-Basis function as a kernel. The "fitness" of the obtained feature space is determined by its costs, where we view the SVMs dual objective as a cost function. Using the newly obtained feature space we are able to more precisely learn feature spaces, and thereby increase the classification performance of the Support Vector Machine.

Acknowledgment

First and foremost, I would like to thank the supervisors of this project Dr. Marco Wiering and Prof. dr. Lambert Schomaker. The discussions Marco and I had during coffee breaks were very inspiring and had great contribution to the project. Especially I would like to thank him for the effort he put in finalizing this project. For his great advice and guidance throughout this project I would like to thank Lambert Schomaker.

Besides, I would like to thank my fellow students, Jean-Paul, Richard, Tom, Mart, Ted and Robert for creating an nice working environment. The brothers, Killian and Odhran McCarthy, for comments on my text.
The brothers Romke, and Lammertjan Dam, for advice.

In addition, I would like to thank Gunn Kristine Holst Larsen for getting me back in university after a short break.

Last but certainly not least, my parents Theo and Freerkje Pietersma for their parental support and wisdom.

Notation:

- Ψ_j, jth support vector machine (SVM).
- Φ_i, ith activation function.
- y, instance label where $y \in \{-1, 1\}$
- $\hat{y}$, predicted label where $\hat{y} \in \{-1, 1\}$
- x, instance where $x \in R^n$
- $\underline{x}$, instance used as support vector (SV) and $\underline{x} \in R^n$.
- D, dataset where $D = \{(x, y)_1, .., (x, y)_m\}$.
- $D' = \{d \in D | \hat{y} \neq y\}$
- ω_j, weights used in kernel space where $\omega_j \in R^n$ and j corresponds to one particular SVM.
- $\langle \alpha \cdot \beta \rangle$, dot-product / inner-product
- $\langle \alpha \cdot \beta \cdot \gamma \rangle$, when all arguments are vectors with equal length then the function is defined as: $\sum_{i=1}^{n} \alpha_i * \beta_i * \gamma_i$

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) is one of the newest sciences, and work in this field started soon after the second World War. In today's society life without AI seems unimaginable, in fact we are constantly confronted with intelligent systems. On a daily basis we use *Google* to search for documents and images, as a break we enjoy playing a game of *chess* against an artificial opponent. Also imagine what the contents of your mailbox would look like if there were no intelligent spam filters: V1@gra, V—i—a—g—r—a, via_gra and the list goes on and on. There are 600,426,974,379,824,381,952[1] different ways to spell Viagra. In order for an intelligent system to recognize the real word "viagra" it needs to learn what makes us associate a sequence of symbols - otherwise known as "a pattern"- with viagra.

One branch of AI is Pattern Recognition. Pattern Recognition is the research area within AI that studies systems capable of recognizing patterns in data. It goes without saying that not all systems are designed to trace unwanted emails. *Handwriting Recognition* is a sub field in Pattern Recognition whose research has enjoyed several practical applications. These applications range from determining ZIP codes from addresses [17] to digitizing complete archives. The latter is a research topic within the Artificial Intelligence and Cognitive Engineering group at the university of Groningen. The problem there is not simply one pattern but rather 600km of book shelves which need to be recognized and digitized [1]. Such large quantities of data simply cannot be processed by humans only and require the help of intelligent systems, algorithms and sophisticated learning machines.

1.1 Support Vector Machines

Support Vector Machines, in combination with Kernel Machines, are "state of the art" learning machines capable of handling "real-world" problems. Most

[1]http://cockeyed.com/lessons/viagra/viagra.html

of the best classification performances at this moment are in hands of these learning machines [15, 8]. The roots of the Support Vector Machine (SVM) come from statistical learning theory, as first introduced by Vapnik [25, 4].

The SVM is a supervised learning method, meaning that (given a collection of binary labeled training patterns) the SVM algorithm generates a predictive model capable of classifying unseen patterns to either category. This model consists of a hyperplane which separates the two classes, and is formulated in terms of "margin-maximization". The goal of the SVM is to create as large as possible a margin between the two categories. In order to generate a strong predictive model the SVM algorithm can be provided with different mapping functions. These mapping functions are called "Kernels" or "kernel functions". The Sigmoid and Radial-Basis function (RBF) are examples of such kernel functions and are often used in a Support Vector Machine and Kernel Machine paradigm. These kernels however do not take into account the relevance each feature has on the target function. In order to learn the underlying function that describes our data we need to learn the relevance of the features. In this study we intend to extend the RBF kernel by adding to it the ability to learn more precisely how the features describe our target function. Adding a weight vector to the features in the RBF kernel results in a Weighted-Radial-Basis function (WRBF). In [21] the WRBF kernel was used in a combination with a genetic algorithm to learn the weight vector. In this study we will use the interplay between the SVM's objective function and the WRBF kernel to determine the optimal weight vector. We will show that by learning the feature space we can further maximize the SVM's objective function which corresponds to greater margins, which answers the following question:

How can a Support Vector Machine maximize its predictive capabilities through feature space learning?

1.2 Outline

The problem of margin-maximization is formulated as a quadratic programming optimization problem. However the basic mathematical concepts are best explained using relatively simple linear functions. Therefore we start the theoretical background with Chapter (2) on linear discriminant functions. In this Chapter we introduce the concepts of separating hyperplanes, and how to solve classification problems that are of linear nature. The SVM will be introduced in Chapter (3), alongside its derivation and concepts we have implemented the SVM algorithm (PySVM) of which a snippet can be found in the appendix. We will use PySVM to give illustrative examples on different types of kernels and the concepts of hard margin and soft margin optimization. Chapter (4) focuses on the concept and theory behind kernels. Furthermore we will give a short introduction to the notion of “Mercer kernels” and will show that the WRBF kernel is a Mercer kernel. Chapter (5) begins with describing how the WRBF kernel is able to learn the target function through “error minimization” and “margin maximization”. We will present the feature-variance-filter algorithm which allows the WRBF to more accurately learn the feature space. The experiments in Chapters (7,8,9,10) that we have conducted are introduced in section (6.1). These experiments range from feature selection to feature weighing. The final two Chapters will address conclusion and future work.

Part I

Theoretical Background

Chapter 2

Linear Discriminant Functions

2.1 Introduction

Linear Discriminant Functions (LDFs) are methods used in statistics and machine-learning, and are often used for classification tasks [9, 18, 11, 22]. A discriminant function realizes this classification by class separation, often referred to as discriminating a particular class, hence "Linear Discriminant Functions". Principle Component Analysis [9] and Fischer Linear Discriminant [9] are methods closely related to LDFs. LDFs posses several properties which make them very useful in practice: (1) Unlike parametric estimators, such as "maximum-likelihood", the underlying density probabilities do not need to be known; we examine the "space" in which the data thrives and not the probabilities coinciding with its features and occurrences. (2) Linear Discriminant Functions are fairly easy to compute making them suitable for a large number of applications.

This chapter consist of the following: (1) defining the basic idea behind linear discriminant functions and their geometric properties; (2) solving the linear separable case with two examples; and (3) solving for the linear inseparable case.

2.2 Linear Discriminant Functions and Separating Hyperplanes

2.2.1 The Main Idea

A discriminant function that is a linear function of the input $\mathbf{x}$ is formulated as:

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b \tag{2.1}$$

Where $\mathbf{w}^t$ is a weight vector and b the bias which describes displacement.

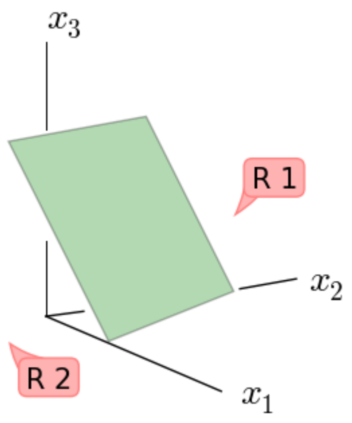


Figure 2.1: Affine hyperplane in 3D environment. The hyperplane separates space into two separate regions $\{R_1, R_2\}$. These regions are used to classify patterns in that space.

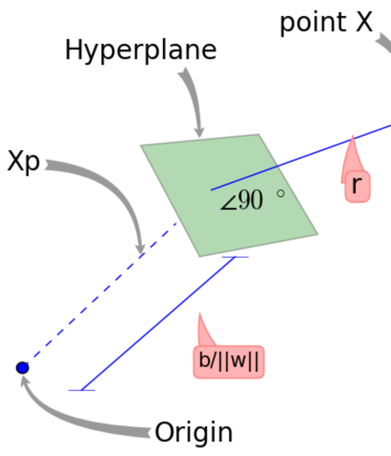
Equation (2.1) describes the well known hyperplane, graphically represented in figure (2.1). It can be seen that the affine hyperplane[1] H divides the space $S(x_1, x_2, x_3)$ into two separate regions, $\{R_1, R_2\}$. These two regions form the basis for our classification task, namely we want to divide S in such a manner that in a binary classification $\{1, -1\}$ problem all the positive labels are separated from the negative labels by H. In a linear separable binary classification task this means there exists no region $r \in R$ in which more than one particular class is positioned.

2.2.2 Properties

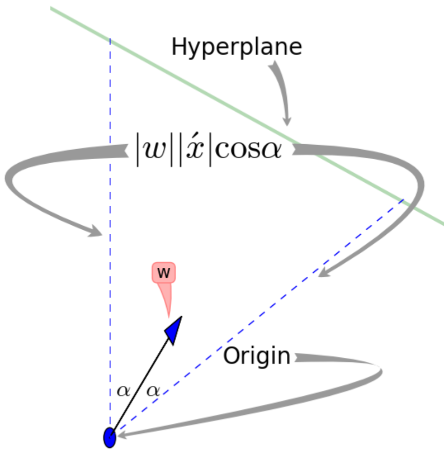
The regions $\{R_1, R_2\}$ as described in the previous subsection are often referred to as the positive and negative side of H, denoted as R_+ and R_-. The offset of a point x in S not lying on H can have either positive or negative offset to H, determined by the region it is positioned in. The latter is similar to: $R_+ = \{\mathbf{x}|g(\mathbf{x}) > 0\}$ and $R_- = \{\mathbf{x}|g(\mathbf{x}) < 0\}$, making equation (2.1) an algebraic distance measurement from a point x to H, and a membership function where x can be member of $\{R_-, R_+, H\}$. Figure (2.2a) illustrates properties for any point x in space $S(x_1, x_2, x_3)$ which can be formulated as:

[1]An affine hyperplane is a (d-1)-dimensional hyperplane in d-dimensional space

$$\mathbf{x} = \mathbf{x_p} + r\frac{\mathbf{w}}{||\mathbf{w}||} \tag{2.2}$$



(a) All points in *Space* can be formulated in terms of distance $|r|$ to the hyperplane. The sign of r can then be used to determine on which side it is located.



(b) Vectors ($\mathbf{x}$) that satisfy $\mathbf{w}^t\mathbf{x}' = -b$ give the displacement of the hyperplane with respect to the origin.

Figure 2.2: By describing points in terms of vectors, it is possible to give these points extra properties. These properties describe its location and offset with respect to certain body.

Where $\mathbf{x_p}$ is a vector from the origin to the hyperplane such that the hyperplane's "grabbing point" has a $90\,^\circ$ angle towards $\mathbf{x}$. This perpendicular vector ensures that we have the shortest distance from H to $\mathbf{x}$. Since we know that for any point on H it holds that: $\mathbf{w}^t\mathbf{x} + b = 0$, we can derive: $r = \frac{g(\mathbf{x})}{||\mathbf{w}||}$, this we show later. We are particularly interested in this r, since its magnitude describes the distance to our hyperplane, and we use its sign for classification. Inserting the new formulation for $\mathbf{x}$ (equation (2.2)) in the discriminant g we obtain r. Equations (2.3) through (2.8) show its derivation and figure (2.2a) gives a graphical impression on the interplay between $\mathbf{x}$ and $\mathbf{x_p}$, r.

$$\begin{aligned}
g(\mathbf{x}) &= \mathbf{w}^t\mathbf{x} + b && (2.3)\\
g(\mathbf{x_p} + r\frac{\mathbf{w}}{||\mathbf{w}||}) &= \mathbf{w}^t\left(\mathbf{x_p} + r\tfrac{\mathbf{w}}{||\mathbf{w}||}\right) + b && (2.4)\\
&= \mathbf{w}^t\mathbf{x_p} + r\tfrac{\mathbf{w}^t\mathbf{w}}{||\mathbf{w}||} + b && (2.5)\\
&= \left[r\tfrac{\mathbf{w}^t\mathbf{w}}{||\mathbf{w}||}\right] + \left[\mathbf{w}^t\mathbf{x_p} + b\right] && (2.6)
\end{aligned}$$

All points located on the hyperplane ($\mathbf{w}^t\mathbf{x} + b = 0$) have zero distance, and therefore trivial rearranging on (2.6) leads to (2.7).
Notice that by definition:

$$\frac{\mathbf{w}^t\mathbf{w}}{||\mathbf{w}||} = \frac{\sum w_i w_i}{\sqrt{\sum w_i w_i}} = \sqrt{\sum w_i w_i} = ||\mathbf{w}||$$

so that,

$$g(\mathbf{x}) = r||\mathbf{w}|| \tag{2.7}$$

giving the offset from r to H,

$$r = \frac{g(\mathbf{x})}{||\mathbf{w}||} \tag{2.8}$$

Equation (2.8) shows how we can finally calculate the offset of a point to the hyperplane.

Another property is the displacement of H to the origin O. Knowing that $\mathbf{w}$ is a normal of H, we can then use the projection for any vector $\mathbf{x}'$ satisfying $\mathbf{w}^t\mathbf{x}' = -b$ on $\mathbf{w}$ to calculate this displacement. For vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ their dot product is given by:

$$\begin{aligned}
\mathbf{x}_1 \cdot \mathbf{x}_2 &= \mathbf{x}_1^t\mathbf{x}_2\\
&= ||\mathbf{x}_1||\ ||\mathbf{x}_2|| \cos\alpha && (2.9)
\end{aligned}$$

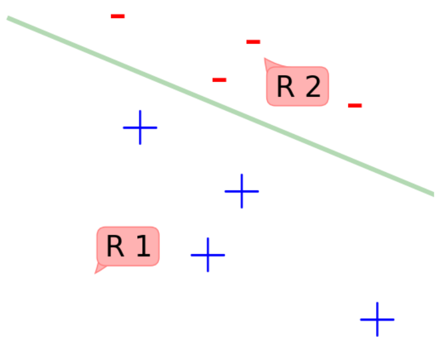
Using equation (2.9) and figure (2.2b) we can see that the projection of $\mathbf{x}$ on the *unit normal*[2] of $\mathbf{w}$ is the length from O to H. Since $\mathbf{w}$ is not per se a unit vector we need to re-scale $||\mathbf{w}||||\mathbf{x}|| \cos\alpha = -b$, giving the offset $O = \frac{b}{||\mathbf{w}||}$.

2.3 Separability

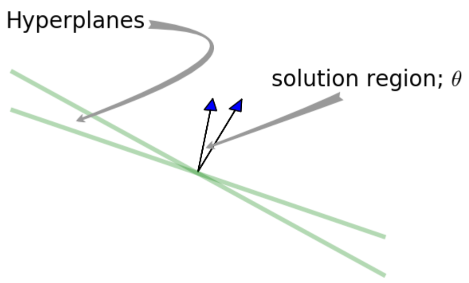
2.3.1 Case: 1 Linearly Separable

This subsection lays the foundation for finding hyperplanes capable of separating the data. A set of points in two dimensional space containing two

[2]a unit normal is vector with length 1 and is perpendicular to a particular line or body.



(a) Toy example: this toy-example shows how a hyperplane can separate the classes $\{+,-\}$ such that each class is located at most one side of the hyperplane.



(b) Solution space θ: the solution space θ contains all those vectors $(\mathbf{w})$ who's solution results in perfect class separation.

Figure 2.3: In a linear separable classification task the hyperplane often can take multiple positions. The locations the hyperplane takes are influenced by different $\mathbf{w}$.

classes is said to be linearly separable if it is possible to separate the two classes with a line. For higher dimensional spaces this holds true if there exists a hyperplane that separates the two classes. The toy example of figure (2.3a) shows how a hyperplane can be constructed that perfectly separates the two classes $\{+,-\}$.

As one can see, it is possible to draw multiple lines in figure (2.3a) that separate the two classes. The latter indicates that there exist more than one $\mathbf{w}$ that perfectly separates the two classes. Therefore our solution space θ is described as follows:

$$\begin{aligned}&\{\mathbf{w_i}|\\ &\qquad \mathbf{w_i}^t\mathbf{x}+b>0,\mathbf{x}\in R_+,\\ &\qquad \mathbf{w_i}^t\mathbf{x}+b<0,\mathbf{x}\in R_-,\\ &\qquad \mathbf{w_i}\in\theta\}\end{aligned} \tag{2.10}$$

and shown in figure (2.3b).

"The vector $\mathbf{w}$ *we are trying to find is perpendicular to the hyperplane. The positions* $\mathbf{w}$ *can hold is our solution space, the hyperplane is merely the result of* $\mathbf{w}^t\mathbf{x}=-b$*".*

In the next two subsections illustrative examples will be given on how to find such a vector $\mathbf{w_i} \in \theta$.

Finding a solution

There are an enormous amount of procedures for finding solution vectors for the problems presented in this section [9] [18]. Most basic algorithms start by initializing a random solution, `rand(w)`, and try to improve the solution iteratively. By defining a criterion function J on $\mathbf{w}$ where $J(\mathbf{w}) \longmapsto \mathbb{R}$, we are left by optimizing a scalar function. In every step a new solution is computed which either improves $J(\mathbf{w})$ or leaves it unchanged. A simple and very intuitive criterion function is that of misclassification minimization. Typically we want to minimize $J(\mathbf{w})$ where J presents the number of elements in D', the set containing misclassified instances.

Algorithm 1: Fixed Increment Single-Sample Perceptron

```
Input: initialize $\mathbf{w}$, i $\leftarrow$ 0
repeat
    $k \leftarrow (k+1)$ mod $n$
    if $y_k$ misclassified then
        $\mathbf{w} \leftarrow \mathbf{w} + \eta\ \mathbf{x}_k y_k$
    end
until all samples correctly classified ;
return w
```

Fixed-Increment Single-Sample Perceptron

One of the most simple algorithms to find a solution vector is that of the Fixed-Increment Single-Sample Perceptron [9], besides its simplicity it is very intuitive in its approach. Assume we have a data set $D = \{(\mathbf{x_1}, y_1), .., (\mathbf{x_n}, y_n)\}$ where $\mathbf{x_i}$ describes the features and y_i its corresponding label. Now let $d \in D$ and test whether or not if our $\mathbf{w}$ correctly classifies d. Whenever d is misclassified we will shift $\mathbf{w}$ towards d. This procedure will be done until the set D' containing all misclassified instances is empty. Equation (2.11) shows how the final solution for $\mathbf{w}$ can be written as a sum of previous non-satisfactory solutions.

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta(y_i - f(\mathbf{x_i}))\mathbf{x_i} \tag{2.11}$$

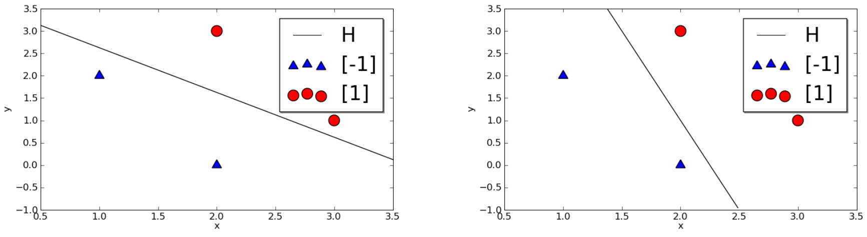
The perceptron function f maps an input vector $\mathbf{x}_i$ to a negative or positive sample $f(\mathbf{x}) \longmapsto \{-1, 1\}$ and is defined as:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^t\mathbf{x} + b > 0 \\ -1 & \text{else} \end{cases} \tag{2.12}$$

Equations (2.11) and (2.12) form the basis for the Fixed Increment Single-Sample Perceptron described in algorithm (1).

Example:

We are given the following: $D = \{(1,2,-1),(2,0,-1),(3,1,1),(2,3,1)\}$, where the last feature describes the corresponding class $\{1,-1\}$ (Appendix(C.1) shows a python implementation of the Fixed Increment Single-Sample Perceptron). The initial starting points were respectively initialized with $\mathbf{w} = (-2,-2)$, $\mathbf{w} = (0,0)$.



(a) Initial $\mathbf{w} = (-2,-2)$ (b) Initial $\mathbf{w} = (0,0)$

Figure 2.4: The Fixed Increment Single-Sample Perceptron algorithm with different starting conditions for $\mathbf{w}$. This algorithm will terminate when all patterns are located on the correct side of the plane. To the naked "eye" this separation is a poor generalization of the "real" location of the patterns.

As can be seen in figure (2.4) both hyperplanes separate the data using different solution vectors. The reason for this is that Fixed Increment Single-Sample Perceptron is not trying to minimize some objective function, but instead depends solely on constraints. This corresponds to the previous figure (2.3b) and equation (2.10). Other commonly used algorithms are: Balanced Winnow, Batch Variable Increment Perceptron, Newton Descent, Basic Descent[9].

Linear Programming

Linear programming (LP) is a mathematical way of determining the best solution for a given problem. The problems we have thus far encountered are of a linear nature, which can often be solved using *"out of the box"* optimization software. The LP technique of linear optimization describes a linear objective function, subjected to linear equalities and or linear inequalities. The following is a typical linear program (canonical form):

$$\textbf{Minimize} \quad \mathbf{c}^t\mathbf{w} \tag{2.13}$$
$$\textbf{Subject to} \quad \mathbf{A}\mathbf{w} \leq \mathbf{y} \tag{2.14}$$

where $\mathbf{c}^t\mathbf{w}$ describes the problems objective function and $\mathbf{A}\mathbf{w} \leq \mathbf{y}$ its corresponding constraints. This objective function often refers to cost or income which one wants to minimize or maximize. Most mathematical models are restricted to some sort of requirements, described as constraints to the problem. We have already seen how to solve a linear separating hyperplane using the Fixed Increment Single-Sample Perceptron, however understanding this linear program will help us understand the inner workings of our main goal, "the Support Vector Machine". In [6] more exact details can be found, here we give an short example.
Assume that in a binary classification task we have stored our instances in two matrices, K and L, where each matrix corresponds to a particular class. The rows indicate instances and the columns represent features. Now we want to separate the instances in K and L by computing a hyperplane such that the K instances are located in R_+ and the L instances in R_-. The hyperplane is once again defined by: $f(\mathbf{x}) = \mathbf{w}^t\mathbf{x}+b$. The fact that we "want" the instances to be located in different regions results in the inclusion of the constraints to our solution for $\mathbf{w}$ and b. These constraint then become:

$$\begin{aligned} &\texttt{Constraints} \\ &\qquad 0 > \mathbf{w}^t K + b, R_+ \\ &\qquad 0 < \mathbf{w}^t L + b, R_- \end{aligned} \tag{2.15}$$

As for an objective function we need to make sure that the search space is bounded and not infinite. Choosing $\texttt{minimize}||\mathbf{w}||_1{}^3 + b$ with an additional constraint that all elements in $\mathbf{w}$ and b are greater then 0 we have limited the search space. Combining the objective function with its corresponding constraints lead to the following LP:

$$\begin{aligned} &\texttt{Minimize} \\ &\qquad ||\mathbf{w}||_1 + b \\ &\texttt{Constraints} \\ &\qquad 0 > \mathbf{w}^t K + b \\ &\qquad 0 < \mathbf{w}^t L + b \\ &\qquad \mathbf{w}, b \succeq 0 \end{aligned} \tag{2.16}$$

where our hyperplane/discriminant function is defined by $f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b$

[3] the first norm is given by: $\sum_{i=1}^{n} |\mathbf{w}_i|$

Example Using Convex Optimization Software

There are a large variety of *"Solvers"* that can be used to solve optimization problems. The one that we will use for a small demonstration is CVXMOD[4] which is an interface to CVXOPT[5]. The previous linear program in equation (2.17) is not sufficient to handle real world problems. Since $\mathbf{w}$ can only take on positive numbers our hyperplane is restricted. In [6] they propose the following $\mathbf{w} = (\mathbf{u} - \mathbf{v})$ with $\mathbf{u}, \mathbf{v} \in (\mathbf{R}^{n+})$, making it possible for $\mathbf{w} \in \mathbf{R}^n$ while still bounding the search space. The latter leads to the following LP:

$$\begin{aligned}
&\texttt{Minimize} && && (2.17)\\
&&& ||\mathbf{u}||_1 + ||\mathbf{v}||_1 + b &&\\
&\texttt{Constraints} && &&\\
&&& \mathbf{u} \quad > \mathbf{w}^t K + b &&\\
&&& \mathbf{v} \quad < \mathbf{w}^t L + b &&\\
&&& \mathbf{u}, \mathbf{v}, b \quad \succeq 0 &&
\end{aligned}$$

The program we constructed for this example is listed in the appendix (C.2) and was performed on the following patterns:

$$K = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0.5 & 1 \\ 0 & 2 \end{bmatrix} \text{ and } L = \begin{bmatrix} 3 & 1 \\ 2 & 3 \\ 1 & 3 \\ 2.4 & 0.8 \end{bmatrix}$$

Figure (2.5a) shows that the solution is indeed feasible. However graphical illustration shows that some points are very close to the hyperplane. This introduces more risk of misclassifying an unseen pattern. This problem is solved by adding the constraint that there needs to be a minimum offset from a pattern to the hyperplane. For the K patterns the constraints become:

$$\mathbf{u} > \mathbf{w}^t K + b + margin \qquad (2.18)$$

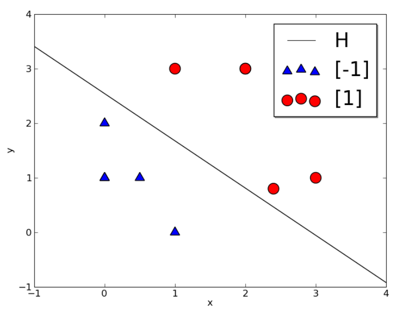
Where the margin describes the minimum offset to the hyperplane.

In figure (2.5b) we see that the distance between the hyperplane and the targets have increased. The "approximate" maximum margin can be found by iteratively increasing the margin until no solution can be found.

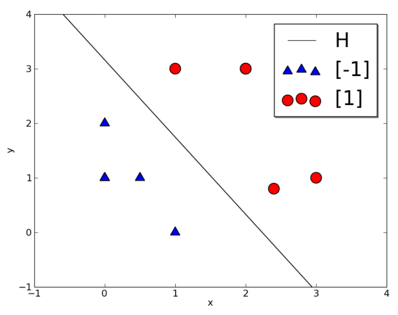
2.3.2 Case: 2 Linearly Inseparable

Applying the previous LP for finding an optimal hyperplane on a linear inseparable data set results in a far from optimal hyperplane, as seen in figure (2.6).

[4]CVXMOD Convex optimization software in Python
[5]CVXOPT Free software package for convex optimization based on the Python



(a) Using the LP from equation (2.18) on the patterns K and L gave the following solution. Even though all patterns are correctly classified, the hyperplane is unnecessary close to some patterns.



(b) Adding an additional margin to constraints of the LP in equation (2.18) results in better class separation. The Hyperplane is less close to the patterns than in figure (2.5a).

Figure 2.5: Using linear programs to solve linear separable data.

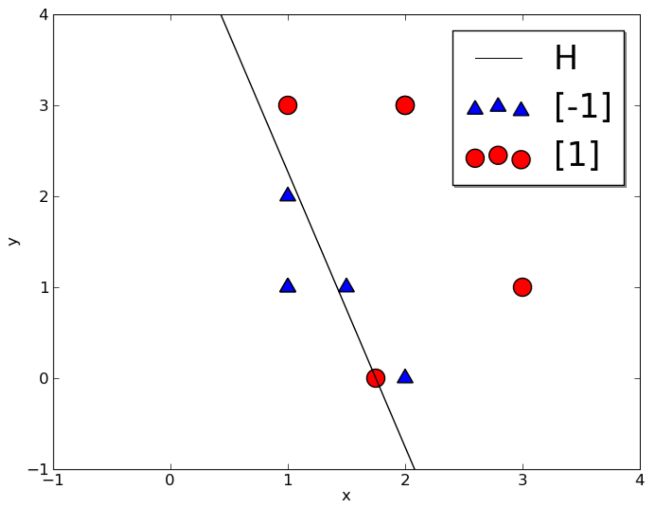
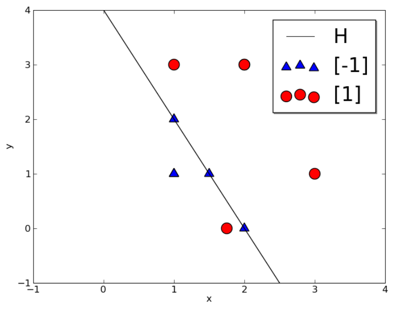
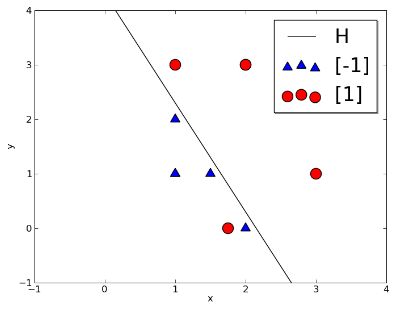


Figure 2.6: Using the linear program in equation (2.18) results in multiple misclassifications. Taking the first norm for **w** as an objective is not suitable for linear inseparable data sets.

In situations as seen in figure (2.6) the previous LP is insufficient. By defining a criterion function that minimizes the number of misclassified instances, `min J'(x)`, where `J'` is the set of misclassified instances by **x**, we will introduce another variable called 'slack'. This slack variable will be used to fill up the negative space that a misclassified instance has towards the optimal hyperplane.



(a) Minimizing the number of misclassified instances by introducing slack variables. Even though the hyperplane separates the data with only one error, its position is not optimal in terms of maximum margin.



(b) Minimizing the number of misclassified instances not only with slack variables but adding additional margins, thereby decreasing the risk on misclassification of unseen data.

Figure 2.7: The introduction of margin and slack improves the position of the hyperplane.

It is possible to define an interesting classifier by introducing slack variables and minimum margins to the hyperplane. If we were to introduce slack without a margin then the border would always be on top of those patterns which harbors the most noise (figure (2.7a)). Figure (2.7b) is graphical representation of a classifier where margin was added. This classifier is capable of finding solutions for linear inseparable data allowing misclassifications. Students familiar with the most famous support vector machine image (figure (3.1)), see some familiarities with figure (2.8). Figure (2.8) is the result of minimizing the numbers of misclassified instances and trying to create a largest possible margin. In the chapter on Support Vector Machines we will see how Vapnik defines a state of the art machine learning technology that translates the problem of "misclassification minimization" to "margin-maximization".

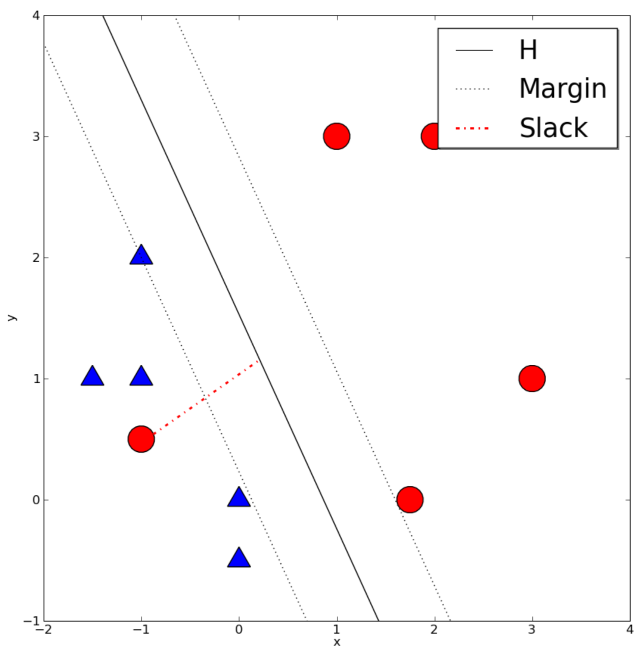


Figure 2.8: By introducing a slack variable the strain the hyperplane has on the data-points can be reduced, allowing the search for greater margins between the two classes. By introducing margin variables it is also possible to increase this margin.

Chapter 3

Support Vector Machines

3.1 Introduction

In the previous chapter on Linear Discriminant Functions we saw that it was possible to compute hyperplanes capable of perfectly separating linear separable data. For the linear inseparable case slack variables where used to achieve acceptable hypothesis spaces. *Misclassification minimization* and *Linear programs* where described as approaches for creating hypothesis spaces. The novel approach that Vapnik [25] introduced is that of *risk minimization*, or as we will come to see *margin maximization*. The idea behind is as follows: high accuracy on the train-data does not guarantee high performance on test-data, a problem which has been the nemesis of every machine learning practitioner. Wiping proposed that it is better to reduce the risk of making a misclassification instead of maximizing train-accuracy. This idea on risk-minimization is in a geometric setting, the equivalent to that of margin-maximization. In Section (2.3.2) we saw that by adding an additional margin to our optimization problem, we could more clearly separate the patterns. The problem there however was that all the patterns had influence on the location of the hyperplane and its margins. The Support Vector Machine counter attacks that problem by only selecting those data patterns closest to the margin.

The goal of this chapter is to get a deeper understanding on the inner workings of a Support Vector Machine. We do this by: (1) introducing the concept of margin; (2) explaining and deriving a constrained optimization problem; (3) implementing our own Support Vector Machine; and (4) Introducing the concept of soft margin optimization.

This chapter will not cover geometric properties already explained in Chapter (2).

3.2 Support Vector Machines

Just as in Chapter (2), the goal of a Support Vector Machine is to learn a discrimination function $g(\mathbf{x})$. The function $g(\mathbf{x})$ calculates on which side an unknown instance is located with respect to the decision-boundary. The main difference between Support Vector Machines and the examples we have seen in Chapter (2) is the way this boundary is obtained. Not only is the problem formulated as a quadratic problem instead of linear, the objective is formulated in terms of a margin. The SVMs objective is to maximize the margin, given by $\frac{2}{||\mathbf{w}||}$, between the support vectors and the decision-boundary. The idea of margin-maximisation is captured in figure (3.1). The resulting function $g(\mathbf{x})$ of an optimized Support Vector Machine is nothing more than a dot-product with an additional bias. Those who are familiar with Support Vector Machines might protest, and note the use of different non linear similarity measures. These however are not part of the Support Vector Machine, and should be regarded as being part of a Kernel Machine. The distinction between the two will become more apparent when we will make our own Support Vector Machine in section (B.1). The intuitive idea behind the Support Vector Machine is as follows: assume that we have a data set D containing two classes $\{+1, -1\}$. We saw in Chapter (2) that every $d \in D$ had its contribution to the final formulation of the discriminant function $g(\mathbf{x})$. The fact that every instance has its influence on the margin causes mathematical problems. If we are interested in defining a margin then it would be more natural to only include those instances closest to the boundary. The Support Vector Machine algorithm will add a multiplier α to every $d \in D$ and its value ($\mathbb{R}^+$) denotes the relevance of that instance for the boundary/hyperplane.

$d_x \in D$	label	α	features
d_1	1	0.00	$\mathbf{x_1}$
d_2	1	0.8	$\mathbf{x_2}$
d_3	-1	0.1	$\mathbf{x_3}$
d_4	-1	0.7	$\mathbf{x_4}$

Table 3.1: The $d_x \in D$ that make up the decision boundary have a non-negative α multiplier. The SVM paradigm returns the most optimal α values, corresponding to the largest margin.

The α multipliers are of course not randomly chosen. In fact those patterns that have non-zero *alpha*'s are positioned on-top of the decision boundary. These support vectors will have an absolute offset of 1 to the hyperplane. The latter is what gives the Support Vector Machine its name, all the non-zero *alpha*'s are supporting the position of the hyperplane, hence the name "Support Vectors".

The support vectors in figure (3.1) are marked with an additional circle. It

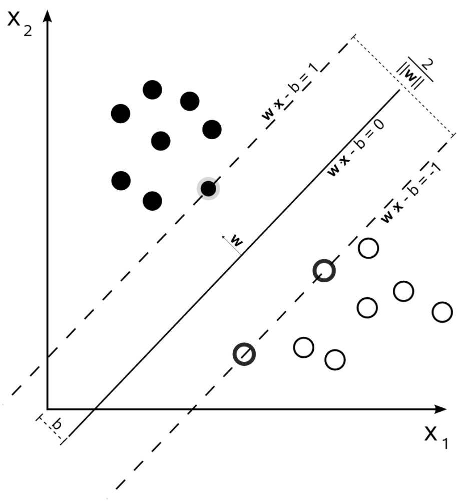


Figure 3.1: The non-zero α multipliers pin the hyperplane to its position. The task of the Support Vector Machine is to find a set of non-zero α's that make up the largest boundary ($\frac{2}{||\mathbf{w}||}$) possible between the two classes.

is clear in figure (3.1) that the hyperplanes position depends on those $d \in D$ that have non-negative α's.

3.2.1 Classification

The classification of an unknown instance $\mathbf{x}$ is calculated as a sum over all similarities $\mathbf{x}$ has with the support vectors ($\mathbf{x}_i$). This similarity measure is the dot-product between $\mathbf{x}$ and the support vectors. In equation (3.1) we see how the support vector coefficients (α) regulate the contribution of a particular support vector. The summed combination of similarity ($\langle \mathbf{x}_i \cdot \mathbf{x} \rangle$) and support vector strength ($\alpha_i y_i$) will determine the output for a single support vector.

$$g(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b \tag{3.1}$$

with l being the number of Support Vectors
and y_i the label of the corresponding Support Vector

The *Sign* of the output function in equation (3.1) determines to which class $\mathbf{x}$ is regarded. Equation (3.1) can only be used for binary classification. In order to solve multi-class problems [14] a multitude of SVMs need be used.

3.3 Hard Margin

From the first two sections we saw that the Support Vector Machine: (1) maximizes the margin which is given by $\frac{2}{||\mathbf{w}||}$; and (2) for that it needs to select a set of non-zero α values.
To increase the margin ($\frac{2}{||\mathbf{w}||}$) between the two classes, we should minimize $||\mathbf{w}||$. The objective here is to minimize $||\mathbf{w}||$. However $\mathbf{w}$ can not take on just any value in $\mathbb{R}^N$: it still has to be an element in our solution space. The solution space is restricted by the following:

$$\{\langle \mathbf{x_i} \cdot \mathbf{w} \rangle + b \geq 0 \quad | \quad y_i = 1\} \tag{3.2}$$
$$\{\langle \mathbf{x_i} \cdot \mathbf{w} \rangle + b \leq 0 \quad | \quad y_i = -1\} \tag{3.3}$$

Equation (3.2) and (3.3) are the constraints that ensure that positive patterns stay on the positive side of the hyperplane and vise-versa. Together they can be generalized in the following canonical form:

$$\{y_i(\langle \mathbf{x_i} \cdot w \rangle + b) \geq 1\} \tag{3.4}$$

The latter combined leads to the following quadratic programming problem:

$$\begin{aligned} \textbf{minimize} \quad & ||\mathbf{w}|| \\ \textbf{subject to} \quad & \{y_i(\langle \mathbf{x_i} \cdot \mathbf{w} \rangle + b) \geq 1\} \end{aligned} \tag{3.5}$$

This is known as the primal form.

As mentioned in the previous section the Support Vector Machine controls the location of the hyperplane through the α multipliers. We will rewrite the *Primal form* into its counter part, known as the *Dual form*, for the following reasons: (1) the data points will solemnly appear as dot-products. This will simplify the introduction of different feature mapping functions. And (2) the constraints are then moved from equation (3.4) to the Lagrangian multipliers (α's), making the problem more easy to handle.

Optimizing $||\mathbf{w}||$ includes the norm of $\mathbf{w}$, which involves a square root. Without changing the solution we can change this to $\frac{1}{2}||\mathbf{w}||^2$. The factor $\frac{1}{2}$ is there for mathematical convenience.

We start with the primal:

$$\begin{aligned} \textbf{minimize} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \textbf{subject to} \quad & \{y_i(\langle \mathbf{x_i} \cdot \mathbf{w} \rangle + b) \geq 1\} \end{aligned} \tag{3.6}$$

If we have the following quadratic optimization problem,

$$\begin{aligned} &\textbf{minimize} && x^2 \\ &\textbf{subject to} && x \geq b \end{aligned} \tag{3.7}$$

then this corresponds with the following Lagrangian formulation.

$$\begin{aligned} &min_x max_\alpha && x^2 - \alpha(x - b) \\ &\textbf{subject to} && \alpha \geq 0 \end{aligned} \tag{3.8}$$

The constraints in equation (3.7) have moved to the objective in equation (3.8). As a result the former constraints are now part of the objective and serve as a penalty whenever violated. Using this formulation allows us to use less strict constraints. Transforming the primal, equation (3.5), into a Lagrangian formulation leads to:

$$\begin{aligned} &min_{\mathbf{w}b} max_\alpha && \frac{1}{2}||\mathbf{w}||^2 - \sum_j \alpha_j \left[y_j(\langle \mathbf{x_j} \cdot \mathbf{w} \rangle + b) - 1\right] \\ &\textbf{subject to} && \alpha_j \geq 0 \end{aligned} \tag{3.9}$$

Equation (3.9) sees the first introduction of the α value's which eventually will correspond to support vectors. For convenient reasons we can rewrite equation ((3.9)) to:

$$\begin{aligned} &min_{\mathbf{w}b} max_\alpha && \frac{1}{2}||\mathbf{w}||^2 - \sum_j \alpha_j \left[y_j(\langle \mathbf{x_j} \cdot \mathbf{w} \rangle + b)\right] + \sum_j \alpha_j \\ &\textbf{subject to} && \alpha_j \geq 0 \end{aligned} \tag{3.10}$$

Wishing to minimize both $\mathbf{w}$ and b while maximizing α leaves us to determine the saddle points. The saddle points correspond to those values where the rate of change equals to zero. This is done by differentiating the Lagrangian-primal (Lp) equation (3.10), with respect to $\mathbf{w}$ and b and setting their derivatives to zero:

$$\frac{\partial Lp}{\mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} - \sum_j \alpha_j y_j \mathbf{x}_i = 0 \tag{3.11}$$

$$\mathbf{w} \quad = \quad \sum_j \alpha_j y_j \mathbf{x}_i \tag{3.12}$$

$$\frac{\partial Lp}{b} = 0 \quad \Rightarrow \quad -\sum_j \alpha_j y_j = 0 \tag{3.13}$$

$$\sum_j \alpha_j y_j \quad = \quad 0 \tag{3.14}$$

By inserting equations (3.12) and (3.14) into our Lp:

$$max_\alpha \quad -\frac{1}{2}\sum_j \alpha_j y_j \mathbf{x}_j \sum_j \alpha_j y_j \mathbf{x}_j + \sum_j \alpha_j \tag{3.15}$$

equals to :

$$max_\alpha \quad \sum_j \alpha_j - \frac{1}{2}\sum_{ij} \alpha_j y_j \mathbf{x}_j \alpha_i y_i \mathbf{x}_i \tag{3.16}$$

Equals to:

$$\begin{aligned} max_\alpha L_{Dual} &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\ \textbf{subject to} \quad & \alpha_j \geq 0 \\ & \sum_j \alpha_j y_j = 0 \end{aligned} \tag{3.17}$$

which is known as the Dual form.
The value of the dual form is negative in the implementation used in the experiments.
Equation (3.17) gives a quadratic optimization problem resulting in those α's that optimize the margin between two classes. This form for optimization is called "Hard Margin optimization", since there are no patterns located inside the margin. Notice that the last term in the dual form is simply the dot-product between two data points. Later on we will see that in this formulation we are able to replace $\langle \mathbf{x_i} \cdot \mathbf{x_j} \rangle$ with other similarity measures. One of the most widely used optimization algorithms used in Support Vector Machines is Sequential Minimal Optimization (SMO) [20]. SMO breaks the "whole" problem into smaller sets, after which it solves each set iteratively.

3.4 Soft Margin

In section (3.3) we saw the creation of a maximal or hard margin hypothesis space. Hard margin in the sense that no patterns are allowed in the margin space $\langle -1, 1 \rangle$. However "real-life" data sets are never without noisy patterns and often can not be perfectly separated. In 1995 Vapnik and Cortes introduced a modified version on Support Vector Machines: the soft margin SVM [4]. The soft margin version allows patterns to be positioned outside their own class space. When a pattern is located outside its target space it receives an error ϵ proportional to the distance towards its target space. Therefore the before discussed initial primal objective saw the extension of an error term (*Slack*):

$$\begin{array}{cc} \frac{1}{2}||\mathbf{w}||^2 \quad \Rightarrow & \frac{1}{2}||\mathbf{w}||^2 + C\sum_j \epsilon_j \\ \textbf{hard margin} & \textbf{soft margin} \end{array} \tag{3.18}$$

The penalty C is a meta-parameter that controls the magnitude of the contribution ϵ and is chosen beforehand. Allowing misclassified patterns in the objective function (equation (3.18)) is not enough, the constraints each $pattern_i$ has on the objective should also be loosened:

$$\{y_i(\langle \mathbf{x_i} \cdot w\rangle + b) \geq 1 - \epsilon_i\} \tag{3.19}$$

The error term ϵ is defined in distance towards the patterns target space and therefore is bounded, $\epsilon \geq 0$. This leads to:

$$\begin{array}{rl} \textbf{minimize} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_j \epsilon_j \\ \textbf{subject to} & \{y_i(\langle \mathbf{x_i} \cdot \mathbf{w}\rangle + b) \geq 1 - \epsilon_i | \epsilon_i \geq 0\} \end{array} \tag{3.20}$$

The primal version of equation (3.20) also has its dual counter part. The derivation is quite similar to that of the hard margin SVM in equation (3.5), extensive documentation and information can be found in almost all books on Support Vector Machines [5, 24]. For their work on the soft margin SVM Cortes and Vapnik received the 2008 Kanellakis Award[1].

3.4.1 C-SVC

The Cost Support-Vector-Classifier (C-SVC) is an algorithm used in soft margin optimization. Soft margin optimization received lots of attention in the machine learning community since it was capable of dealing with noisy data. The most commonly used soft margin optimization algorithms are C-SVC and ν-SVC [24, 5]. The quadratic optimization problem using C-SVC effects the constraints on the support vectors, and has the following form:

$$\begin{array}{rcl} max_\alpha L_{Dual} & = & \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j\rangle \\ \textbf{subject to} & & 0 \leq \alpha_j \leq C \\ & & \sum_j \alpha_j y_j = 0 \end{array} \tag{3.21}$$

[1]The Paris Kanellakis Theory and Practice Award is granted yearly by the Association for Computing Machinery (ACM) to honor specific theoretical accomplishments that have had a significant and demonstrable effect on the practice of computing.

By giving the support vectors an upper-bound of C the optimization algorithm is no longer able to stress the importance of a single instance. In hard margin optimization instances located near instances of the opposite class often would obtain extreme α values in order to meet its constraints.

3.4.2 C-SVC Examples

In this section we will compare the hard margin and soft margin classifiers. The concept of soft margin is easiest explained with examples. Figure (3.2) shows the result of a hard margin optimization of a simple data set containing one noisy pattern where an RBF kernel is used.

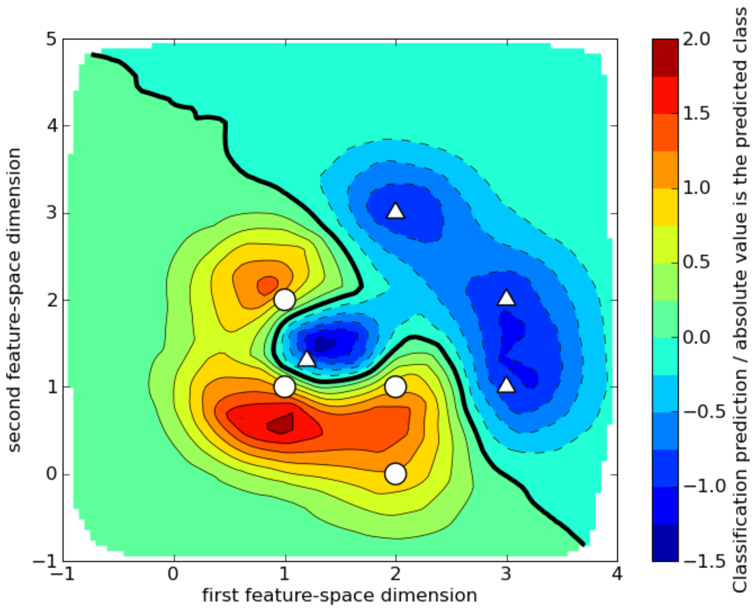


Figure 3.2: In the middle of this figure one sees an awkward shaped decision-boundary, which is caused by the most left triangle instance. It is reasonable to assume that this particular instance is a "noisy" pattern and should be considered so. For the hard margin optimizer it is impossible to create a more homogeneous space (without the lump in the middle) since all the support vectors must have a distance of 1 to the decision boundary.

In section (3.4.1) the idea behind soft margins, and how these algorithms can suppress the influence of noisy patterns was explained. Applying the C-SVC algorithm on the simple data set from figure (3.2) gives us the following hypothesis space, figure (3.3).

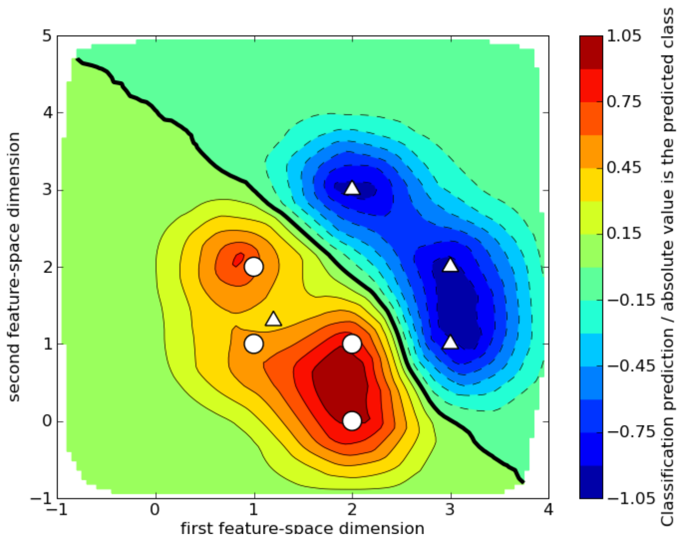
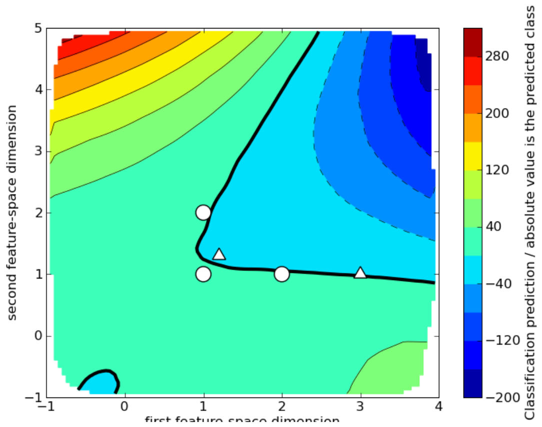


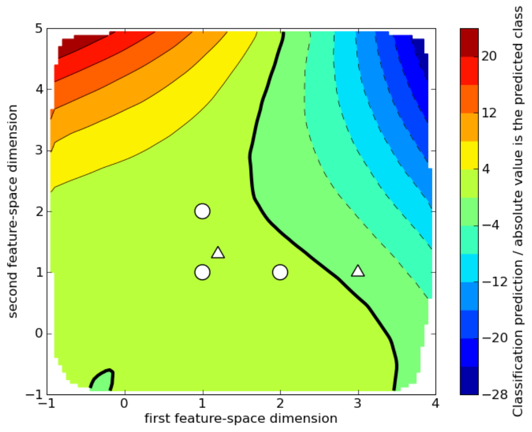
Figure 3.3: The hypothesis space generated using a soft margin gives a more homogeneous space. If the most left triangle is regarded as a noisy pattern then this hypothesis space is a great improvement over that in figure (3.2) and is almost identical to that of figure (B.1d). The *Cost* parameter here is set to 1.

Through soft margin optimization, see figure (3.3), we obtain a completely different hypothesis space. The obtained decision boundary is much smoother and gives a better description of the "actual" class boundary.

The concept of soft margin optimization does not only work on radial-basis kernels, it can be applied to all kernels. In the following example we have replaced the *RBF* kernel with a third degree *polynomial* (figure (3.4)).



(a) The most left triangular instance is causing the stress on the decision-boundary in hard margin optimization giving it an unnatural contour.



(b) Applying the soft margin optimization using C-SVC it is possible to relieve the stress the most left triangular instance has on the decision-boundary.

Figure 3.4: The decision boundary takes on smoother contours in soft margin optimization.

Chapter 4

Kernel Functions

4.1 Introduction

One might have noticed the simplicity of the problems presented in the previous two chapters. The focus of the latter Chapters was on the theories behind SVMs and LDFs, which are best presented using intuitive examples rather than real-world examples. In this Chapter however we will show the limited computational performance of a linear learning machine, and how "kernels" can help to overcome this. The hypothesis space in real-world problems can often not be simply described by a linear combination of attributes, but more complex relationships between the attributes needs to be found. The latter laid the foundation of multi-layers of linear thresholded functions, resulting in *Neural Networks* [11]. For linear machines the usage of "Kernels" offers an alternative solution for getting a more complex hypothesis space. As will become clear in the next sections, a kernel or combination of kernels can project data into a higher dimensional feature space, thereby increasing the computational power of the linear machine. The majority of these kernels assume that each dimension in feature space is equally important, there is however no guarantee that this holds true. Therefore we will introduce a feature space capable of distinguishing the importance between the different features. For SVMs using a kernel merely results in replacing the dot product in the dual-form by a chosen kernel. The usage of "Kernels" has a number of benefits: (1) the input space does not need to be pre-processed into our "new" space; (2) it is computationally inexpensive to compute; (3) the number of features does not increase the number of parameters to tune; and (4) kernels are not limited only to SVMs.

The structure of this chapter is as follows: (1) we give a toy-example of how a complicated problem can more easily be solved in a "new" space; (2) we give a more formal definition of kernels and feature mapping; (3) we explain the notion of a "valid kernel"; and (4) introduce a new type of kernel.

4.2 Toy Example

Intuitively a linear learning machine should have problems with quadratic functions describing our target function. Therefore we will show that for a linear learning machine it is impossible to learn the correct target functions without transforming the input space.

Consider the following target functions:

$$f(x, y) = x^2 + y^2 \leq 1 \tag{4.1}$$
$$g(x, y) = x^2 + y^2 \geq 3 \tag{4.2}$$

At first glance (figure (4.1)) the latter quadratic equations $\{f, g\}$ seem easily

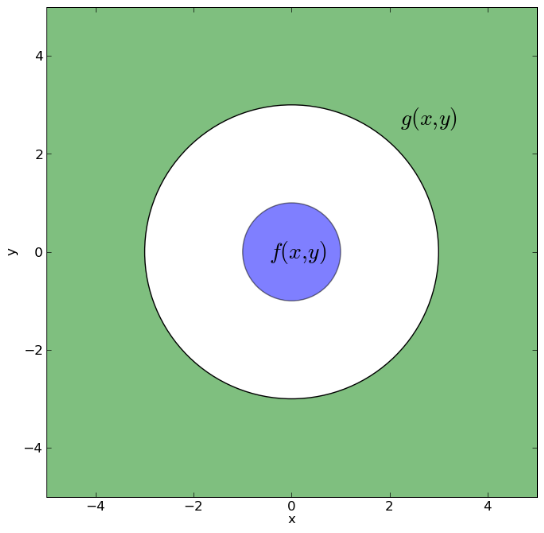


Figure 4.1: It is a trivial task "For the naked-eye" to separate the two functions f and g. A linear classification algorithm has virtually no means to correctly distinguishing the two.

distinguishable in a homogeneously spread space. Although it looks effortless for humans, this is impossible for linear learning machines. As can be seen in figure (4.1) there is not a linear combination of x and y that can separate the two functions/labels. Figure (4.2) shows the result of adding one extra feature/dimension, namely the distance to the origin:

$$(x, y) \longmapsto (x, y, z) \tag{4.3}$$

with z being $\sqrt{x^2 + y^2}$

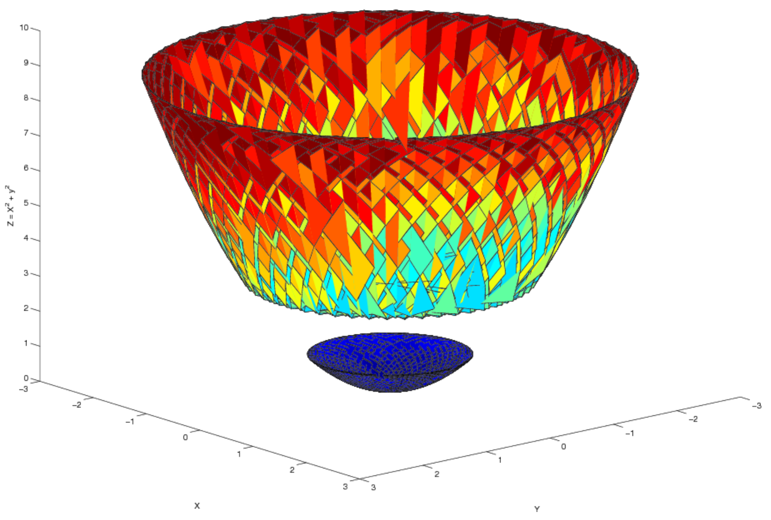


Figure 4.2: In some certain cases it can be easier to solve a problem when it is projected into a different space. In this situation adding an additional feature $z = \sqrt{x^2 + y^2}$ to figure (4.1) makes linear separation possible as can be seen from the gap in the vertical axis.

In figure (4.2) one can see that it is possible for an SVM or other linear machines to separate the two functions. In this toy-example we found a simple relationship between x and y, namely their combined length, which made it possible to linearly separate $\{f, g\}$. In practice this is not always as easy as it might look, since we often do not know which space projection is best for our data set. Therefore the choice of kernel is still often a heuristic.

4.3 Kernels and Feature Mappings

The example in the section (4.2) gave a nice introduction to how a relatively “simple” mapping can lead to a much better separation between different classes. In this section we will take a more formal look at the definition of a kernel.

As explained earlier, in order to correctly classify “real-world” data we need to be able to create a more complex hypothesis space. The function in equation (4.4) shows how an input space with size N can be transformed to a different space.

$$g(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(x) + b \tag{4.4}$$

where $\phi : X \to F$ can be a non linear mapping.

Definition 1. *We call ϕ a feature mapping function from $X \to F$ with X being the input space and F the feature space, such that $F = \{\phi(x) : x \in X\}$*

As we have seen in the previous chapter the activation of an SVM given a test sample is a follows:

$$g(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b \tag{4.5}$$

with l being the number of support vectors.
In order to calculate the similarity between a test and a train sample in a different "space" we replace the dot product -$\langle \mathbf{x}_i \cdot \mathbf{x} \rangle$ in equation (4.5)- with its counter part in feature space:

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b \tag{4.6}$$

with ϕ being a mapping function defined by definition (1).
The method of directly computing the inner-product is called *kernel* function.

Definition 2. *A kernel function K has the form:*

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \rangle \tag{4.7}$$

where $\mathbf{x}$,$\mathbf{y} \in X$
with ϕ being a mapping function defined by definition (1).

In literature the activation values described in equations (4.5) and (4.6) are often formulated as:

$$g(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{4.8}$$

with K being a kernel function.

Equation (4.8) presents the role of a kernel in a Support Vector Machine. In this form it is possible to interchange different types of kernels K.

4.4 Kernel Types

As we saw in chapter (3) the original hyperplane algorithm was a linear classifier. In the 90's Vapnik and others suggested a way to create non-linear classifiers through the introduction of the "kernel trick". This "trick" merely suggested the replacement of the $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ in equation (3.17) with different types

Polynomial (homogeneous)	$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$
Polynomial (inhomogeneous)	$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$
Radial Basis Function	$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$
Gaussian Radial Basis Function	$k(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}\right)$
Hyperbolic Tangent	$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)$

Table 4.1: Different types a non-linear kernels that are often used in Support Vector Machines.

of kernels. Only the similarity space k changes while the maximum-margin hyperplane algorithm is kept intact. This results in a maximum-margin hyperplane even though the input-space does not necessarily need to be linear. Table (4.1) shows the most familiar kernels used in machine learning.

4.5 Valid (Mercer) Kernels

As previously mentioned, a kernel creates a more complex hypothesis space. And there are two approaches for computing such spaces: (1) design a certain inner product space after which one computes the corresponding kernel; and (2) design a kernel on intuition and test its performance. The latter option is of interest to this section; in particular: “what properties of a kernel function are necessary to ensure that there indeed exists a mapping for such a feature space”. In order to test this we need to test if the kernel is “valid”. Definition (3) describes what we mean when a kernel is referred to as a valid or Mercer's kernel.

Definition 3. *We say that a kernel K is valid when:*

$$\exists \phi \text{ such that } K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \tag{4.9}$$

where ϕ is a map function by definition (1).

Note: we must make sure that $K(\mathbf{x}, \mathbf{y})$ describes some feature space!
One way of looking at this is as follows:
The angle or projection between two vectors tells us how similar the two vectors are, at least in “dot-space”. In order to keep that similarity measure we need to make sure that every space we transform our data towards must be fabricated out of “dot-spaces”.

There are three conditions that need to be guaranteed for definition (3) to hold.

Condition (3.1): Symmetry

The symmetric case is more or less trivial since it follows from the definition of the inner product [5] on page (32).

$$K(x, y) = \langle x \cdot y \rangle = \langle y \cdot x \rangle = K(y, x) \tag{4.10}$$

Also it makes sense that the similarity between two instances is not depending on the order.

Condition (3.2): Cauchy-Schwarz Inequality

$$\begin{aligned} K(\mathbf{x}, \mathbf{y})^2 &= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \rangle^2 \leq ||\phi(\mathbf{x})^2|| \; ||\phi(\mathbf{y})||^2 && (4.11) \\ &= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{x}) \rangle \langle \phi(\mathbf{y}) \cdot \phi(\mathbf{y}) \rangle = K(\mathbf{x}, \mathbf{x}) K(\mathbf{y}, \mathbf{y}) && (4.12) \end{aligned}$$

The Cauchy-Schwarz inequality is best understood if we again see the inner product as a similarity measure, equation (4.13).

$$\langle \mathbf{x} \cdot \mathbf{y} \rangle = ||\mathbf{x}|| \; ||\mathbf{y}|| \cos \gamma \tag{4.13}$$

When two vectors have maximum similarity, their angle $\gamma = 0\,^\circ$. It is only in this case of parallel vectors that $\langle \mathbf{x} \cdot \mathbf{y} \rangle = ||\mathbf{x}|| ||\mathbf{y}||$. In all other cases, if $\gamma > 0\,^\circ$ and $\gamma < 180\,^\circ$ then, $|\cos \gamma| < 1$. The difference between equations (4.12) and (4.14), is that of the function ϕ. We know however that: (1) for every valid kernel-function there exists an inner product space; and (2) that the Cauchy-Schwarz inequality holds for all inner products. Hence equation (4.12) holds.
From the fact that $|\cos \gamma| \leq 1$, it is easy to see that equation (4.14) holds.

$$\langle \mathbf{x} \cdot \mathbf{y} \rangle \leq ||\mathbf{x}|| \; ||\mathbf{y}|| \text{ Cauchy-Schwarz inequality} \tag{4.14}$$

The first two properties need to hold and are considered general properties of inner products but they are not sufficient for a kernel being a "Mercer" kernel. The last condition comes from a theorem by Mercer; hence the name Mercer kernel.

Condition (3.2): A Semidefinite Kernel

The following proposition needs to hold for a symmetric function on a finite input space to be a kernel function [5]:

Proposition 1. *Given a symmetric function $K(\mathbf{x}, \mathbf{y})$, and a finite input space* $\mathbf{v}$ *such that* $\mathbf{v} = \{v | v \in \mathbb{R}^n, n < \infty\}$, *, then $K(\mathbf{x}, \mathbf{y})$ is a kernel function if and only if*

$$\mathbf{K} = (K(\mathbf{v}_i, \mathbf{v}_j))_{i,j=1}^{N} \tag{4.15}$$

is positive semi-definite.

Definition 4. *A matrix M is called positive-semidefinite if $\forall v$ such that $v \in \mathbb{R}^n$ and $n < \infty$.*

$$v'Mv \geq 0 \tag{4.16}$$

In order to show that our kernel matrix is positive-semidefinite we need to show that: $\mathbf{z}'\mathbf{K}\mathbf{z} \geq 0$. *"Note: given that our function is symmetric!"*

$$\begin{aligned}
\mathbf{z}'\mathbf{K}\mathbf{z} &= \sum_i^n \sum_j^n z_i \mathbf{K}_{ij} z_j && (4.17)\\
&= \sum_i^n \sum_j^n z_i \phi(x_i)\phi(x_j) z_j && (4.18)\\
&= \sum_i^n \sum_j^n z_i \left[\sum_k^n \phi(x_i)_k \phi(x_j)_k\right] z_j && (4.19)\\
&= \sum_i^n \sum_j^n \sum_k^n z_i \phi(x_i)_k \phi(x_j)_k z_j && (4.20)\\
&= \sum_k^n \left[\sum_j^n z_j \phi(x_j)_k\right]^2 && (4.21)
\end{aligned}$$

We can see that equation (4.21) is in fact a sum of squares, which by definition is ≥ 0. Therefore $\mathbf{z}'\mathbf{K}\mathbf{z} \geq 0$. The latter conditions and propositions give body to the well known Mercer theorem:

Theorem 1. *(Mercer)*
Given a kernel function, $K(\mathbf{x}, \mathbf{y})$ then it said to be valid by definition (3), if and only if $v \in \mathbb{R}^n$ and $n < \infty$. The kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is symmetric and positive-semidefinite.
where $\mathbf{K} = (K(\mathbf{x}_i, \mathbf{y}_j))_{i,j=1}^{N}$

4.6 Creating a Kernel

In this section we will demonstrate the validity of the kernel that is proposed in this work to improve the classification performance of an SVM. As mentioned during the introduction not every feature is equally important in describing our target class. Therefore we are interested in a feature space which not only maps data to higher dimensions, but in addition has the ability to give different importance to these features.

The following two definitions are directly taken from [5] on page (42-43), and for a deeper understanding and proofs we refer to their book [5] Chapter (3). The basic idea is to decompose a complex kernel into less complex kernels, thereby showing their validity.

Definition 5. *Let K_1 and K_2 be kernels over $X \times X$,$X \subseteq \mathbb{R}^n, a \in \mathbb{R}^+, f(\cdot)$ a real-valued function on X. Furthermore we use a feature mapping function ϕ, given by:*

$$\phi : X \to \mathbb{R}^m \tag{4.22}$$

with K_3 a kernel over $\mathbb{R}^m \times \mathbb{R}^m$, and $\boldsymbol{B}$ a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels:

1. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$
2. $K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z})$
3. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z})K_2(\mathbf{x}, \mathbf{z})$
4. $K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$
5. $K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$
6. $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\boldsymbol{B}\mathbf{z}$

Definition 6. *Let K_1 be a kernel over $X \times X$, $\mathbf{x}, \mathbf{z} \in X$, and $p(x)$ a polynomial with positive coefficients. Then the following functions are also kernels:*

1. $K(\mathbf{x}, \mathbf{z}) = p(K_1(\mathbf{x}, \mathbf{z}))$
2. $K(\mathbf{x}, \mathbf{z}) = exp(K(\mathbf{x}, \mathbf{z}))$
3. $K(\mathbf{x}, \mathbf{z}) = exp(-||\mathbf{x} - \mathbf{z}||^2/\sigma^2)$

4.6.1 Weighted-Radial-Basis Function

The kernel which we will decompose,to show its validity, is the weighted-radial-basis function (WRBF), with constant w and σ:

$$K_{wrbf}(\mathbf{x}, \mathbf{z}) = exp\left(-\frac{1}{\sigma}\sum_{i=1}^{n} w_i(x_i - y_i)^2\right) \tag{4.23}$$

As mentioned w and σ are constants, and therefore not dependent on either $\mathbf{x}$ *or* $\mathbf{y}$. The w_i in equation (4.23) is the variable that controls the influence of the ith dimension. If we were to set w_i to zero for a certain dimension than the contribution that dimension has on all instances would be equal, and thus in practice neglected.
First we will decompose K, equation (4.23), into smaller kernels.

$$\begin{aligned}
K(\mathbf{x}, \mathbf{y}) &= exp\left(-\frac{1}{\sigma}\sum_{1}^{n} w_i(x_i - y_i)^2\right) && (4.24)\\
&= exp\left(-\frac{1}{\sigma}\sum_{1}^{n} w_i\left[x_i^2 + y_i^2 - 2xy\right]\right) && (4.25)\\
&= exp\left(-\frac{1}{\sigma}\sum_{1}^{n} w_i x_i^2\right) exp\left(-\frac{1}{\sigma}\sum_{1}^{n} w_i y_i^2\right)\\
&\quad exp\left(\frac{1}{\sigma}\sum_{1}^{n} w_i 2x_i y_i\right) && (4.26)
\end{aligned}$$

The step above is approved by step 3 in definition (5) which tells us that multiplying two valid kernels results in a new valid kernel. This leaves us showing that less complex kernels need to be kernels.

In the second step we will show that the first two kernels in equation (4.26) are kernels. Since kernels $exp\left(-\frac{1}{\sigma}\sum_1^n w_i x_i^2\right)$ and $exp\left(-\frac{1}{\sigma}\sum_1^n w_i y_i^2\right)$ only either depend on $\mathbf{x}$ or $\mathbf{y}$, then by definition step 4 in (5) we know that together they form a kernel if $f(\cdot)$ is a real-valued function on X. Given of course is that our input space X is real-valued, then every real-valued scalar multiplication of our input space should be real-valued. This holds true if the following constraints are met:

- σ is a real-valued scalar
- w is a real-valued vector

Having shown that from the first two functions we can create a valid kernel, this leaves us with the last part of equation (4.26), namely: $exp\left(\frac{1}{\sigma}\sum_1^n w_i 2x_i y_i\right)$. Using equation (4.28) we show that the we can rewrite the last kernel in the following form:

$$exp\left(a(\mathbf{x}'\mathbf{W}\mathbf{y}))\right) \tag{4.27}$$

with $a = \frac{2}{\sigma}$.
We do this by transforming our weight vector w to a matrix $\mathbf{W}$,

$$\sum_{1}^{n} w_i x_i y_i \quad = \quad \mathbf{x}'\mathbf{W}\mathbf{y} \tag{4.28}$$

were $\mathbf{W}$ is diag($\mathbf{w}$).

$$diag(\mathbf{w}) = \mathbf{W}_{i,j} = \begin{cases} \text{if } i = j & w_i \\ \text{else} & 0 \end{cases} \tag{4.29}$$

Equation (4.27) is a kernel by definition (6.2) if and only if $a(\mathbf{x}'\mathbf{W}\mathbf{y}))$ is a kernel. Now by combining step 2 and 6 from definition (5) we see that this is indeed the case. There are however two details we must not forget: $\mathbf{W}$ must be a symmetric positive semi-definite matrix, and $a \in \mathbb{R}^+$.

Definition 7. *A symmetric positive semi-definite matrix has the following properties*

1. *symmetric*
2. *non-negative eigenvalues*

If we assume that the values of our initial w are all positive then it is easy to see that (1) it is symmetric since all values $W_{i \neq j}$, are zero; and (2) the eigenvalues of a diagonal matrix are the diagonals themselves, assuming that $W_{i,i} \geq 0$ then so must be its eigenvalues.

Note: every dimension in feature space will have its own w_i, by ensuring that a dimension is not allowed to receive a negative w_i we stay true to the kernel definition!
We have shown that K_{wrbf}, equation (4.23) is Mercer kernel if the following constraints are met:

1. $w \in \mathbb{R}^{+n}$
2. $\sigma \in \mathbb{R}^+$

Conclusion

Simply mapping data into higher dimensional feature space is a naive approach, since we know that not every feature has equal contribution to the target class. In section (4.6) we have shown that the WRBF is a valid Mercer kernel capable of overcoming these problems. Using a WRBF kernel allows us to create a feature space in which it is more easy to solve a particular problem, this since it gives us ability to stress or mask certain features. The second part of this work focuses on choosing different configurations in feature space weighing, and how this leads to classification improvement.

Part II

Methods

Chapter 5

Error Minimization and Margin Maximization

5.1 Introduction

Section (4.6.1) introduced the Weighted-Radial-Basis function (WRBF). We intend to use this kernel in a standard SVM/Kernel-Machine paradigm. With this combination -of an SVM with WRBF- we aim at improving the SVM's predictive capabilities, to that over an SVM with RBF. Equation (5.1) shows the difference between the two kernels. The weights coefficient $\mathbf{w}_i$ for the features is what distinguishes the WRBF from the RBF. By choosing different weight configurations we are able to change the SVM's dual-objective. We intend to show that configurations that have a positive effect on the objective also should have a positive effect on the accuracy. We propose two different approaches for finding the most optimal weight configurations: *error-minimization* and *margin-maximization*. Where error-maximization is trying to correct for misclassified instances, it is margin-maximization that stays more true to the philosophy behind SVMs [25], obtaining the largest margin possible between the different classes. This Chapter provides the mathematical derivations for both approaches, and discusses its implementation.

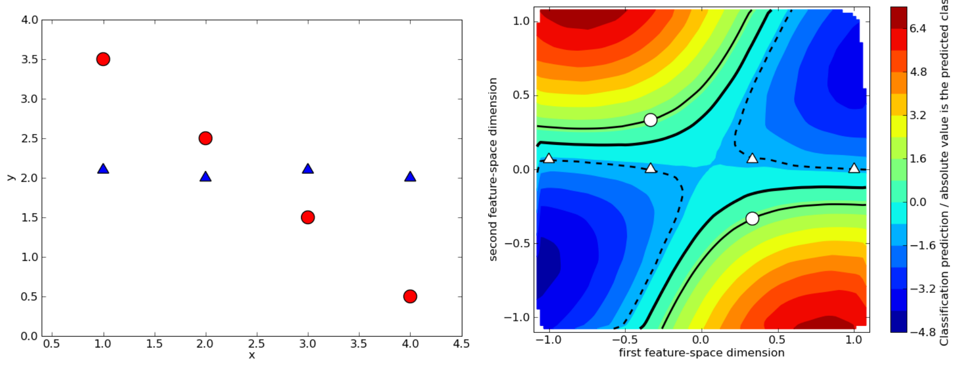
$$\begin{aligned} exp\left(-\frac{1}{\sigma}\sum_{1}^{n}(\mathbf{x}_i-\mathbf{y}_i)^2\right) &\rightarrow exp\left(-\frac{1}{\sigma}\sum_{1}^{n}\mathbf{w}_i(\mathbf{x}_i-\mathbf{y}_i)^2\right) \\ rbf(\mathbf{x},\mathbf{y}) &\rightarrow wrbf(\mathbf{x},\mathbf{y}) \end{aligned} \tag{5.1}$$

We adopt the following structure: (1) Introduction to the concept of feature-space weighing using a WRBF kernel; and (2) Finding weight configurations through error-minimization and margin-maximization.

5.2 Improving the Dual-Objective

Vapnik showed that by optimizing the dual-objective function the Support-Vector-Machine is able to obtain the largest margin possible. From this we can deduce that: "Assume we are given a set of train-data, then any model M_n is non-maximal if there exists another M_{n+1} for which the SVM achieves a higher dual-objective". This last statement does not state that the test-accuracy will be improved, it is rather a measurement of optimization. We believe that if we can find a weight configurations that increases the dual objective that this corresponds to a larger margin.

The following example will show that by changing **w** in the WRBF kernel we are able to improve the margin between the two boundaries which make up the eventual classification. The following plots/hypothesis have been created using our own software program, PySVM (see section (B.1)).

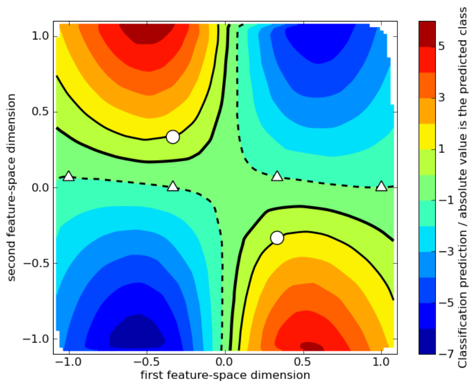


(a) Data set used.

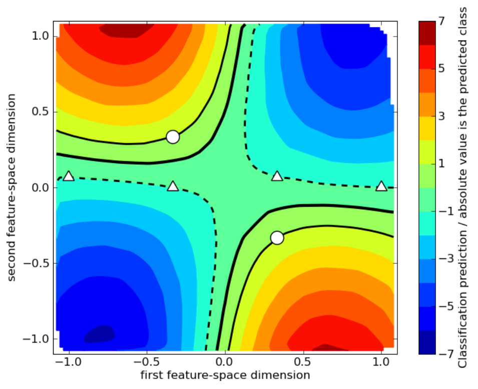
(b) Dual-Obj: -116.08 $w_x = 1, w_y = 1$

Figure 5.1: The red-circle data points cross the blue triangles in an angle not equal to 90 °. By doing so we have ensured that the two-feature dimensions are unequally important. The x-axis contains less information on the labels compared to the y-axis.

Figure (5.1b) shows the hypothesis space generated using the WRBF with $\mathbf{w} = (1, 1)$. Choosing ones for all weights resembles the behavior of the RBF kernel. Starting with these initial values is a good idea since it provides the RBF's hypothesis space. The hypothesis space shows diagonal activation for the triangle class. However these labels are projected almost uniformly over the x-axis, and there exist no triangles in either the upper-right or lower-left corner. By altering **w** we are able to rotate the hypothesis space. Figures (5.2a,5.2b) show the different hypothesis spaces where the importance of the x-axis (least informative) was emphasized.



(a) Dual-Obj: -158.98 $w_x = 2, w_y = .5$



(b) Dual-Obj: -204.93 $w_x = 4, w_y = .25$

Figure 5.2: Both hypothesis spaces show the result of rotating the hyperplane towards the x-axis. This rotation causes extra strain on the system resulting in a lower-optimum (Dual-Objective) compared to the initial weights. This results in 4 separate regions which were not at all apparent in figure (5.1a)

The strain caused by rotating the hyperplane towards the wrong axis can be seen in the objective value in table (5.1). Table (5.1) shows how the objective function changes with different rotations of **w**. This strain is caused by the stress the high activation of the support-vectors have on the whole system. By comparing the support vectors of the three different hypothesis spaces in table (5.2) we clearly see the effect incorrect rotation has on the support vectors. Even though some SVs obtain a slightly lower coefficient, the increase in the others is substantial larger.

w		dual-objective
$w_x = 1$	$w_y = 1$	-116.08
$w_x = 2$	$w_y = .5$	-158.98
$w_x = 4$	$w_y = .25$	-204.83

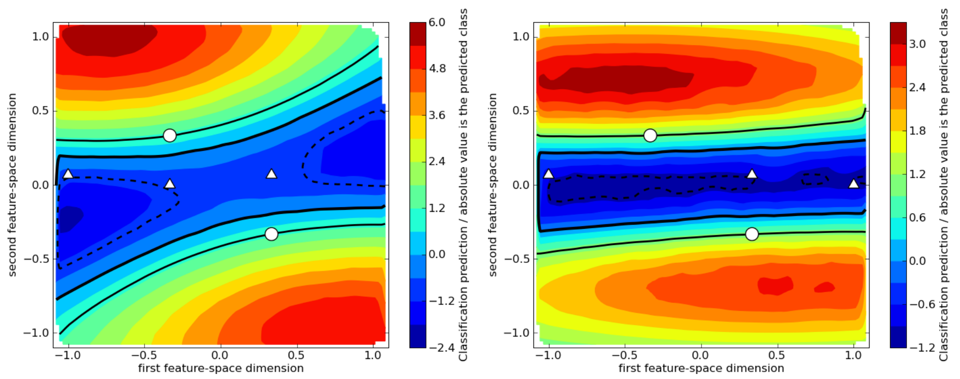
Table 5.1: Emphasizing the wrong features leads to more costs / lower objective since the strain on the SVs increases. This stress is illustrated in table (5.2) . (note that the obtained dual-objective values are given by libsvm).

In the first two altered hypothesis spaces figures (5.2a,5.2b), the x-axis/first-dimension carried more weight than the second. As stated earlier it is the

label	Support-Vector		$\alpha_{w=(1,1)}$	$\alpha_{w=(2,.\frac{1}{2})}$	$\alpha_{w=(4,.\frac{1}{4})}$
△	-1.	0.066	2.71	1.40	1.01
△	-0.33	0	59.96	86.48	114.56
△	0.33	0.0067	49.93	68.97	88.267
△	1.	0	1.53	1.21	0.97
○	-0.33	0.33	65.0	88.74	115.61
○	0.33	-0.33	53.02	71.2	89.43

Table 5.2: illustrates the stress gained on the support vectors when the wrong features carry increasingly more weight. This can be seen through the increasement on the Lagrange multipliers which in some cases have gained twice their value.

y-axis that carries the most information on the locations of our classes, and of course we intend to improve the SVM's hypothesis space. Rotating the hyperplane in the other direction shows the WRBF's capability of improving the SVM's objective functions.



(a) Dual-Obj: -63.66 $w_x = .5, w_y = 2$ (b) Dual-Obj: -25.72 $w_x = .25, w_y = 4$

Figure 5.3: Hypothesis spaces using a RBF kernel with different **w**. The dual-objective function gets more and more optimized when the weights converge to the y-axis.

Both figures (5.3a,5.3b) illustrate the effect **w** has on the hyperplane when it is rotated towards the y-axis. It can be seen that not only have we found a higher optimum, but we have lost one SV in the process. We expect that a feature-space in which it is easier to create separating hyperplanes requires less support vectors, and lower SV coefficients.

label	Support-Vector		$\alpha_{w=(1,1)}$	$\alpha_{w=(\frac{1}{2},2)}$	$\alpha_{w=(\frac{1}{4},2)}$
△	-1.	0.066	2.71	7.28	8.07
△	-0.33	0	59.96	22.46	0
△	0.33	0.0067	49.93	31.73	15.97
△	1.	0	1.53	0	0.002
◯	-0.33	0.33	65.0	37.16	16.50
◯	0.33	-0.33	53.02	28.69	10.8776

Table 5.3: illustrates the effect on the SVs when the most informative features carry the most weight. Most SV coefficients have drastically been reduced or in some cases even removed.

w		dual-objective
$w_x = 1$	$w_y = 1$	-116.08
$w_x = .5$	$w_y = 2$	-63.66
$w_x = .25$	$w_y = 4$	-25.71

Table 5.4: illustrates the effect on the dual objective when the most informative features carry increasingly more weight. The dual objective tends to converge to zero, this can be viewed as cost reduction because of the dual objective's negative sign.

5.2.1 Influences of Hyperplane Rotation

From the previous examples and figures we saw that it is possible to improve Vapnik's optimization formulation by rotating the hyperplane. Not only did the dual-objective increase, but two other properties became apparent: (firstly) It was shown that the need for high support-vector coefficients decreases when a more suitable space has been obtained. High α's are needed at points where there exists tension between the two classes. By releasing this tension we improve the objective and decrease the need for high multipliers. (secondly) The mean value of the SV multipliers are seen to decreases. In a more homogeneous spread space the need for specific activation becomes less relevant thus evening out the differences between the multipliers.

$\mathbf{w}$		dual-objective	α_μ	α_σ
$w_x = 4$	$w_y = .25$	-204.83	68.312	48.79
$w_x = 2$	$w_y = .5$	-158.98	52.99	37.25
$w_x = 1$	$w_y = 1$	-116.08	38.69	26.306
$w_x = .5$	$w_y = 2$	-63.66	25.46	10.25
$w_x = .25$	$w_y = 4$	-25.71	10.28	6.03

Table 5.5: Emphasizing the correct features leads to dual objective improvement.

The next part of this Chapter focuses on the mathematical derivations we need in order to find the perfect rotation for a given problem.

5.3 Error Minimization

The key idea behind error minimization is that of reducing misclassified instances in our training data, which can be done using techniques such as gradient descent. With gradient descent we search through our hypothesis space for a $\mathbf{w}$ that best fits our training data. This embodies the fact that we need an error function for defining misclassified instances after which we update our $\mathbf{w}$. This process is repeated until there is no more improvement.

5.3.1 Error Definition

First we define the set containing misclassified instances: $D' = \{d \in D | error(\hat{y}, y) > 0\}$ with $d = \{\mathbf{x}, y\}$ and $\hat{y}$ being the predicted label. Since we are only interested in wrong classified samples, we define the *error* as :

$$error(\hat{y}, y) = \begin{cases} 0 & \text{if } y = 1, \hat{y} \geq 1 \\ 0 & \text{if } y = -1, \hat{y} \leq -1 \\ (\hat{y} - y)^2 & \text{Otherwise} \end{cases} \tag{5.2}$$

Equation (5.2) describes the amount of error one instance has, for the whole misclassified set this would be:

$$E = \frac{1}{2} \sum_{d \in D'} error(y_d, \hat{y_d})^2 \tag{5.3}$$

5.3.2 Gradient Descent Derivation

In order to find a better solution for our $\mathbf{w}$ we need to find the direction of our error. We accomplish this by taking the gradient of the error-function E with respect to the variables which we want to optimize, $\nabla E(\mathbf{w})$, where $\nabla E(\mathbf{w})$ is given in [16] by:

$$\nabla E(\mathbf{w} \in \mathbb{R}^n) = \left[\frac{\partial E}{\partial \mathbf{w}_1}, .., \frac{\partial E}{\partial \mathbf{w}_n} \right] \tag{5.4}$$

Figure (5.4) shows how the error landscape would look for $\mathbf{w} \in \mathbb{R}^2$. Intuitively $\nabla f(\mathbf{x})$ gives the rate of change, which in this specific case is the error. Of course we do not want to increase the error but rather decrease it. This means that we need to update our $\mathbf{w}$ in the opposite direction resulting in the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \left[-\eta \frac{\partial E}{\partial \mathbf{w}} \right] \tag{5.5}$$

with η corresponding to the learning rate.

General Case:

First we will give the general derivation of the gradient descent after which the solutions for the $WRBF$ and $WTanh$ are presented.

We start with our defined error function (5.3), which for the set of misclassified instances equals to $(y - \hat{y})^2$. Let Ψ be our output function:

$$\hat{y} = \Psi(\mathbf{x}) \tag{5.6}$$

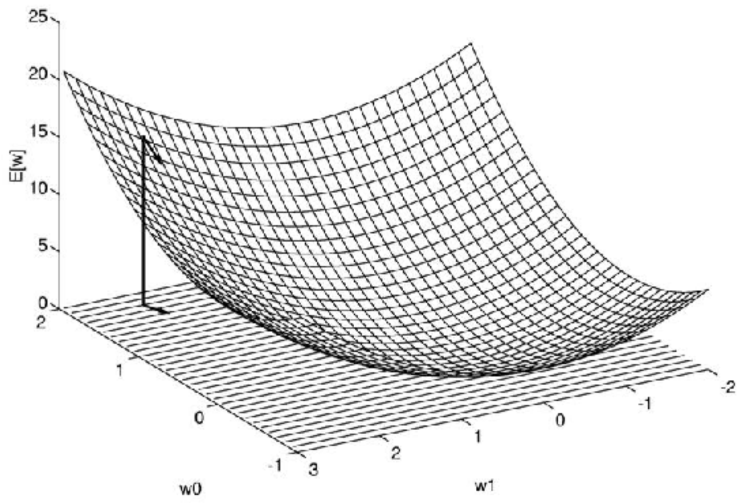


Figure 5.4: Error of different locations in the hypothesis space. The arrows describe the negative error/rate of change **w** has with respect to our target function. The horizontal plane describes the Hypothesis and the vertical axis describes the corresponding error.

Now differentiating E with respect to w gives us (for simplicity we do not insert the subscript d):

$$\begin{aligned}
E = & \quad \frac{1}{2}\sum_{d\in D}(y-\hat{y})^2 && (5.7)\\
\frac{\partial E}{\partial \mathbf{w}_i} = & \quad \frac{\partial}{\partial \mathbf{w}_i}\frac{1}{2}\sum_{d\in D'}(y-\Psi(\mathbf{x}))^2 && (5.8)\\
= & \quad \sum_{d\in D'}\frac{\partial}{\partial \mathbf{w}_i}\frac{1}{2}(y-\Psi(\mathbf{x}))^2 && (5.9)\\
= & \quad \sum_{d\in D'}\frac{\partial(y-\Psi(\mathbf{x})))}{\partial \mathbf{w}_i}(y-\Psi(\mathbf{x})) && (5.10)
\end{aligned}$$

Equation (5.10) shows the final general result of our gradient descent and can be used for squared error minimization.

5.4 Weighted-RBF

The activation of the Radial Basis Function with homogeneous weights in kernel/feature space is defined as:

$$K(\mathbf{x}, \underline{\mathbf{x}}) = e^{-\gamma \sum_{j=1}^{n} \mathbf{w}_j (\mathbf{x}_j - \underline{\mathbf{x}}_j)^2} \tag{5.11}$$

The SVM classifies an instance x as:

$$sign(\Psi(\mathbf{x})) = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}, \underline{\mathbf{x}}_i) \tag{5.12}$$

where m is the number of support vectors for the corresponding SVM and α_i the coefficient for ith support vector.
Since we use D' instead of D it is possible to directly insert equations (5.11) and (5.12) into equation 5.10.

$$\begin{aligned} \frac{\partial(y - \Psi(\mathbf{x}))}{\partial \mathbf{w}_i} &= \frac{\partial\left(y - \sum_{j=1}^{m} \alpha_j y_j e^{-\gamma \sum_{k=1}^{n} \mathbf{w}_k (\mathbf{x}_k - \underline{\mathbf{x}}_k)^2}\right)}{\partial \omega_i} && (5.13) \\ &= -\sum_{j=1}^{m} -\gamma \alpha_j y_j (\mathbf{x}_i - \underline{\mathbf{x}}_i)^2 e^{-\gamma \sum_{k=1}^{n} \mathbf{w}_k (\mathbf{x}_k - \underline{\mathbf{x}}_k)^2} && (5.14) \end{aligned}$$

Which results in:

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{w}_i} &= \sum_{d \in D} \left[(y - \Psi(\mathbf{x})) \sum_{j=1}^{m} \gamma \alpha_j y_j (\mathbf{x}_i - \underline{\mathbf{x}}_i)^2 e^{-\gamma \sum_{k=1}^{n} \mathbf{w}_k (x_k - \underline{\mathbf{x}}_k)^2} \right] && (5.15) \\ &= \sum_{d \in D} \left[(y - \Psi(\mathbf{x})) \sum_{j=1}^{m} \gamma \alpha_j y_j (\mathbf{x}_i - \underline{\mathbf{x}}_i)^2 K(\mathbf{x}, \underline{\mathbf{x}}_j) \right] && (5.16) \end{aligned}$$

Equation(5.16) shows the final derivation which can be used for a gradient descent procedure for finding optimal weights.

5.5 Weighted-Tanh

The beginning of the derivation is equal to that of the weighted-RBF. Therefore we will start at:

$$\frac{\partial E}{\partial \mathbf{w}_i} = \sum_{d \in D} \frac{\partial(y - \Psi(\mathbf{x})))}{\partial \mathbf{w}_i} (y - \Psi(\mathbf{x})) \tag{5.17}$$

In this case we should redefine K since we are deriving the derivative of the Wtanh-kernel with respect to $\mathbf{w}$.

$$K(\mathbf{x}, \underline{\mathbf{x}}) = \tanh(\gamma \cdot \langle \mathbf{w} \cdot \mathbf{x} \cdot \underline{\mathbf{x}} \rangle + B) \tag{5.18}$$

$$\frac{\partial(y - \Psi(\mathbf{x}))}{\partial \mathbf{w}_i} = \frac{\partial(y - \sum_j^m \alpha_j y_j \tanh(\gamma \cdot \langle \mathbf{w} \cdot \mathbf{x} \cdot \underline{\mathbf{x}} \rangle + B)}{\partial \mathbf{w}_i} \tag{5.19}$$

$$= - \sum_j^m \frac{\gamma \alpha_j y_j (\mathbf{x}_i \cdot \underline{\mathbf{x}_i})}{\cosh^2(\gamma \cdot \langle \mathbf{w} \cdot \mathbf{x} \cdot \underline{\mathbf{x}} \rangle + B)} \tag{5.20}$$

Which results in

$$\frac{\partial E}{\partial \mathbf{w}_i} = \sum_{d \in D} - \left[(y - \Psi(\mathbf{x})) \sum_j^m \frac{\gamma \alpha_j y_j (\mathbf{x}_i \cdot \underline{\mathbf{x}_i})}{\cosh^2(\gamma \cdot \langle \mathbf{w} \cdot \mathbf{x} \cdot \underline{\mathbf{x}} \rangle + B)} \right] \tag{5.21}$$

$$\tag{5.22}$$

5.5.1 Margin-Maximization

This chapter started with the concept of Error-Minimization. After having trained a Support-Vector-Machine (SVM), the misclassified instances were used for correcting the error caused by **w**. The approach of Margin-Maximization is slightly different. Instead of optimizing **w** in the error/misclassification-domain we intend to use the dual-objective function, equation (3.17), used in SVM training. This objective function describes the margin between the training patterns and the hyperplane (figure (3.1)). The objective of the SVM is to create the largest margin possible. We intend to improve this optimization/maximization by choosing **w** such that the SVM can increase its maximization. To achieve this we need to find the direction of the improvement with respect to **w**:

$$\text{maximize } \frac{\partial \text{dual}}{\partial \mathbf{w}} \tag{5.23}$$

To make it more comprehensive it will be solved in two steps:

$$\frac{\partial \text{dual}}{\partial \mathbf{w}} = \frac{\partial \text{dual}}{\partial K} \times \frac{\partial K}{\partial \mathbf{w}} \tag{5.24}$$

with K being kernel-function $K_{wrbf}(\mathbf{x}_i, \mathbf{x}_j)$ Solving the first part of equation (5.24):

$$\frac{\partial \text{dual}}{\partial K} = \frac{\partial \left[\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j exp \left(-\gamma \sum_{r=1}^{s} \mathbf{w}_r (\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2 \right) \right]}{\partial \left[exp \left(-\gamma \sum_{r=1}^{s} \mathbf{w}_r (\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2 \right) \right]}$$

$$= -\alpha_i \alpha_j y_i y_j \tag{5.25}$$

We dropped the $\frac{1}{2}$ for simplicity. And for the second part:

$$\frac{\partial K}{\partial \mathbf{w}} = \frac{\partial exp\left(-\gamma\ \sum_{r=1}^{s} \mathbf{w}_r(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2\right)}{\partial \mathbf{w}} \tag{5.26}$$

For one particular dimension $\mathbf{w}_n$:

$$\frac{\partial K}{\partial \mathbf{w}_n} = -\gamma(\mathbf{x}_i - \mathbf{x}_j)^2 exp\left(-\gamma\ \sum_{r=1}^{s} \mathbf{w}_r(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2\right) \tag{5.27}$$

Combining equations (5.25) and (5.27) forms the filter function for instances i and j:

$$\begin{aligned}\frac{\partial \text{dual}_{i,j}}{\partial \mathbf{w}_n} &= \Big[-\alpha_i\alpha_j y_i y_j\Big] \times \Big[-\gamma(\mathbf{x}_{i,n} - \mathbf{x}_{j,n})^2\Big] \times \Big[exp\left(-\gamma\ \sum_{r=1}^{s} \mathbf{w}_r(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2\right)\Big] \\ &= \gamma\alpha_i\alpha_j y_i y_j\Big|_1 (\mathbf{x}_{i,n} - \mathbf{x}_{j,n})^2\Big|_2 K(\mathbf{x}_i, \mathbf{x}_j)\Big|_3 \end{aligned} \tag{5.28}$$

Note: equation (5.28) can be used as an update rule: $\mathbf{w} = \mathbf{w} - \eta\frac{\partial dual_{i,j}}{\partial \mathbf{w}}$

Equation (5.28) consists of three different parts:

(part 1) $\gamma\alpha_i\alpha_j y_i y_j$, describes the magnitude and direction. This direction is caused by $y_i * y_j$, depending on class similarity this is either 1 or -1. It is the $\alpha's$ that determine the importance of the comparison.

(part 2) $(\mathbf{x}_{i,n} - \mathbf{x}_{j,n})^2$, shows the similarity of the instances i, j with respect to the dimension n. The higher the similarity the lower the second part becomes. Therefore this part emphasizes dissimilarity on single feature-space level.

(part 3) $K(\mathbf{x}_i, \mathbf{x}_j)$, describes the total similarity of the instances $\mathbf{x}_i$ and $\mathbf{x}_j$.

Upon studying all the different parts together we notice that it can be used to control the role of variance between instances that have an overall similarity. By detecting which feature-space dimensions have variance under overall similarity, we are able to exploit that concept.

Part III

Experiments

Chapter 6

Experiments

6.1 Introduction

The first experiment in our series is of an explorative nature. Gaining experience [13, 2] in using the Support Vector Machine is essential for understanding its behavior. Therefore Chapter (7) consists of straightforward SVM training and testing using an unmodified version of [3]. After every test the main properties of the SVM system will be stored and analyzed. The study of the effect on the pre-calculation of the meta-parameters was included. Since all the tools needed for this experiment were already available, we decided that such an extension would be easily integrated. In the following experiment, in Chapter (8), we will demonstrate our novel feature-variance filter. With this we aim to filter out those features most informative for a particular class. We will apply a feature selection method which suppresses those features that show high variance and strong similarity. We extend the idea of filtering in the experiment in Chapter (9). Here however we move from "feature selecting" to "feature weighing". The Weighted-Radial-Basis function is used for Feature weighing. Using this metric we are able to give different weighings for each individual feature. In Chapter (10) we extend the train algorithm with an controller. This controller measures the rate of change on the dual objective function and will ensure that the most fit model will be

Figure 6.1: Illustration depicting Pythagoras performing harmonics experiments with stretched vibrating strings. From Franchino Gafori, Theorica Musice (Milan, 1492).

used for final testing. The feature-variance filter algorithm works on the concept of margin maximization and was covered in Chapter (5).

Chapter (5) also suggest the approach of error minimization for feature-space learning. We conducted extensive studies on error minimization. However, due to bad performances, we decided not to include these results in the experiment section. Error minimization suffers from two main defects. The first is the concept itself: in order to reduce the amount of error you need to have misclassified instances in your training model. As we will come to see in Chapter (7) the high training accuracies coming from SVM optimization result in just a few training-errors. Correcting the model on only a few instances however reeks of over fitting, and bring us to the second problem: in order to detect over fitting you need to leave out a subset of your actual training data. This affects the number of training data available used to estimate the underlying function describing the data.

Chapter 7

Data Exploration and Performance Analysis Of a Standard SVM Implementation

7.1 Introduction

Upon reading articles and books one finds oneself wondering if the authors had an instinctive idea on how to "tackle" their problem, and if the experiments were conducted according to a well thought-out plan. This of course is almost never the case: new problems and opportunities arise constantly during exploring and testing, such that research becomes an iterative process. In order to get a good idea of a problem it is a good idea to start-off by examining the data, and by conducting test performances with several algorithms and/or data sets. Since our goal is to improve the Support Vector Machine it should be of no surprise that we need to test the current performance using a well-known SVM library [3]. During this step we will examine and analyze some properties of an SVM. We consider the characteristics of an SVM, without diving into formalities and details; it is more an overview of ideas and intuitions. This overview of performances is vital for our main goal -"improving classification accuracy using SVM"- therefore we need well documented comparison data. This Experiment will mainly be used to understand and work with the Support Vector Machine rather then presenting novel scientific ideas.

7.2 Properties and Measurements

One of the most important properties of any classifier is undoubtedly its accuracy. The interesting part is, of course, how it achieves these accuracies. If we take a look at the definition of an SVM classification (equation (7.1)), we see that SVM classification is made out of many different gears.

$$g(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{7.1}$$

(1^{st} Gear) The α_i is the gear which gives the SVM its name, the "support vector". Every SVM has l supporting gears which make up the support vectors. If we denote the l variables as the complexity of the system then there could exist a relation between complexity and accuracy. In a Linear Vector Quantization system [9] one could argue that the L prototypes present different target functions, where classification is based on distances to those prototypes. The complexity of such a system can be expressed in L. By having just a few prototypes one reduces the complexity of that system. One argument is that the less complex (prototypes) the system is the more it generalizes. For our system this could mean that an SVM with a small number of l could outperform the more complex ones.
(2^{nd} Gear) Another variable that we observe is the K. This gear maps our data into a different space in which the SVM has a greater chance of separating the data. Different mapping functions should yield different behaviors, if we were to find a K_2 mapping function that increases the predictive power of an SVM then we would be able to improve its accuracy.
(3^{rd} Gear) Not all parts of a system are transparent, and can not be observed directly. For example the values coinciding with the α_i are determined by an optimization algorithm. It is therefore important to understand how this works, and if improvements in this optimizing algorithm are possible.

7.3 First Steps

At the current moment the Radial Basis Function is most often used in SVMs. This mapping function -which is described in almost every machine learning book- is controlled by two variables γ and C. Both variables influence the activation of the RBF, and are called *Meta-Variables*. As mentioned in the previous Chapter, in order to improve a system we need to know the role of each individual part. The first research we will conduct is therefore of an explorative nature. We will use several different data sets and gather the following information: (1) test-accuracy; (2) training-accuracy; (3) number of support vectors; and (4) optimal meta parameters C and γ.

In classification algorithms it is not uncommon to pre-calculate the Meta-Variables since it is too time consuming to do this for every experiment. The drawback of this method is that in pre-calculation often the whole data set is used, thereby violating the notion of unseen data on test accuracy.

7.3.1 The Exploration Experiment Design

In this first, explorative experiment, we examine the performance as well as different properties of our Support Vector Machine on different data sets. Comparing test and train accuracies can help to detect over-fitting. Furthermore we would like to test whether or not pre-calculation on the Meta-Variables has an effect on the accuracy.

Every data set is tested 1000 times, which corresponds to 1000 reshuffles of the test and train sets. The train set consist out of 80% of the original data and the test fills up the last 20%. Both algorithms will perform computations

Trial	A	B	$\delta =$ A-B
1	p_1	p_1	
2	p_2	p_2	
3	p_3	p_3	
n	p_n	p_n	
			$\bar{x}_\delta$

Table 7.1: Paired sample t test.

on the same data set as can been seen in table (7.1). By ensuring that both algorithms are applied to the same data set, we are able to calculate their differences ($\bar{x}_\delta$) and the standard deviation (s_δ) on those differences. Our null hypothesis is that there are no difference between the two populations:

$$H_0 = \mu_\delta = 0 \tag{7.2}$$
$$H_1 = \mu_\delta \neq 0 \tag{7.3}$$

We choose for a two-sided test since the relation between the two algorithms is uncertain, even though we expect that pre Meta-parameter optimizing outperforms the normal. We present both t and the confidence level. All the classification and Meta-Variables finding software used comes with the default LIBSVM [3] package.

7.3.2 Comparing Properties

The second part of this research takes a first glance at the underlying principles of a Support Vector Machine namely: the support vectors and the meta-variable influences. We are interested in the following: (1) What is the relationship between accuracy on number SVs; (2) What is the relationship between accuracy and meta-variables; and (3) What is the relationship between number SVs and meta-variables.

Accuracy vs number SVs

The support vectors are used for prototype matching, where every SV gives a level of similarity. Therefore we have reasons to believe that the number of SVs used has influence on the accuracy.

Accuracy vs Meta-Variables

In the case of the Radial-Basis-Function Kernel in an SVM setting we need to provide two Meta-Variables, γ and C. The γ gives the influence radius of RBF and the C is used to control the optimization. Since we are also testing whether or not pre-meta calculation leads to different classification it would be interesting to see their inner-relationship.

number SVs vs Meta-Variables

One of the more interesting relationships is that of meta-variables and SVs. We know that γ increases the radius, it might be the case that for larger radii a smaller amount of SVs is needed to cover the feature-space,

7.3.3 Correlation

In order to find relationships between our data points we propose to calculate there correlation coefficients as follows (documentation numerical python):

The relationship between the correlation coefficient matrix, P, and the covariance matrix, C, is
$P_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$

Covariance indicates the level to which two variables vary together. If we examine N-dimensional samples, ‘$X = [x_1, x_2, ...x_N]^T$‘, then the covariance matrix element C_{ij} is the covariance of x_i and x_j. The element C_{ii} is the variance of x_i.

7.4 Results

Tables (7.2-7.9) report the results found. For every table the first two rows describe the mean and standard error for a given parameter. These parameters consist of:

- $\%Acc_{test}$: accuracy acquired on the test set.
- $\%Acc_{train}$: accuracy acquired on the train set.

- $\#SVs$: the number of support vectors used.
- $\%SVs$: total percentage of support vectors used.
- C: the cost used in softmargin optimization.
- γ: determines the width of the RBF kernel.

Both rows have entries `Const` and `Var`, these describe how the gridsearch was performed. With `Const` gridsearch was performed on the whole data set (including test data), whereas in `Var` only the train data was used. All tables end with a statistic on the differences between accuracies acquired on the test set. A two sided Paired sample t test was performed using a 5% confidence interval.

Table 7.2: Breast-Cancer-Wisconsin

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	96.76	97.15	61.75	11.03	8.0	0.008
	`Var.`	96.41	97.24	76.14	13.6	1119.6	0.09
σ	`Const.`	1.37	0.34	3.51	0.63		
	`var.`	1.46	0.5	34.31	6.13	5121.4	0.21

`T-test`	`p`	`t`	σ	`ci`	`alpha`	`df`
	< 0.01	14.234	0.775	<0.30,0.397>	0.05	999

Table 7.3: Ecoli

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	82.39	87.54	185.39	68.92	128	0.008
	`Var.`	85.96	90.64	141.03	52.43	1179.5	2.6
σ	`Const.`	5.31	1.56	4.89	1.82		
	`var.`	4.17	1.72	16.5	6.13	5235.4	3.05

`T-test`	`p`	`t`	σ	`ci`	`alpha`	`df`
	< 0.01	-25.652	4.3901	<-3.8336,-3.2888>	0.05	999

Table 7.4: Glass

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	68.54	77.59	143.51	83.44	32.0	0.125
	`Var.`	67.72	88.91	128.67	74.81	6508.8	0.97
σ	`Const.`	7.15	1.82	4.53	2.63		
	`var.`	7.37	6.12	12.38	7.2	11375.0	1.85

`T-test`	`p`	`t`	σ	`ci`	`alpha`	`df`
	< 0.01	4.7455	5.4737	< 0.48176,1.1611>	0.05	999

Table 7.5: Hepatitis

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	83.03	95.61	64.97	52.4	2.0	0.125
	`Var.`	80.66	96.17	67.5	54.44	1597.0	0.17
σ	`Const`	6.1	1.09	4.03	3.25		
	`var.`	6.51	3.89	23.41	18.88	6466.6	0.2

	`p`	`t`	σ	`ci`	`alpha`	`df`
`T-test`	< 0.01	16.599	4.517	<2.0907,2.6513>	0.05	999

Table 7.6: Ionosphere

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	94.42	99.18	79.79	28.39	8.0	0.125
	`Var.`	93.53	98.78	114.06	40.59	11.5	0.25
σ	`Const`	2.5	0.29	4.88	1.74		
	`var.`	2.66	1.15	39.36	14.01	23.85	0.19

	`p`	`t`	σ	`ci`	`alpha`	`df`
`T-test`	< 0.01	13.905	2.03	<0.767,1.019>	0.05	999

Table 7.7: Iris

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	95.25	96.15	72.17	60.14	128	0.008
	`Var.`	94.28	96.35	54.58	45.49	2759.3	0.34
σ	`Const`	3.8	1.0	2.98	2.49		
	`var.`	5.71	3.04	22.81	19.01	7930.2	0.81

	`p`	`t`	σ	`ci`	`alpha`	`df`
`T-test`	< 0.01	6.218	4.9335	<0.664,1.276>	0.05	999

Table 7.8: Pima-Indians-Diabetes

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	77.11	78.09	321.14	52.22	3967.8	0.0005
	`Var.`	76.65	78.93	337.15	54.82	3967.8	0.09
σ	`Const`	2.95	0.86	8.36	1.36		
	`var.`	2.99	1.37	25.04	4.07	9474.4	0.17

	`p`	`t`	σ	`ci`	`alpha`	`df`
`T-test`	< 0.01	9.623	1.491	<0.361,0.546>	0.05	999

7.4.1 Conclusion

All the results described by tables (7.2-7.9) support the rejection of our *null-hypothesis*. This rejection literally means that there is no supporting evidence that the results with meta-variable pre and non-pre calculation are drawn from the same population. In order to perform the most fairest of classification we should use non-pre meta-variable calculation. Observing the data we notice one perhaps counter intuitive data point. For the ecoli data set we see that the pre-calculation of the meta-variables leads to a lower classification accuracy. It is not part of this research to investigate this phenomena therefore at best we can guess at the cause.

7.5 Correlations

Upon studying tables (7.10-7.17) we found no interesting relationships. We see that the accuracy on test is most of the time negative correlated with the accuracy on train. This may be explained by the fact there is overfitting to the training set. We see that number of SVs usually is positive correlated with γ: gamma controls the radius of influence of RBF, if the RBF influence decreases then logically more SVs are needed to generate the hypothesis space. Tables (7.10-7.15) report the results:

Table 7.9: *Congressional Voting Records*

		$\%Acc_{test}$	$\%Acc_{train}$	$\#SVs$	$\%SVs$	$Cost$	γ
μ	`Const.`	96.11	99.37	130.17	37.40	2.0	0.125
	`Var.`	95.79	97.98	58.48	16.80	1146.4	0.025
σ	`Const`	1.94	0.26	4.96	1.42		
	`var.`	2.03	1.06	29.46	8.47	4754.1	0.039

`T-test`	`p`	`t`	σ	`ci`	`alpha`	`df`
	< 0.01	7.335	1.3992	<0.239,0.4124>	0.05	999

Table 7.10: Breast-Cancer-Wisconsin

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.871	0.233	0.037	0.105
$\%Acc_{train}$	-0.871	1.0	-0.2	-0.024	-0.09
SVs	0.233	-0.2	1.0	-0.194	0.605
$Cost$	0.037	-0.024	-0.194	1.0	-0.09
γ	0.105	-0.09	0.605	-0.09	1.0

Table 7.11: Ecoli

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.019	0.154	0.048	-0.027
$\%Acc_{train}$	-0.019	1.0	0.02	-0.05	0.083
SVs	0.154	0.02	1.0	0.023	0.083
$Cost$	0.048	-0.05	0.023	1.0	-0.19
γ	-0.027	0.083	0.083	-0.19	1.0

Table 7.12: Glass

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.669	0.136	0.011	0.119
$\%Acc_{train}$	-0.669	1.0	-0.119	0.001	-0.095
SVs	0.136	-0.119	1.0	-0.374	0.446
$Cost$	0.011	0.001	-0.374	1.0	-0.275
γ	0.119	-0.095	0.446	-0.275	1.0

Table 7.13: Hepatitis

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.264	0.059	0.041	-0.014
$\%Acc_{train}$	-0.264	1.0	-0.07	-0.04	-0.006
SVs	0.059	-0.07	1.0	-0.286	0.98
$Cost$	0.041	-0.04	-0.286	1.0	-0.216
γ	-0.014	-0.006	0.98	-0.216	1.0

Table 7.14: Ionosphere

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.25	0.174	-0.037	0.14
$\%Acc_{train}$	-0.25	1.0	-0.025	0.004	0.006
SVs	0.174	-0.025	1.0	-0.269	0.937
$Cost$	-0.037	0.004	-0.269	1.0	-0.149
γ	0.14	0.006	0.937	-0.149	1.0

Table 7.15: Iris

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.355	-0.031	0.064	-0.091
$\%Acc_{train}$	-0.355	1.0	0.122	-0.036	0.129
SVs	-0.031	0.122	1.0	-0.215	0.001
$Cost$	0.064	-0.036	-0.215	1.0	-0.142
γ	-0.091	0.129	0.001	-0.142	1.0

Table 7.16: Pima-Indians-Diabetes

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.787	0.305	-0.01	0.066
$\%Acc_{train}$	-0.787	1.0	-0.321	0.007	-0.091
SVs	0.305	-0.321	1.0	-0.295	0.307
$Cost$	-0.01	0.007	-0.295	1.0	-0.206
γ	0.066	-0.091	0.307	-0.206	1.0

Table 7.17: Congressional Voting Records

	$\%Acc_{test}$	$\%Acc_{train}$	SVs	$Cost$	γ
$\%Acc_{test}$	1.0	-0.466	0.001	0.01	-0.094
$\%Acc_{train}$	-0.466	1.0	0.067	-0.046	0.122
SVs	0.001	0.067	1.0	-0.206	0.951
$Cost$	0.01	-0.046	-0.206	1.0	-0.15
γ	-0.094	0.122	0.951	-0.15	1.0

Chapter 8

Feature Selection

8.1 Introduction

One of the dis-advantages of using regular kernels is the fact that they base their similarity on the inner-product. The inner-product cannot distinguish between important and unimportant/noisy features, and thus regards them equally important. To assign equal importance in SVMs is strange, since we have no a priori knowledge that this will provide the largest margins! In machine learning and relative fields feature selection is a technique used to determine a subset of an initial set of features. One desires a subset of features if the subset either improves computational complexity or increases the performance. *Stepwise regression* is an example of an often used technique which can be used for feature selection. Its greedy algorithm either adds the current best feature or removes the worst. The stopping criterion for adding or removing features is in machine learning often done using cross-validation methods. Other more advanced techniques are *branch and bound* [26] and *piecewise linear networks.*

The Weighted-Radial-Basis Kernel introduced in chapter (4) can also be used for feature selection. In this experiment we will add irrelevant features to an existing data set, after which we try to obtain the data set's original features. In section (5.5.1) we obtained an update function for $\mathbf{w}$ in the Support-Vector-Machine Kernel-Machine paradigm. Its complete form is described in equation (8.1):

$$\begin{aligned} \mathbf{w_r} &= \gamma\alpha_i\alpha_j y_i y_j(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2 K_{wrbf}(\mathbf{x}_i, \mathbf{x}_j) \\ K_{wrbf}(\mathbf{x}_i, \mathbf{x}_j) &= exp\left(-\gamma \sum_{r=1}^{s} \mathbf{w}_r(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2\right) \end{aligned} \tag{8.1}$$

where $\mathbf{x}_i$, $\mathbf{x}_j$ describe instances and r denotes a particular feature. In this method does not require a learning rate.

Equation (8.1) consists of three different parts, who's combination leads to a feature-variance filter. The different parts being:

- magnitude: $(\gamma \alpha_i \alpha_j y_i y_j)$
- variance: $(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2$
- similarity: $K_{wrbf}(\mathbf{x}_i, \mathbf{x}_j)$

Equation (8.1) allows us to find variance in features among instances that share similarity. We can control this variance by changing the similarity space K_{wrbf}. This control of K_{wrbf} is done through the weight vector $\mathbf{w}$, which can stress or reduce the importance of each feature. Given instances $\mathbf{x}_i$,$\mathbf{x}_j$ and a particular feature k, the following update is applied:

$$\begin{aligned} \mathbf{w}_k \leftarrow \mathbf{w}_k \; & + \; \mathbf{magnitude}(\mathbf{x}_i, \mathbf{x}_j) \\ & \times \; \mathbf{variance}(\mathbf{x}_{i,k}, \mathbf{x}_{j,k}) \\ & \times \; \mathbf{similarity}(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{8.2}$$

By controlling the magnitude we can adjust the behavior of the feature selection technique. First, however, we will reduce the magnitude to $y_i y_j$ by removing α_i, α_j and γ. These variables come from the Support Vector Machine, and for simplicity we neglect them here. By adjusting $\mathbf{w}$ during feature-selection we automatically adjust K, this, however effects our selection and is biased towards the ordering of the data. We therefore substitute K_{wrbf} with K_{rbf} in our feature-variance-filter. The K_{rbf} is the standard Radial-Basis function given by $exp\left(-\gamma \sum_{r=1}^{s}(\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2\right)$. The latter leads to our feature-variance-filter:

$$\textbf{feature-variance-filter}(\mathbf{x}_i, \mathbf{x}_j, r) = y_i y_j (\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2 K_{rbf}(\mathbf{x}_i, \mathbf{x}_j) \tag{8.3}$$

where $\mathbf{x}_i$, $\mathbf{x}_j$ describe instances and r denotes a particular feature.

From equation (8.3) we see that only the Kernel-Machine is needed for feature selection where it is even possible to substitute different kernels. By changing the magnitude (m) given by $y_i y_j$ we can distinguish four different types of behavior:

1:Opposing

$$m(y_1, y_2) = \begin{cases} 1 & \text{if } y_1 = y_2 \\ -1 & \text{else} \end{cases} \tag{8.4}$$

Using the opposing behavior one reduces same class variance while increasing opposite class variance.

2:Unsupervised

$$m(y_1, y_2) = 1 \tag{8.5}$$

In unsupervised learning the labels of the classes are unknown and the feature-variance is obtained without taking into account the different classes.

3:Same-Class

$$m(y_1, y_2) = \begin{cases} 1 & \text{if } y_1 = y_2 \\ 0 & \text{else} \end{cases} \tag{8.6}$$

In the same-class case the feature-selection scheme will only take into account instances belonging to the same class.

4:Opposite-Class

$$m(y_1, y_2) = \begin{cases} 0 & \text{if } y_1 = y_2 \\ 1 & \text{else} \end{cases} \tag{8.7}$$

The opposite-class feature-selection scheme will only take into account instances belonging to different classes.

8.2 Experiment Setup

We intend to keep this experiment as simple as possible and focus only on the problem at hand, "feature-selection". We will use the data sets previously utilized in chapter (7). For experimental reasons, we will add the same amount of noise as there are features in a data set. The latter leads to new data sets consisting of 50 % original and 50 % randomly sampled data. The goal in our experiment/algorithm is to obtain the original features, and is defined in algorithm (2). The feature-variance-filter is of the form described in equation (8.3), and will perform on all the four different types of behaviors. Algorithm (2) ends by returning the percentage of the original features located in the first half of the **sorted** array. All experiments are executed 10-fold, in each of which the noise features are re-sampled.

8.3 Results

Tables (8.1) through (8.4) were obtained using algorithm (2) and were implemented using the Kernel-Machine in PySVM. In our first rough approach to see whether or not our algorithm is able to pick up noise features we see the

Algorithm 2: Feature-Selection

```
input  : a data set $D$ of size $n$, $fvf$ = feature-variance-filter
output: percentage of original features obtained

for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow i$ to $n$ do
        for $f \leftarrow 1$ to number of features do
            $\mathbf{w}_f \leftarrow \mathbf{w}_f + fvf(D_i, D_j, f)$
        end
    end
end
sorted = $argsort(\mathbf{w})$
$m = \frac{\text{number of features}}{2}$
return (count in sorted$[1, .., m] \leq m)/m$)
```

low performance obtain by the *opposite-class* learning type seen in table (8.2). The other 3 learning types showed an average original feature retrievement of ≈73 %. Even though this retrieval seems fair we have to be cautious on its implications. For example "we know" how many features we are looking for. In later experiments the system will be complete autonomous, in the sense that we will not inform of the signal to noise ratio.

8.4 Conclusion

The *Opposite-Class* experiment leads to the worst performance, and will not be taken into account. Even though some single entries in table (8.2) show good performances, the overall performance is too low. Although *opposing* behavior shows the highest scores, it is the *same-class* behavior with the highest average. Table (8.1) shows that the average of opposing suffers from the glass data set (32.86 %) entry. This could be caused by the high class count in glass, we don't see the other behaviors suffer from it. Both the *Unsupervised* table (8.4) and *Same-Class* show good overall feature-selection. In *Unsupervised* feature-selection we see that both iris and ecoli have the lowest averages. The reasons for this might be that both classes are multi-class data sets.

From all the tables we see that hepatitis causes the most problems for obtaining the original features. Reasons for this could be the type of features and/or missing values. Some features are binary others contain a set of possibilities such as age $\{10, 20, 30, 40, 50, 60, 70, 80\}$. These non-continuous features might cause similarity and feature-selection problems.

Algorithm (2) is very simple in its approach and has low computational complexity since it only takes $(n+1) * (n/2)$ iterations making it suitable for

		Noise								
		bcw	ecoli	glass	hepat	ionos	iris	pimaI	votes	mean
Original	bcw	×	73.33	85.56	43.33	93.33	92.22	88.89	88.89	80.79
	ecoli	85.71	×	94.28	88.57	95.71	98.57	91.43	81.43	90.81
	glass	24.44	38.89	×	64.45	15.55	12.22	33.33	41.11	32.86
	hepat	73.68	65.26	62.63	×	80.52	85.79	69.47	62.63	71.43
	ionos	88.24	68.24	85.88	41.47	×	100.0	99.41	63.53	78.11
	iris	90.0	72.5	82.5	52.5	97.5	×	97.5	95.0	83.93
	pimaI	85.0	72.5	82.5	43.75	96.25	100.0	×	87.5	81.07
	votes	83.13	68.75	80.0	45.0	87.5	87.5	87.5	×	77.05
	mean	75.74	65.64	81.91	54.15	80.91	82.33	81.08	74.3	74.51

Table 8.1: The feature-variance filter used with the opposing filter strategy. In this strategy variance all label comparisons will be used. Instances having opposite label get a negative learning rate compared to instances belonging to the same class.

		Noise								
		bcw	ecoli	glass	hepat	ionos	iris	pimaI	votes	mean
Original	bcw	×	45.56	23.33	23.33	64.45	67.78	25.55	27.78	39.68
	ecoli	42.86	×	31.43	31.43	65.71	61.43	38.57	37.14	44.08
	glass	82.22	65.56	×	32.22	94.45	98.89	76.67	76.67	75.24
	hepat	49.47	44.21	30.53	×	63.68	69.47	40.0	51.05	49.77
	ionos	69.71	69.12	63.24	35.59	×	86.18	80.0	49.41	64.75
	iris	35.0	37.5	12.5	20.0	52.5	×	20.0	22.5	28.57
	pimaI	68.75	58.75	36.25	15.0	85.0	81.25	×	48.75	56.25
	votes	67.5	56.25	49.38	43.13	69.38	71.25	54.38	×	58.75
	mean	59.36	53.85	35.24	28.67	70.74	76.61	47.88	44.76	52.14

Table 8.2: The feature-variance filter with Opposite-Class strategy only compares instances that do not belong to the same class.

		Noise								
		bcw	ecoli	glass	hepat	ionos	iris	pimaI	votes	mean
Original	bcw	×	68.89	83.34	43.33	90.0	90.0	88.89	88.89	79.05
	ecoli	78.57	×	72.86	31.43	95.71	100.0	84.28	85.71	78.37
	glass	88.89	61.11	×	41.11	91.11	93.33	88.89	88.89	79.05
	hepat	71.58	64.73	62.1	×	81.05	83.68	67.37	60.53	70.15
	ionos	89.12	69.71	87.65	40.3	×	99.71	100.0	68.53	79.29
	iris	85.0	77.5	52.5	22.5	97.5	×	70.0	80.0	69.29
	pimaI	86.25	73.75	47.5	30.0	90.0	100.0	×	67.5	70.71
	votes	84.38	71.88	76.88	47.5	86.88	88.13	85.0	×	77.23
	mean	83.4	69.65	68.97	36.59	90.32	93.55	83.49	77.15	75.39

Table 8.3: Same-Class :: here the feature-variance filter only takes the variance into account obtained from classes having the same label.

		Noise								
	bcw	ecoli	glass	hepat	ionos	iris	pimaI	votes	avera	
Original	bcw	×	71.11	84.45	41.11	91.11	88.89	88.89	88.89	79.21
	ecoli	74.29	×	37.14	34.28	85.71	87.14	47.14	47.15	58.98
	glass	86.67	70.0	×	37.78	91.11	98.89	77.78	82.22	77.78
	hepat	69.47	69.47	57.36	×	80.53	84.74	66.31	60.0	69.7
	ionos	91.47	73.53	88.24	41.77	×	100.0	100.0	70.88	80.84
	iris	80.0	70.0	22.5	22.5	87.5	×	57.5	47.5	55.36
	pimaI	83.75	63.75	43.75	21.25	96.25	100.0	×	53.75	66.07
	votes	80.63	73.13	78.13	49.38	86.25	88.75	85.63	×	77.41
	mean	80.9	70.14	58.8	35.44	88.35	92.63	74.75	64.34	70.67

Table 8.4: Unsupervised :: in this strategy the labels are neglected and the overall variance is obtained.

large data sets.

Chapter 9

Uncontrolled Feature Weighing

9.1 Introduction

The experiment presented in this Chapter is a continuation of the experiment conducted in Chapter (8) on Feature Selection. In the Feature Selection experiments we removed the Support-Vector-Machine from the equations and isolated the algorithm. By isolating this we were able to determine whether or not our feature-variance-filter had the ability to detect noise. In this experiment we want to incorporate the Support-Vector-Machine and the corresponding WRBF kernel with our feature-variance-filter algorithm (algorithm (2)). This means that we will move from feature-selection to feature-weighing. In the previous algorithm all feature-space dimensions were of equal importance in the feature-variance-filter. We intend to use the Support Vector Machine to determine which of these dimensions are important for hypothesis space generation. Incorporating that information into the feature-variance-filter algorithm, we attempt to increase the performance of the Support-Vector-Machine. From chapter (3) we know that the non-zero α values correspond to the support vectors that the Support Vector Machine uses for its Hypothesis space. These α's can be interpret as cost to our objective function, which we would like to minimize. These costs will form an additional coefficient in our novel feature-variance-filter. Our belief is that: "the higher the cost of an instance, the more influential it is". We expect that by altering the weights lower costs emerge and an increase in the performance of the Support Vector machine.

9.2 Setup

The general setup for this experiment is identical to that of the Feature Selection experiment. Once again we will compile all possible combinations of data sets. Each new combination is an original data set containing f features, and f random sampled features (noise) from the second data set. Each of these combinations is re-sampled 100 times, and tested. The Support-Vector-Machine and Kernel-Machine depend on two variables C and γ. Both of these parameters will be set to 1. By controlling these meta-variables and keeping them constant we can -more accurately- assess the performance of our feature-variance-filter. One experiment consists of 100 iterations of our feature-variance-filter, equation (9.1), after which the Support Vector Machine will be re-trained.

$$\textbf{feat-var-filter}(\mathbf{x}_i, \mathbf{x}_j, r) = \alpha_i \alpha_j y_i y_j (\mathbf{x}_{i,r} - \mathbf{x}_{j,r})^2 K_{wrbf}(\mathbf{x}_i, \mathbf{x}_j) \tag{9.1}$$

where $\mathbf{x}_i$, $\mathbf{x}_j$ describe instances and r denotes a particular feature. The α and y are the support vector multiplier and label.

Algorithm (2) utilized in the previous experiments requires a small modification, and leads to algorithm (3). Since we are interested in the actual relevance of a dimension in feature-space -instead of an ordering of importance- algorithm (3) adjusts the weight vector.

As a result the following variables will be noted:

- accuracies on test-set containing 20% of the data set for both the traditional RBF and WRBF
- Dual-objective function for both RBF and WBF
- Number of support-vectors used for both RBF and WBF

Algorithm 3: Feature-Weighing

```
input : a data set $D$ of size $n$, where $d \in D = (\mathbf{x}, y, \alpha)$
input : weight vector $\mathbf{w}$, $fvf$ = feature-variance-filter
input : learning rate $\eta$
output: updated weight vector $\mathbf{w}$

initialize $\mathbf{v}$ with zero values.
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow i$ to $n$ do
        for $f \leftarrow 1$ to number of features do
            $\mathbf{v}_f \leftarrow \mathbf{v}_f + fvf(D_i, D_j, f)$
        end
    end
end
return $\mathbf{w} - \mathbf{v} * \eta$
```

9.3 Results

The results are divided into three different groups, *opposing*, *unsupervised* and *same-class*. In each of these groups there are two types of results, the cut and uncut versions. The uncut versions display all the results without looking at any variables that might indicate bad performance. Since we iterate 100 times without any stopping criteria it is possible for the learning algorithm to perform extreme poorly. We expect that we can detect poor performances by consulting the dual-objective function. This is the function the Support Vector Machine optimizes, and returns the cost obtained. For there to be an improvement we need to reduce these costs. If there is no reduction we will remove this entry. These costs are obtained before we do the actual performance measure and not after we have obtained the accuracies of both RBF and WRBF. Our findings can be found at: *http://www.ai.rug.nl/ aukepiet/results/feature_weighing.tar.gz*.

9.3.1 Opposing Classes

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	-0.03	-1.08	0.33	-3.79	0.44	0.52	-0.6
	ecol	14.51	×	-2.88	7.85	9.34	-2.91	35.32	10.21
	glas	0.17	-3.67	×	4.03	4.66	-3.26	2.72	0.77
	iono	0.0	0.0	0.0	×	0.0	0.13	0.0	0.02
	iris	4.43	1.53	0.83	2.07	×	-0.06	10.3	3.18
	pima	-2.78	-1.49	-2.66	-1.43	-3.05	×	-3.46	-2.48
	vote	0.19	3.7	3.75	0.0	0.0	3.15	×	1.8
	mean	2.75	0.01	-0.34	2.14	1.19	-0.42	7.57	1.84

Table 9.1: Opposing ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies.

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	380.48	240.65	170.74	280.87	86.34	515.51	347.67	288.89
	WRBF	283.51	236.99	171.36	280.85	70.42	516.97	342.9	271.86
Obj	RBF	-110.97	-251.35	-224.38	-121.6	-67.11	-257.9	-151.48	-169.25
	WRBF	-784.65	-210.84	-226.27	-121.66	-44.52	-284.72	-144.9	-259.65

Table 9.2: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	0.64	-0.27	0.46	-0.44	0.65	0.49	0.25
	ecol	14.51	×	-0.75	7.85	9.34	0.99	35.32	11.21
	glas	2.16	0.0	×	3.57	3.86	-1.19	3.72	2.02
	iono	0.0	0.0	0.0	×	0.0	0.75	0.0	0.13
	iris	4.43	2.22	0.92	2.07	×	-0.06	10.3	3.31
	pima	0.0	-0.49	-0.11	0.0	0.0	×	0.0	-0.1
	vote	0.5	7.52	8.53	0.0	0.0	9.57	×	4.35
	mean	3.6	1.65	1.39	2.32	2.13	1.79	8.31	3.03

Table 9.3: Opposing ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies. All data points which showed no improvement in the objective are filtered out.

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	380.23	240.44	142.72	280.85	86.34	234.71	231.68	228.14
	WRBF	301.89	234.84	142.88	280.57	70.32	236.78	222.73	212.86
Obj	RBF	-110.88	-252.17	-185.35	-120.95	-67.11	-132.5	-98.51	-138.21
	WRBF	-91.03	-205.89	-181.65	-120.04	-44.03	-131.82	-84.92	-122.77

Table 9.4: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

We can conclude from tables (9.1) that our novel approach saw an average accuracy increase of 1.84 %. This result is already very promising since we have added no “smart” selection methods, meaning: when should we choose the WRBF over the RBF.

The results showed that that: in none of the cases our feature-variance-filter was able to pick-up the real features for the ionosphere data set. The means displayed in table (9.2) are somewhat misleading. There are some extreme values that clearly are outliers. The following entry is taken from the breast-cancer-wisconsin data set:
100 94.964 94.964 -56.294696 -263311.364977 1 1 255 155
The fourth and fifth are respectively the objective-function for RBF and WRBF. The extreme values for WRBF are nearly 10.000 times greater clearly effect the whole mean drastically.

Table (9.3) and (9.4) give the averages after data points were removed in which the final objective function showed no cost reduction. This effect saw a ≈ 3 % accuracy increase using the WRBF.

9.3.2 Unsupervised

In the unsupervised experiment the feature-variance-filtering no longer depends on labels, thus simulating unlabeled data patterns. The averages obtained posses a high level of similarity to the Opposing label experiment, but they appear to be a little lower. Comparing the entry iris of table (9.1) and (9.5) we see that unsupervised even out-scores supervised. Tables (9.7) and (9.8) show the results after removing experiments which showed increase in costs. Again we see a ≈ 1.5 % increase in accuracy.

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	-0.57	0.05	0.24	0.26	0.29	0.28	0.09
	ecol	1.95	×	-13.97	0.98	6.65	-7.26	12.27	0.1
	glas	-1.29	-9.89	×	-0.05	-3.98	-3.88	2.03	-2.84
	iono	0.0	0.01	0.0	×	0.0	0.07	0.0	0.01
	iris	2.73	0.7	2.7	2.7	×	-4.77	6.7	1.79
	pima	-0.62	-0.11	-0.11	0.0	×	0.0	0.03	-0.14
	vote	0.0	3.73	2.27	0.0	0.0	1.23	×	1.21
	mean	0.4	-0.88	-1.29	0.55	0.42	-2.05	3.04	0.03

Table 9.5: Unsupervised ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies.

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	382.21	239.0	171.15	281.0	79.64	514.85	347.91	287.97
	WRBF	354.04	233.18	170.65	280.87	67.11	516.09	344.75	280.96
Obj	RBF	-114.39	-244.79	-225.37	-122.46	-62.09	-255.49	-150.53	-167.87
	WRBF	-108.52	-298.15	-1934.49	-122.4	-49.3	-257.01	-146.23	-416.58

Table 9.6: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

9.3.3 Same Class

In this experiment we allowed the feature-variance-filter to only learn from instances carrying the same class label. Event though the uncut accuracies show some negative performances (table 9.9), the cut values show an average improvement of ≈ 2 % ((table 9.11). The glass entry showed improvement over both *Opposing* and *unsupervised.* Comparing table (9.10) and (9.12) we see that the number of support vectors and dual-objective function show the most improvement of the three experiments.

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	0.1	0.42	0.23	0.24	0.49	0.18	0.28
	ecol	7.3	×	0.74	4.34	8.97	1.19	14.22	6.13
	glas	-3.01	-1.02	×	-0.88	0.0	-2.38	5.21	-0.35
	iono	0.0	0.48	0.0	×	0.0	0.55	0.0	0.17
	iris	2.73	1.89	2.62	2.7	×	0.45	7.76	3.02
	pima	0.0	-0.07	0.0	0.0	0.0	×	0.04	-0.01
	vote	0.0	7.88	3.2	0.0	0.0	1.7	×	2.13
	mean	1.0	1.32	1.0	0.91	1.32	0.29	3.92	1.63

Table 9.7: Nolabels ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies. All data points which showed no improvement in the objective are filtered out.

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	382.2	238.8	142.59	281.0	79.67	513.85	289.9	275.43
	WRBF	352.04	231.39	141.81	279.83	65.53	513.97	285.51	267.15
Obj	RBF	-114.41	-244.74	-188.58	-123.25	-62.14	-255.13	-123.86	-158.87
	WRBF	-101.39	-218.74	-185.17	-120.47	-42.91	-255.1	-117.19	-148.71

Table 9.8: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	-2.12	-0.53	-0.36	0.5	0.13	0.2	-0.36
	ecol	4.66	×	-11.57	2.11	1.46	-17.63	8.63	-2.06
	glas	1.97	-11.47	×	-0.36	1.14	-6.43	0.88	-2.38
	iono	0.0	0.05	0.12	×	0.0	-0.02	0.0	0.02
	iris	-2.7	1.5	-2.0	1.53	×	-0.4	5.2	0.52
	pima	0.0	0.0	-0.44	0.0	0.0	×	0.01	-0.07
	vote	0.0	3.88	2.2	0.0	0.0	0.27	×	1.06
	mean	0.56	-1.17	-1.75	0.42	0.44	-3.44	2.13	-0.47

Table 9.9: Opposing ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies.

9.4 Conclusion

Applying our novel feature-variance-filter on the different data set compositions we obtained higher accuracies in both cut and uncut results. Even though the three different types of filtering were slightly different, all showed an overall improvement. In the unsupervised experiment we saw that both

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	366.21	229.82	171.45	281.0	77.2	524.66	347.91	285.47
	WRBF	333.66	225.07	171.21	280.86	66.14	524.97	346.23	278.31
Obj	RBF	-107.34	-237.74	-224.49	-121.83	-61.59	-252.84	-151.27	-165.3
	WRBF	-373.69	-269.88	-749.27	-121.76	-65.24	-253.83	-148.61	-283.18

Table 9.10: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

		Noise							
		bcw	ecol	glas	iono	iris	pima	vote	mean
Original	bcw	×	0.44	0.21	-0.03	0.55	0.48	0.12	0.29
	ecol	6.53	×	2.99	5.25	6.43	0.0	11.72	5.49
	glas	2.19	0.0	×	2.01	6.35	5.55	1.06	2.86
	iono	0.0	0.57	0.6	×	0.0	-0.07	0.0	0.18
	iris	3.43	1.5	-1.68	1.8	×	2.3	8.93	2.71
	pima	0.0	0.0	0.0	0.0	0.0	×	0.01	0.0
	vote	0.0	8.77	5.23	0.0	0.0	0.34	×	2.39
	mean	1.74	1.61	1.05	1.29	1.9	1.23	3.12	1.99

Table 9.11: Opposing ::Difference measured between RBF and WRBF kernels in accuracy, positive values indicate higher WRBF accuracies. All data points which showed no improvement in the objective are filtered out.

		bcw	ecol	glas	iono	iris	pima	vote	mean
#SVs	RBF	366.29	196.38	143.15	281.0	77.26	524.22	289.91	268.32
	WRBF	337.07	189.43	142.83	280.14	64.2	524.21	286.56	260.63
Obj	RBF	-107.47	-203.18	-187.4	-122.0	-61.57	-252.36	-125.19	-151.31
	WRBF	-95.86	-182.93	-184.85	-120.41	-43.41	-252.36	-118.89	-142.67

Table 9.12: Learning different feature-spaces results in changes to the number of support vectors used, and the objective function.

glass and pima-indian diabetes show a decrease in accuracy.

In the opposing experiment we saw several compositions where the WRBF outperformed the RBF by 8 % or more.

Chapter 10

The Main Experiment: Controlled Feature Weighing

10.1 Introduction

This final experiment embodies the main experiment of this project. Here we demonstrate that by learning different weights for each individual feature -the strength each feature/dimension has on the similarity measure- we are able to obtain higher classification accuracies. These higher accuracies are due to feature-space transformation for which a lower *cost* could be obtained. We will show that *cost* reduction -optimizing the dual objective- leads to greater margins as first described in section (5.2). In short: "if the transformed feature space allows a higher double objective (greater margins) then we have reduced the *risk* of misclassification". In the main experiment we extended the experiment in chapter (9) by adding a *controller* between the Support-Vector machine and the Kernel Machine. This controller ensures that the best configuration -svm model with the highest objective- will be used for the final test and train.

The three different types of learning, i.e, (1) opposing, (2) same-class and (3) unsupervised show similar results, therefore in the sections to come we will focus only on one learning type at a time.

This chapter has the following structure: We (1) compare the difference in counts between the WRBF and RBF, (2) use *win or lose* strategy and apply this using a binomial test, (3) introduce and perform the Wilcoxon Ranked-sums test. This non-parametric statistical test ranks the difference in scores allowing it to be used for non-normal distributed data. Through ranking these differences this test is stronger than the *win or lose* strategy. (4) show the relationship between *cost* reduction and higher accuracy increasement, (5) combine and discuss the results and (6) present the conclusions.

10.2 Comparing the Number of Correctly Classified Instances

The most easy and intuitive approach for comparing two classifiers operating on the same problems is plotting their compared performance. Figure (10.1) illustrates the compared performances between the RBF (x-axis) and WRBF (y-axis) algorithms. The data points presented (4200 observations) are the

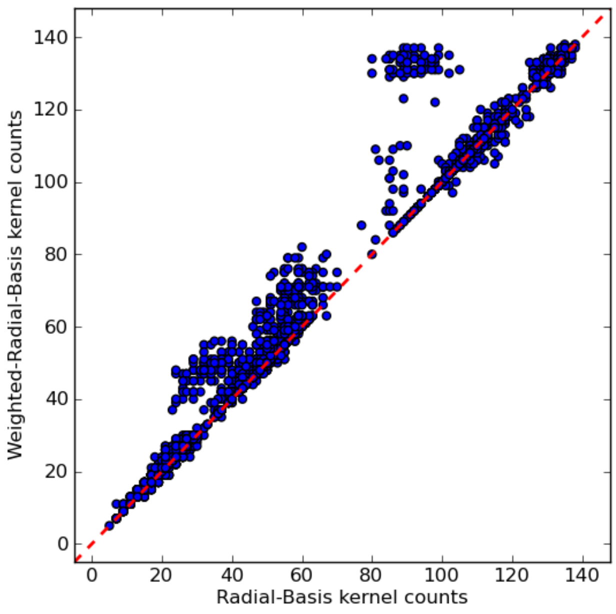


Figure 10.1: illustrates the comparison between the number of correctly classified instances by the RBF (x-axis) and the WRBF (y-axis). The red dashed line represents "equal" performance. All possible data combinations between original and noise suggests that the WRBF has classified the most instances correct, this since most data points are located at the left-side of the equality line. All 4200 observations are drawn from the *opposing* learning strategy.

test results -the number of correctly classified instances- on all combinations of "original" and "noise" data sets. The red dashed diagonal line represents equal scores between the two algorithms. We observe a large amount of observations on the left-side of this equality line suggesting that the WRBF has classified more instances correct than the RBF.

Table (10.1) gives a more detailed view on the scores obtained. It is shown by the first column that the observations are ordered by "original" data set.

Data Set	WRBF	RBF	Total	WRBF - RBF
bcw	78731	74735	83400	3996
eco	30963	28638	40200	2325
gla	11525	11365	25200	160
ion	26837	26685	42000	152
iri	16574	16072	18000	502
pim	64303	64263	91800	40
vot	34960	32581	52200	2379

Table 10.1: gives an overview on the number of correct classified instances by algorithms WRBF and RBF out of a `total` -third column- number of test instances. The Data shown is ordered by “Original” -instead of original plus noise table (A.3)- and is coherent with the assumption made in figure (10.1), that the WRBF algorithms has classified more instances correct than the RBF.

The `WRBF - RBF` column states that the WRBF has classified more instances correct -all data set entries- than the RBF approach. If we were to zoom in on each “original plus noise” entry -see table (A.3) for detailed information- then the RBF outperformed the WRBF in 3 out of the 42 data sets, with 11 ties. The latter performances indicate that if there is a difference in performance -in 31 observations- then in most cases the WRBF performs better. However in order to make statistical claims we first need do some analysis.

Analysis Introduction

In [7] the machine learning community has been accused of ignoring the statistical validity for the comparison of two or more algorithms on multiple data sets. They warn against the misusage of T-test (Paired). As stated in [7] -describing the same context as our experiment- the t-test suffers from 3 weaknesses.

(1) Commensurability: the question arises whether or not comparing the magnitude in the differences between different data sets “makes at all sense”. This since we know that comparing the averages between the classifiers on different data sets make no sense, then why should the differences?

(2) Sample Size: using the paired t-test requires that the differences between the algorithms on the different data sets are normally distributed, with a minimum of 30 samples, both premises are often violated.

(3) Outliers: the t-test is affected by outliers causing some skewness on the distribution.

Therefore they state that: if the performance differences on data sets from which we pair the differences suffers from one of the 3 weaknesses mentioned, the paired t-test becomes incorrect and misleading.

In [7, 23, 10] they recommend the usage of non-parametric alternatives to the paired t-test, such as the *Wilcoxon Signed-Rank Test*, *Friedman Test* and a more simple comparison method of *win or lose* in conjunction with a Binomial distribution.

As mentioned in previous experiments our data set collection might suffer from a fourth weakness, namely independence. We have to consider the fact that by combining multiple data sets we might have caused a bias for one classifier over the other, since the "original" data set features will appear in several data sets. To counter attack this weakness we will perform analysis on: (1) assume that there exist only 7 data sets -combine all the noise- and (2) assess every combination -original plus noise- individually.

The Sign Test: "Win or Lose"

A very simple, yet often used method is to simply count the number of data sets on which an algorithm is the overall winner. The algorithm with the most counts can be regarded as the "better".

These counts can also be used in inferential statistics, where they are used in conjunction with a binomial distribution. Let A and B be algorithms and measure their performances on a set of data sets. One observation is measured as: which algorithm has won the most trials (folds) for a particular data set. If n is the number of data sets for which A and B compute independent answers ,we are observing Bernoulli trials, then we can use the binomial test for comparison. Let s (success) be the number of times $(A > B)$, and f (failure) the number of time $(B > A)$. In our experiment this s would be: $(WRBF > RBF)$. Consider the idea that algorithms A and B are approximately equal ,the probability of $A > B$ is equal to $B < A$, then the expected value E(s) = $0.5n$. Suppose we want to test whether or not $WRBF > RBF$ and denote a *win* on data set n_i as a *success*, then our statistic becomes:

$$P(s \geq \textbf{observed value} | p(\textbf{success}) = 0.5) \tag{10.1}$$

Equation (10.1) describes the probability that the $WRBF$ wins over RBF, given the number of observed *successes*, assuming equal chance between success and failure. Which can easily be computed and is given by:

$$f(s, p, n) = \frac{n!}{s!(n-s)} p^s (1-p)^{n-s} \tag{10.2}$$

As one might observe we are not testing whether or not there is a significant difference between the performances of A and B on an individual data set. Even though this is often done ,filtering out what authors believe to be “random wins/losses”, its practice is highly questionable.

Let us assume that we are comparing two spam filtering algorithms A and B 600,426,974,379,824,381,952 times. Of all these comparisons algorithm A was better than algorithm B, however never significant. What would the chance of this happening be with equal p? Could A be that lucky? In short: “we are not interested if A was significantly better than B on data set i”, rather we investigate whether or not winning n out of the N data sets is significant.

Applying the Sign Test: “Binomial Test”

In this section we will apply the previously discussed sign test to compare the differences between the WRBF and RBF algorithm. We will only take into account the results gained from the “same-class” learning algorithm, since the two other show only positive results, making it impossible for us to show the weakness of this test. In the final section we will discuss the results of the other two -”unsupervised” and “opposite-class”- learning algorithms.

Number of instances: 4198

$P(s >= \textbf{observed value}|p(\textbf{success}) = 0.5)$

Data set	$\alpha_{one-sided}$	$\alpha_{two-sided}$	n	s	f	$s > f$
bcw	0.0	0.0	383	299	84	+
eco	0.0	0.0	556	520	36	+
gla	0.0	0.0	368	295	73	+
ion	0.0	0.0	55	54	1	+
iri	0.0	0.0	331	251	80	+
∗pim	0.87	≈ 1	$\underline{13}$	4	9	-
vot	0.0	0.0	287	274	13	+
Total	0.0625	0.125	7(n)	6(s)	1(f)	

Table 10.2: gives the scores -win or lose- of the original data sets when comparing the WRBF against the RBF applying the “same-class” learning type. The `n`, `s` and `f` columns respectively are the number of unequal ($WRBF <> RBF$) correct classification, number of *successes* ($WRBF > RBF$) and the number of *failures* ($WRBF < RBF$) . The acquired loss by WRBF -marked by *- on pim results in a non-significant improvement over the RBF. The underlined entry -$\underline{13}$- at pim shows that the loss endured was based on small n, nevertheless it has equal amount of influence as the high n entry winnings.

There is one row -marked by *- in table (10.2) in which the RBF outperformed the WRBF. One notices the small amount of data samples -namely

13- in which there occurred differences between the two algorithms on the pim entry. The low number of N data sets makes the binomial test vulnerable to rejecting: "that there exists no differences between the two algorithms". In order to allow one mistake, and still assuming that the algorithms are different, we need at least 9 (N) data sets. Another weakness is the neglection of the magnitude in differences between the different datasets. Even though data entry eco (Ecoli) has a total of 520 wins over 36 losses it has equal contribution to the statistic as pim (Pima Indians Diabetes).

10.3 Wilcoxon Signed-Ranks Test

The Wilcoxon signed-ranks (1945) is an non-parametric alternative to the paired-t and binomial test. The test *ranks* the absolute differences in performances between the classifiers and uses this ranking to draw statistical conclusions (illustrated in table (10.3)). By ranking we have counter-attacked the weakness the binomial test suffers from, since observations with small n -pim entry in previous section- can never achieve high ranks. As mentioned the differences are ordered according to their absolute values, in the case of ties average ranks are assigned. Let d_i be the difference in performance between algorithms A and B on the i-th of N data sets.

`data set`	1	2	3	4
`diff`	-2	-3	1	4

⇒

`data set`	3	1	2	4
`Rank`	1	2	3	4
`|diff|`	1	-2	-3	4

Table 10.3: The Wilcoxon Signed-Ranks Test orders the observations by the absolute difference in performance. In doing so we compare the differences qualitatively: "better scores count more", however we ignore their absolute differences since we have moved from magnitudes to *ranking*.

Let R^+ be the sum of ranks for the data set on which algorithm A outperformed B, and R^- sums the opposite performance. Ranks of no difference, $d_i = 0$, are divided evenly amongst A en B:

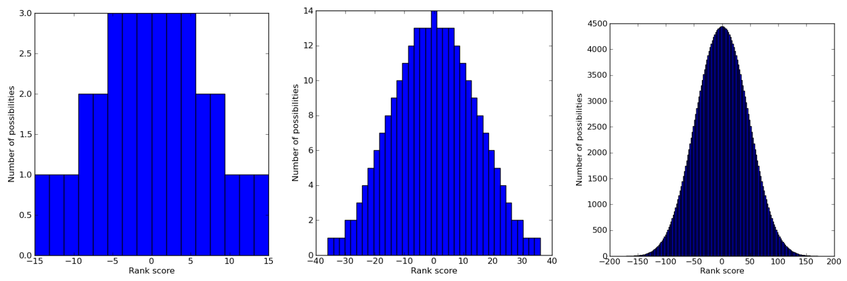
$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$
$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$

The test continues by selecting the smaller of the sums, $T = \min(R^+,R^-)$. Most statistical software programs or books contain the critical values for T

and small N testing for significant differences. For large N (25 or more) the statistic

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(N+2)}} \tag{10.3}$$

is distributed normally as shown in figure (10.2) and can be used to obtain the critical values.



(a) N = 5 (b) N = 8 (c) N = 18

Figure 10.2: illustrates that the Wilcoxon ranked-sums test rapidly converges to a normal distribution. By choosing an N of 18 the smoothness of the distribution is already apparent.

By setting the confidence of α to 0.05, we can reject the null-hypothesis if the acquired *z-score* is smaller than -1.96.

10.4 Applying the Wilcox Ranked-Sums Test

In order to compare the performances on our algorithms we first need to make sure that the scores are comparable which is illustrated in table (10.4). The first step (fifth column) is to obtain the differences (total correct classified test patterns) between the WRBF and RBF. Secondly we divide it by the total number of test patterns (fourth column) for that particular data set.

	Number of instances: 4198						
`Data set`	`WRBF`	`RBF`	`Total`	`Diff`	`D/T`	`|D/T|`	`Rank`
bcw	78092	74926	83400	3166	0.0379	0.0379	5
eco	32321	28494	40200	3827	0.0951	0.0951	7
gla	12186	11458	25158	728	0.0289	0.0289	3
ion	26658	26539	42000	119	0.0028	0.0028	2
iri	17003	16378	18000	625	0.0347	0.0347	4
pim	64387	64409	91647	-22	-0.0002	0.0002	1
vot	34717	32604	52200	2113	0.0404	0.0404	6
	$R^+ = 27.0$		$R^-= 1.0$		$R^0= 0.0$		$\|R\|= 28.0$
	$T = \min(R^+,R^-) = 1.0$						

Table 10.4: illustrates how to obtain the Wilcoxon ranked-sums test. Column `D/T` describes the ratio on differences ,as described in equation (10.4) between `WRBF` and `RBF` scores. The last two columns show the ranking on these ratios, higher obtained rankings have a bigger effect on the statistical outcome. The R variables are summed values for their class of *win, lose* and *equal* ranking as described in section (10.3). The static (T) describes the probability of obtaining $(1, 27)$ ranked-score. In order to dismiss the notion that both algorithms are equal we need to have a $T <= 2$ (specific for 7 data sets) which is indeed the case.

The sixth column gives the ratio of differences computed for all data sets,

$$d_i = \sum_k^K \sum_i^N \frac{WRBF(k,i) - RBF(k,i)}{N} \tag{10.4}$$

where K N respectively denote then number of *folds* and size of *test data*. Both functions $((W)RBF(k,i))$ give 1 for correct classification and zero otherwise.

The $T = 1$ value obtained from table (10.4) rejects (two tailed significance with 0.05 confidence) the hypothesis that algorithms WRBF and RBF can be drawn from the same population, i.e., their scores are approximately equal. The reason why Wilcoxon's test and the binomial test differ in their findings comes from the lack of *ordering* in the binomial test. By ordering/ranking the Pima-Indians Diabetes as the least important (there were only 13 observations) the Wilcoxon ranked-sums test can reject the H_0 whereas the binomial-test could not. It makes sense to stress the importance of those observations which carry more instances than others. However, we must not forget to investigate the possible bias towards merging all the noisy data sets. Table (A.8) illustrates the Wilcoxon ranked-sum test when viewing combinations of noise and original as separate data sets. For large N we can use equation (10.3) to obtain the critical values. The Obtained z-score of -4.68 is also sufficient for rejecting H_0. There is no need to elaborate on the other two algorithms since the WRBF always outperformed the RBF, when combining all the noise, resulting both times in maximum ranking. Tables (A.6) and (A.7) illustrate ranking performed on all combinations of noise and original. In both comparisons, opposing and unsupervised, the obtained z-scores (respectively -5.05 and -5.28) are also sufficient for rejecting the H_0 hypothesis.

We have shown that for both data set comparisons -joined and separated noise- the H_0 -that their is no difference in performances between the WRBF and the RBF algorithm- is rejected using the Wilcoxon ranked-sums test.

10.5 Cost Reduction And Accuracy

The task of the controller mentioned in the introduction is to ensure that all the models ,the support vector machine's output model, have lower or equal *cost* than the initial RBF algorithm. These costs are the direct result of dual objective optimization, and form the backbone of our system. The weight vector $\mathbf{w}$ we are modifying during our optimization process manipulates feature-space in such a manner that it should make the process of *maximum-margin hyperplane* optimization more easy. How this "improved" feature-space is obtained is not of any concern in this section, we only need to know that through this cost-controlled learning we only have obtained equal or better feature-spaces.

In this section we will demonstrate that there is a strong relationship between cost reduction and performance improvement. For this experiment we combined all our observations into one big result set, resulting in 12598 observations. From each observation we extracted two variables,

$$\begin{aligned}\Delta_{acc} &= WRBF_{acc} - RBF_{acc} \\ \Delta_{cost} &= WRBF_{cost} - RBF_{cost}\end{aligned} \tag{10.5}$$

allowing to compute the relationship -correlation coefficient- between accuracy improvement and cost reduction. We obtained the correlation coefficient

$r = 0.646$ over all data sets and three different learning types (N= 12598). This strong linear relation between accuracy improvement and cost reduction is significant, two-sided with $p < 0.0001$, using the statistic $t = \frac{r}{\sqrt{\frac{(1-r^2)}{(N-2)}}}$.

From the latter we conclude that improving the objective of the Support Vector Machine has strong positive effects on classification.

10.6 Conclusion

This chapter consists of three main parts (1) comparing counts between the WRBF and the RBF, (2) testing whether or not there is significant difference between the two algorithms and (3) how much of the performance increase can be described to *cost* reduction.

On comparing the two figures (10.1, A.1) we see that the WRBF has classified more instances correct than the RBF. Notice that in all three figures there are no outliers on the RBF's side of the equality line. This is one of the strong properties of our novel algorithm, it virtually never performs worse than the old method, on the other hand it shows some major improvements.

By introducing and applying the Wilcoxon Ranked-sum test we gave convincing results that the two algorithms are performing significantly different. Combining this notion and that of higher counts we conclude that our novel system of feature space weighing clearly outperforms the standard system.

This chapter ended with the section on *cost reduction and accuracy*, illustrating the power behind cost reduction / optimizing the dual objective. By combining all the data from the three different learning algorithms we showed a significant relation between cost reduction and accuracy improvement. In other words: we are optimizing the SVM more without any indication of overfitting. Therefore we are sure that other algorithms than the 3 provided can benefit from this concept.

Part IV
Discussion

Chapter 11

Discussion

11.1 Summary of the Results

In the first experiment the Support Vector Machine's (SVM) performance was tested on several different data sets. As a metric the Radial-Basis function was used in all experiments. Besides obtaining classification accuracies, and properties of the SVM's model, we also decided to test the effect of pre-calculation of the meta-parameters. In pre-calculation we used the whole available data set to acquire these meta-parameters. As expected the pre-calculation of the meta-parameters had a significant effect on accuracy. In all but one case, the pre-calculation lead to higher accuracies. Comparing the differences between the meta-parameters obtained from train-sets, with those of the pre-calculated, the reasons for the differences became clear. When meta-parameters searching is done only on the train-set, the values for C and γ take on extreme values.

In the second experiment the feature-variance filter was tested. The feature-variance filter algorithm learns the relevance that each dimension in feature-space has on the target function. We decided to first test the performance on a "stripped-down" version, resulting in a feature selection method. With this method we tested 4 different types of learning behaviors. By repeatedly mixing two UCI data sets, where only the labels of one data set was kept, we incorporated "real-world" noise as features to an existing data set. The results from this experiment show that in 3 out of the 4 behaviors the algorithm recovered approximately 73% of the original features.

In the third experiment we used the Weighted-Radial-Basis function (WRBF) as a metric in our system. The WRBF metric has the ability to scale each dimension in feature-space. Here we used the complete feature-variance filter as a learning medium between the SVM and the WRBF. By repeatitely adjusting the WRBF after each SVM optimization -where the SVM uses the updated metric of the WRBF- we obtained the learned feature-space. The final learned WRBF kernel was then used for training and classification. This experiment saw an average increase in accuracy by approximately 2.5%.

In the fourth and final experiment we decided to address the shortcomings of the learning algorithm that used in the third experiment. Even though the feature-variance filter worked satisfactorily, the third lacked an intelligent stopping criteria. This means that in the third experiment we repeated learning n times without looking at any performance measures, the n^{th} solution metric was simply chosen. In the fourth experiment we added a simple yet effective update rule: a new metric was to be chosen over the old "if and only if" the costs of the new metric were lower then the old metric. The SVM's objective function is defined in term of margin. However the value of the objective corresponds to the cost of using support vectors. Each support vector brings with it a certain cost, described by its coefficient. The intuition behind it is as follows: if the newly obtained feature-space is better, then the SVM should have less problems in computing a separating hyperplane, and result in lower costs. From the results we can see that our novel approach of feature-space learning significantly outperforms the standard SVM approach.

11.2 Conclusion

In this thesis a feature-space learning system was presented which can be used in combination with a Support Vector Machine (SVM) [25]. The primary goal for this system was to learn the underlying function which described our training data. By using a Weighted-Radial-Basis function [21] we were able to give different weights to each dimension in feature-space. For finding optimal weights we developed a feature-variance filter, capable of learning these weights.

One of the most commonly used SVM libraries [3] was modified in order to handle a feature-space weighted kernel. In order to be able to test large quantities of data, programs where developed to allow distributed computing within the university network.

The data sets used during the experiments were constructed from two existing data sets, where the label of one of the data sets was used as the target function. By ensuring that not all features where equally informative, and originated from two different data sets, we were able to test our feature-space learning system. The results showed that our novel feature-variance filter was not only successful in retrieving the original features, but it also significantly outperformed the traditional SVM with Radial-Basis function. Our results answer the research question posed in Chapter (1):

How can a Support Vector Machine maximize its predictive capabilities through feature-space learning?

The process of filtering dimensions in feature-space according to relevance has been shown to be very effective when a Support Vector Machine is used in conjunction with the Weighted-Radial-basis function. By consulting the SVM's objective function, and testing for improvement, we showed that significant greater margins can be achieved between classes, in comparison to the regular Radial-Basis function.

11.3 Future Work

In addition to feature-weighing variables the Weighted-Radial-basis function has a radius (γ). This radius describes the width of influence the support vectors have in feature-space. Since this variable is a constant coefficient, for every dimension its influence can be moved to the feature-space weight vector. The removal of γ has two mayor advantages, it: (1) drastically reduces the complexity of the gridsearch; and (2) its value is obtained through gradient-descent allowing infinite precision.

The "stripped-down" feature-variance filter can be used for feature selection and requires little computation. A comparison study with other feature selection methods can be conducted, testing whether or not it can be used for fast feature selection.

Kernels are not limited to SVMs only, and are used in a wide range of machine learning methods. Research could be done in how to incorporate the WRBF in other methods than the SVM.

Appendix A

Tables and Figures

Data Set	WRBF	RBF	Total	WRBF - RBF
bcw	78888	74693	83400	4195
eco	32778	29077	40200	3701
gla	12269	11630	25200	639
ion	26562	26427	42000	135
iri	16988	16369	18000	619
pim	64629	64625	91800	4
vot	35455	33719	52200	1736

Table A.1: illustrates the differences in counts between the WRBF and the RBF kernel using the unsupervised learning algorithm. All data entries indicate that the WRBF outperformed the RBF, even though on the pim data set one could hardly speak of improvement.

Data Set	WRBF	RBF	Total	WRBF - RBF
bcw	78092	74926	83400	3166
eco	32321	28494	40200	3827
gla	12186	11458	25158	728
ion	26658	26539	42000	119
iri	17003	16378	18000	625
pim	64387	64409	91647	-22
vot	34717	32604	52200	2113

Table A.2: illustrates the differences in counts between the WRBF and the RBF kernel using the same class learning algorithm. The pim entry shows a small loss for the WRBF kernel, all other data points indicate that the WRBF has outperformed the RBF kernel.

Data Set	WRBF	RBF	Total	WRBF - RBF
bcweco	13175	13170	13900	5
bcwgla	13278	13173	13900	105
bcwion	13240	13026	13900	214
bcwiri	13279	13198	13900	81
bcwpim	13199	13150	13900	49
bcwvot	12560	9018	13900	3542
ecobcw	4976	4666	6700	310
ecogla	5563	5568	6700	-5
ecoion	5034	4853	6700	181
ecoiri	5059	4716	6700	343
ecopim	5505	5503	6700	2
ecovot	4826	3332	6700	1494
glabcw	1894	1864	4200	30
glaeco	2440	2436	4200	4
glaion	1857	1834	4200	23
glairi	1527	1475	4200	52
glapim	2408	2383	4200	25
glavot	1399	1373	4200	26
ionbcw	4454	4454	7000	0
ioneco	4534	4503	7000	31
iongla	4493	4463	7000	30
ioniri	4374	4374	7000	0
ionpim	4563	4472	7000	91
ionvot	4419	4419	7000	0
iribcw	2735	2619	3000	116
irieco	2859	2796	3000	63
irigla	2840	2827	3000	13
iriion	2837	2757	3000	80
iripim	2822	2774	3000	48
irivot	2481	2299	3000	182
pimbcw	10569	10545	15300	24
pimeco	11132	11191	15300	-59
pimgla	11284	11209	15300	75
pimion	10723	10726	15300	-3
pimiri	10842	10839	15300	3
pimvot	9753	9753	15300	0
votbcw	5315	5315	8700	0
voteco	5959	5241	8700	718
votgla	6623	5721	8700	902
votion	5299	5299	8700	0
votiri	5303	5303	8700	0
votpim	6461	5702	8700	759

Table A.3: gives an overview on the number of correct classified instances using the opposing learning algorithm for the kernels WRBF and RBF. Out of allt the observations only 3 times did the WRBF loose to the RBF -underlined entries- “suggesting” that the WRBF alogrithms is at least not worse than the RBF algorithm.

Data Set	WRBF	RBF	Total	WRBF - RBF
bcweco	13291	13235	13900	56
bcwgla	13239	13089	13900	150
bcwion	13226	13121	13900	105
bcwiri	13285	13151	13900	134
bcwpim	13193	13083	13900	110
bcwvot	12654	9014	13900	3640
ecobcw	5485	4934	6700	551
ecogla	5667	5567	6700	100
ecoion	5393	4494	6700	899
ecoiri	5574	5210	6700	364
ecopim	5661	5608	6700	53
ecovot	4998	3264	6700	1734
glabcw	1767	1664	4200	103
glaeco	2361	2324	4200	37
glaion	2165	1992	4200	173
glairi	2112	1941	4200	171
glapim	2464	2315	4200	149
glavot	1400	1394	4200	6
ionbcw	4345	4345	7000	0
ioneco	4432	4374	7000	58
iongla	4423	4415	7000	8
ioniri	4434	4434	7000	0
ionpim	4557	4488	7000	69
ionvot	4371	4371	7000	0
iribcw	2779	2738	3000	41
irieco	2830	2807	3000	23
irigla	2845	2830	3000	15
iriion	2845	2782	3000	63
iripim	2859	2836	3000	23
irivot	2830	2376	3000	454
pimbcw	10576	10572	15300	4
pimeco	11260	11260	15300	0
pimgla	11360	11360	15300	0
pimion	10882	10882	15300	0
pimiri	10661	10661	15300	0
pimvot	9890	9890	15300	0
votbcw	5187	5187	8700	0
voteco	6420	5806	8700	614
votgla	6794	6174	8700	620
votion	5279	5279	8700	0
votiri	5305	5305	8700	0
votpim	6470	5968	8700	502

Table A.4: gives an overview on the number of correct classified instances using the unsupervised learning algorithm for the kernels WRBF and RBF. Out of allt the observations the WRBF never lost to the RBF.

Data Set	WRBF	RBF	Total	WRBF - RBF
bcweco	13240	13194	13900	46
bcwgla	13280	13207	13900	73
bcwion	13221	13062	13900	159
bcwiri	13273	13340	13900	-67
bcwpim	13190	13102	13900	88
bcwvot	11888	9021	13900	2867
ecobcw	5245	4595	6700	650
ecogla	5578	5333	6700	245
ecoion	5464	4904	6700	560
ecoiri	5557	4924	6700	633
ecopim	5607	5444	6700	163
ecovot	4870	3294	6700	1576
glabcw	2056	1866	4200	190
glaeco	2291	2225	4200	66
glaion	1973	1819	4158	154
glairi	1943	1716	4200	227
glapim	2391	2372	4200	19
glavot	1532	1460	4200	72
ionbcw	4447	4447	7000	0
ioneco	4419	4408	7000	11
iongla	4416	4371	7000	45
ioniri	4434	4434	7000	0
ionpim	4514	4451	7000	63
ionvot	4428	4428	7000	0
iribcw	2858	2695	3000	163
irieco	2813	2800	3000	13
irigla	2845	2838	3000	7
iriion	2840	2849	3000	-9
iripim	2821	2810	3000	11
irivot	2826	2386	3000	440
pimbcw	10853	10853	15300	0
pimeco	11172	11172	15300	0
pimgla	11216	11238	15300	-22
pimion	10685	10685	15300	0
pimiri	10943	10943	15300	0
pimvot	9518	9518	15147	0
votbcw	5253	5253	8700	0
voteco	6247	5515	8700	732
votgla	6263	5481	8700	782
votion	5248	5248	8700	0
votiri	5247	5247	8700	0
votpim	6459	5860	8700	599

Table A.5: gives an overview on the number of correct classified instances using the opposing learning algorithm for the kernels WRBF and RBF. Out of allt the observations only 2 times did the WRBF loose to the RBF -underlined entries- “suggesting” that the WRBF alogrithms is at least not worse than the RBF algorithm.

	Number of instances: 4200						
Data set	WRBF	RBF	Total	Diff	D/T	|D/T|	Rank
bcweco	13175	13170	13900	5	0.000359712	0.000359712	11
bcwgla	13278	13173	13900	105	0.007553957	0.007553957	26
bcwion	13240	13026	13900	214	0.015395683	0.015395683	29
bcwiri	13279	13198	13900	81	0.005827338	0.005827338	22
bcwpim	13199	13150	13900	49	0.00352518	0.00352518	15
bcwvot	12560	9018	13900	3542	0.254820144	0.254820144	42
ecobcw	4976	4666	6700	310	0.046268657	0.046268657	35
ecogla	5563	5568	6700	-5	-0.000746269	0.000746269	12
ecoion	5034	4853	6700	181	0.027014925	0.027014925	33
ecoiri	5059	4716	6700	343	0.05119403	0.05119403	36
ecopim	5505	5503	6700	2	0.000298507	0.000298507	10
ecovot	4826	3332	6700	1494	0.222985075	0.222985075	41
glabcw	1894	1864	4200	30	0.007142857	0.007142857	25
glaeco	2440	2436	4200	4	0.000952381	0.000952381	13
glaion	1857	1834	4200	23	0.00547619	0.00547619	21
glairi	1527	1475	4200	52	0.012380952	0.012380952	27
glapim	2408	2383	4200	25	0.005952381	0.005952381	23
glavot	1399	1373	4200	26	0.006190476	0.006190476	24
ionbcw	4454	4454	7000	0	0.0	0.0	4.0
ioneco	4534	4503	7000	31	0.004428571	0.004428571	19
iongla	4493	4463	7000	30	0.004285714	0.004285714	17
ioniri	4374	4374	7000	0	0.0	0.0	4.0
ionpim	4563	4472	7000	91	0.013	0.013	28
ionvot	4419	4419	7000	0	0.0	0.0	4.0
iribcw	2735	2619	3000	116	0.038666667	0.038666667	34
irieco	2859	2796	3000	63	0.021	0.021	31
irigla	2840	2827	3000	13	0.004333333	0.004333333	18
iriion	2837	2757	3000	80	0.026666667	0.026666667	32
iripim	2822	2774	3000	48	0.016	0.016	30
irivot	2481	2299	3000	182	0.060666667	0.060666667	37
pimbcw	10569	10545	15300	24	0.001568627	0.001568627	14
pimeco	11132	11191	15300	-59	-0.003856209	0.003856209	16
pimgla	11284	11209	15300	75	0.004901961	0.004901961	20
pimion	10723	10726	15300	-3	-0.000196078	0.000196078	8.5
pimiri	10842	10839	15300	3	0.000196078	0.000196078	8.5
pimvot	9753	9753	15300	0	0.0	0.0	4.0
votbcw	5315	5315	8700	0	0.0	0.0	4.0
voteco	5959	5241	8700	718	0.082528736	0.082528736	38
votgla	6623	5721	8700	902	0.103678161	0.103678161	40
votion	5299	5299	8700	0	0.0	0.0	4.0
votiri	5303	5303	8700	0	0.0	0.0	4.0
votpim	6461	5702	8700	759	0.087241379	0.087241379	39
	R^+ = 835.5		R^-= -36.5		R^0= 28		$\|R\|$= 903.0
	T = min(R^+,R^-)	= 50.5	z = -5.05				

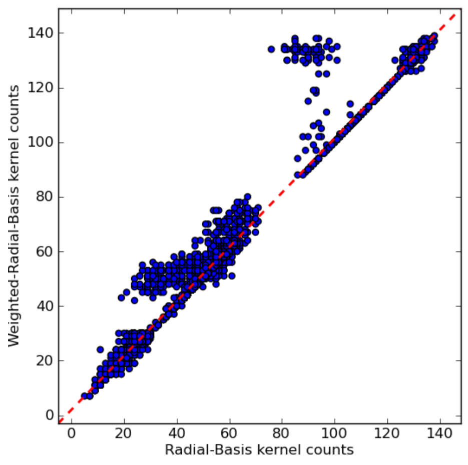
Table A.6: illustrates the ranking done using the Wilcoxon Ranked-Sums test. The high difference between R^+ and R^- suggests that the algorithms are significant different as confirmed by a -5.05 z-score.

	Number of instances: 4200						
Data set	WRBF	RBF	Total	Diff	D/T	|D/T|	Rank
bcweco	13291	13235	13900	56	0.004028777	0.004028777	15
bcwgla	13239	13089	13900	150	0.010791367	0.010791367	26
bcwion	13226	13121	13900	105	0.007553957	0.007553957	17
bcwiri	13285	13151	13900	134	0.009640288	0.009640288	24
bcwpim	13193	13083	13900	110	0.007913669	0.007913669	21
bcwvot	12654	9014	13900	3640	0.261870504	0.261870504	42
ecobcw	5485	4934	6700	551	0.082238806	0.082238806	38
ecogla	5667	5567	6700	100	0.014925373	0.014925373	28
ecoion	5393	4494	6700	899	0.134179104	0.134179104	39
ecoiri	5574	5210	6700	364	0.054328358	0.054328358	34
ecopim	5661	5608	6700	53	0.007910448	0.007910448	20
ecovot	4998	3264	6700	1734	0.25880597	0.25880597	41
glabcw	1767	1664	4200	103	0.02452381	0.02452381	30
glaeco	2361	2324	4200	37	0.008809524	0.008809524	23
glaion	2165	1992	4200	173	0.041190476	0.041190476	33
glairi	2112	1941	4200	171	0.040714286	0.040714286	32
glapim	2464	2315	4200	149	0.03547619	0.03547619	31
glavot	1400	1394	4200	6	0.001428571	0.001428571	14
ionbcw	4345	4345	7000	0	0.0	0.0	6.0
ioneco	4432	4374	7000	58	0.008285714	0.008285714	22
iongla	4423	4415	7000	8	0.001142857	0.001142857	13
ioniri	4434	4434	7000	0	0.0	0.0	6.0
ionpim	4557	4488	7000	69	0.009857143	0.009857143	25
ionvot	4371	4371	7000	0	0.0	0.0	6.0
iribcw	2779	2738	3000	41	0.013666667	0.013666667	27
irieco	2830	2807	3000	23	0.007666667	0.007666667	18.5
irigla	2845	2830	3000	15	0.005	0.005	16
iriion	2845	2782	3000	63	0.021	0.021	29
iripim	2859	2836	3000	23	0.007666667	0.007666667	18.5
irivot	2830	2376	3000	454	0.151333333	0.151333333	40
pimbcw	10576	10572	15300	4	0.000261438	0.000261438	12
pimeco	11260	11260	15300	0	0.0	0.0	6.0
pimgla	11360	11360	15300	0	0.0	0.0	6.0
pimion	10882	10882	15300	0	0.0	0.0	6.0
pimiri	10661	10661	15300	0	0.0	0.0	6.0
pimvot	9890	9890	15300	0	0.0	0.0	6.0
votbcw	5187	5187	8700	0	0.0	0.0	6.0
voteco	6420	5806	8700	614	0.070574713	0.070574713	36
votgla	6794	6174	8700	620	0.071264368	0.071264368	37
votion	5279	5279	8700	0	0.0	0.0	6.0
votiri	5305	5305	8700	0	0.0	0.0	6.0
votpim	6470	5968	8700	502	0.057701149	0.057701149	35
	R^+ = 837		R^-= -0.0		R^0= 66.0		$\|R\|$= 903.0
	T = min(R^+,R^-)	= 33.0	z = -5.28				

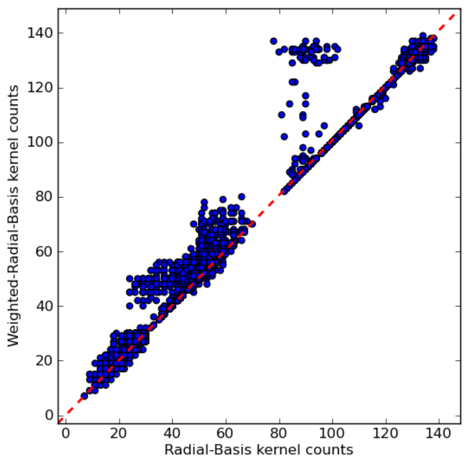
Table A.7: illustrates the ranking done using the Wilcoxon Ranked-Sums test. The high difference between R^+ and R^- suggests that the algorithms are significant different as confirmed by a -5.28 z-score.

	Number of instances: 4198						
Data set	WRBF	RBF	Total	Diff	D/T	|D/T|	Rank
bcweco	13240	13194	13900	46	0.003309353	0.003309353	16
bcwgla	13280	13207	13900	73	0.005251799	0.005251799	21
bcwion	13221	13062	13900	159	0.011438849	0.011438849	25
bcwiri	13273	13340	13900	-67	-0.004820144	0.004820144	20
bcwpim	13190	13102	13900	88	0.006330935	0.006330935	22
bcwvot	11888	9021	13900	2867	0.206258993	0.206258993	41
ecobcw	5245	4595	6700	650	0.097014925	0.097014925	39
ecogla	5578	5333	6700	245	0.036567164	0.036567164	29
ecoion	5464	4904	6700	560	0.08358209	0.08358209	35
ecoiri	5557	4924	6700	633	0.094477612	0.094477612	38
ecopim	5607	5444	6700	163	0.024328358	0.024328358	28
ecovot	4870	3294	6700	1576	0.235223881	0.235223881	42
glabcw	2056	1866	4200	190	0.045238095	0.045238095	31
glaeco	2291	2225	4200	66	0.015714286	0.015714286	26
glaion	1973	1819	4158	154	0.037037037	0.037037037	30
glairi	1943	1716	4200	227	0.054047619	0.054047619	32
glapim	2391	2372	4200	19	0.00452381	0.00452381	19
glavot	1532	1460	4200	72	0.017142857	0.017142857	27
ionbcw	4447	4447	7000	0	0.0	0.0	6.0
ioneco	4419	4408	7000	11	0.001571429	0.001571429	13
iongla	4416	4371	7000	45	0.006428571	0.006428571	23
ioniri	4434	4434	7000	0	0.0	0.0	6.0
ionpim	4514	4451	7000	63	0.009	0.009	24
ionvot	4428	4428	7000	0	0.0	0.0	6.0
iribcw	2858	2695	3000	163	0.054333333	0.054333333	33
irieco	2813	2800	3000	13	0.004333333	0.004333333	18
irigla	2845	2838	3000	7	0.002333333	0.002333333	14
iriion	2840	2849	3000	-9	-0.003	0.003	15
iripim	2821	2810	3000	11	0.003666667	0.003666667	17
irivot	2826	2386	3000	440	0.146666667	0.146666667	40
pimbcw	10853	10853	15300	0	0.0	0.0	6.0
pimeco	11172	11172	15300	0	0.0	0.0	6.0
pimgla	11216	11238	15300	-22	-0.001437908	0.001437908	12
pimion	10685	10685	15300	0	0.0	0.0	6.0
pimiri	10943	10943	15300	0	0.0	0.0	6.0
pimvot	9518	9518	15147	0	0.0	0.0	6.0
votbcw	5253	5253	8700	0	0.0	0.0	6.0
voteco	6247	5515	8700	732	0.084137931	0.084137931	36
votgla	6263	5481	8700	782	0.089885057	0.089885057	37
votion	5248	5248	8700	0	0.0	0.0	6.0
votiri	5247	5247	8700	0	0.0	0.0	6.0
votpim	6459	5860	8700	599	0.068850575	0.068850575	34
	R^+ = 790.0		R^-= -47.0		R^0= 66.0		$\|R\|$= 903.0
	T	=	min(R^+,R^-)	= 80.0	z = -4.68		

Table A.8: illustrates the ranking done using the Wilcoxon Ranked-Sums test. The high difference between R^+ and R^- suggests that the algorithms are significant different as confirmed by a -4.68 z-score.



(a) unsupervised learning type



(b) same class learning type

Figure A.1: illustrates the comparison between the number of correctly classified instances by the RBF (x-axis) and the WRBF (y-axis). The red dashed line represents "equal" performance. All possible data combinations between original and noise suggests that the WRBF has classified the most instances correct, this since most data points are located at the left-side of the equality line. This holds true for both learning algorithms.

Appendix B

Making A Support Vector Machine Using Convex Optimization

B.1 PySVM

One of our stated objectives was to provide a good introduction for students not familiar with Support Vector Machines. The best way to understand a complicated machine is to take it apart and rebuild it. That is exactly the goal of this section. Students not interested in the engineering part can skip this section.

We will construct a small Support Vector Machine in python (*http://www.python.org/*), and use an "out of the box" optimization library (*http://abel.ee.ucla.edu/cvxopt/))* to solve the "dual" in equation (3.17). The following Convex optimization solvers are available in CVXOPT:

```
conelp:   solves linear cone programs.
coneqp:   solves quadratic cone programs.
cp:       solves nonlinear convex problem.
cpl:      solves nonlinear convex problems with linear objectives.
gp:       solves geometric programs.
lp:       solves linear programs.
qp:       solves quadratic programs.
sdp:      solves semidefinite programs.
socp:     solves second-order cone programs.
options:  dictionary with customizable algorithm parameters.
```

The problem that we are presented with is part of the quadratic optimization problems. CVXOPT defines its quadratic solver as:

```
Solves a quadratic program

        minimize    (1/2)*x'*P*x + q'*x
        subject to  G*x <= h
                    A*x = b.
```

Upon comparing the objective function of the qp and the dual we see a strong resemblance. The dual being :
maximize$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$
and qp : `minimize (1/2)*x'*P*x + q'*x`
We need to rewrite the dual in the form of the qp. The qp has a minimizing objective in contrast to the dual. By changing the sign of the dual we can change it to a minimizing function:

$$\text{minimize} \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle - \sum_{i=1}^{l} \alpha_i \tag{B.1}$$

The α's in dual and the `x` in qp are almost identical. We can rewrite the double summation in the dual into the matrix form `P` used in the qp

$$\sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle = x' P x \tag{B.2}$$

We can compute `P` by calculating the dot product for all combinations of patters multiplied by their label.

```
P = Matrix
for $i = 0$ to $numPatterns$ do
  for $j = 0$ to $numPatterns$ do
    P(i,j) = $y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle)\rangle$
  end for
end for
```

The next problem with the objective function is the last term in the dual in equation (B.1). We need to convert $\left[-\sum_{i=1}^{l} \alpha_i\right]$ into $[+q'x]$. This problem is simply solved by choosing `q` to be a vector with solemnly -1 at each index. This leaves us only to rewrite the constraints. The dual has the following constraints $\forall \alpha_j > 0$. From the constraints the `qp` has we only have to use the first $G * x <= h$. Setting G to $-1 \cdot I_n$ [1] and h to a zero vector we have solved the constraints on `qp` in terms of the dual since:

$$G \times x <= h <=> \alpha_j \geq 0 \tag{B.3}$$

B.1.1 C-SVC

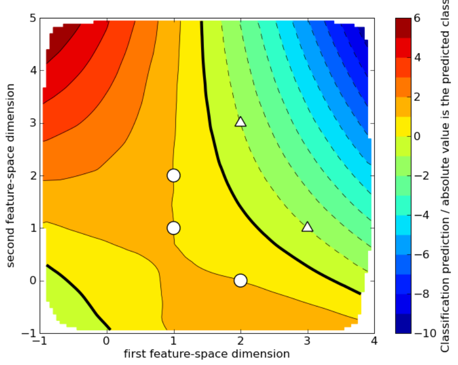
Section (3.4) introduces the soft margin SVM. In order to achieve soft margin optimization using pysvm we need to make two small adjustments. Equation (B.3) describes the current constraints on the support vectors, $\alpha_j \geq 0$. In C-SVC optimization there is an additional constraint: $0 \leq \alpha_j \leq C$. This constraint C on α can be added by adding an additional set of rows to G and h. The rows added for G are simple the identity matrix, whereas h is extended with the $Cost$ parameter. Giving the following constraints to `qp`:

[1] I_n is the $n \times n$ identity matrix

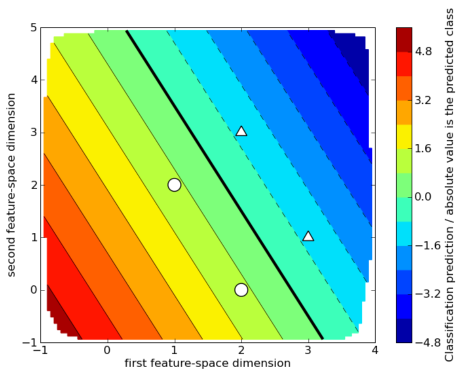
$$G = \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \text{ and } h = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ C \\ \vdots \\ C \end{bmatrix}$$

B.1.2 Examples

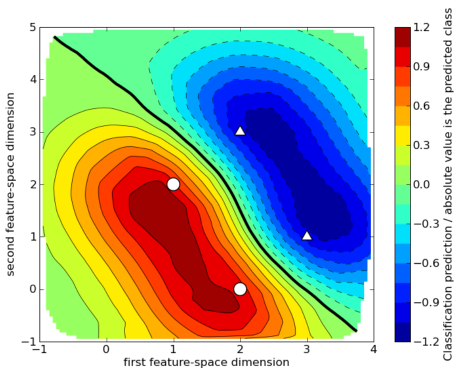
Figure (B.1) illustrates how different types of hypothesis spaces can be constructed using PySVM. Both the Support Vector Machine and Kernel Machine can be found in appendix (C.3) and (C.4)



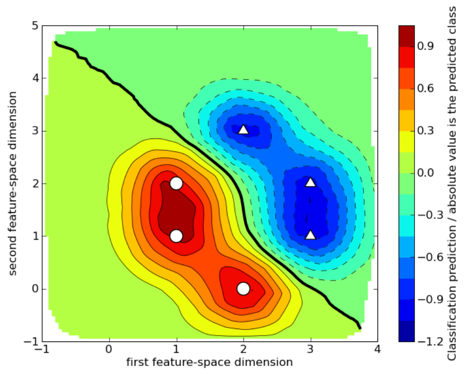
(a) second degree Polynomial Kernel



(b) Linear Kernel



(c) WRBF Kernel



(d) RBF Kernel

Figure B.1: the PySVM Support Vector Machine with different kernel functions.

Appendix C

Source Code Examples

Listing C.1: Implementation of Fixed Increment Single-Sample Perceptron

```
#!/usr/bin/python
from scipy import *
from pylab import *

# M:= Features Y:= Label iw:= start l:= learning rate
M = mat([[1, 2, 1 ],[2, 0, 1 ],[3, 1, 1],[2, 3, 1]])
Y = mat([1, 1, -1, -1])
iw = mat([[-2,-2,2]])
l = 0.1

# Mapping to {1,-1}
def f(z,w):
   if z * w.T > 0:
       return 1
   return -1

# Update rule
def update(label,f,Z):
    return float(l/2.0) * (label - f) * Z

# Itterative algorythm
def fixedIncSinSamPer(M,Y,w,l):
    [rows,cols] = M.shape
    done = False

    while not done:
        done = True
        for idx in xrange(rows):
            dw = update(Y[0,idx],f(M[idx],w),M[idx])
            if dw.any(): # misclassified
                done = False
                w = w + dw
    return w

# Discriminant Functions wich is given by w'x + b = 0
def g(M,w):
    [rows,cols] = M.shape
    for idx in xrange(rows):
        print idx,":", f(w,M[idx]), w*M[idx].T

# Calculate Linear Discriminant:
w = fixedIncSinSamPer(M,Y,iw,l)
g(M,w)
```

Listing C.2: Linear programming for solving linear seperability

```
#!/usr/bin/python
from cvxmod import *

# Datapoints
H = tp( matrix([[0.,1.],[1.,0.],[0.,1.],[0.5,1.],[0,2]]))
K = tp( matrix([[3.,1.],[2.,3.],[1.,3.],[2,3],[2.4,0.8]]))

# Variables to optimize
a = optvar("a",H.size[1])
b = optvar('b')

margin = 0.8

y = optvar("y",1,H.size[0])
z = optvar("z",1,K.size[0])

# Objective
l = problem( minimize(  sum(y) + sum(z)))

# Adding the constraints
l.constr.append( y  >= tp(a) * tp(H) + b + margin)
l.constr.append( z  <= tp(a) * tp(K) + b - margin)

l.constr.append(a[1]==1)

l.constr.append(y>=0)
l.constr.append(z>=0)

l.solve()

# Show g(X)
print value(tp(a) * tp(H) + b)
print value(tp(a) * tp(K) + b)
```

Listing C.3: Support Vector Machine / CVXOPT

```
#!/usr/bin/env python
from cvxopt import *
from cvxopt.solvers import qp
from model import Model

class SupportVectorMachine:

    def __init__(self, kf):
        self.__kernelFunction = kf
        .. ..
    def data(self, labels, features):
        .. ..
    def optimize(self):

        # Initialize Variables
        H = matrix(0.0,(self.__rows,self.__rows))
        G = matrix(0.0,(self.__rows,self.__rows))
        q = matrix(-1.,(self.__rows,1))
        h = matrix(0.,(self.__rows,1))

        # Short notation
        kf = self.__kernelFunction
        y  = self.__labels
        x  = self.__features

        # Compute H
        for idx in xrange(self.__rows):
            for idy in xrange(self.__rows):
                H[idx,idy] = y[idx] * y[idy] * kf(x[idx,:], x[idy,:])

        # Compute G
        for idx in xrange(self.__rows):
            G[idx,idx] = -1.
        # Solve
        self.__solver = qp(H, q, G, h)

    def model(self, km):
        # Filter alpha's
        alpha = list(self.__solver['x'])
        .. ..
        return Model(km,self.__kernelFunction, labels, alphas, svs)
```

Listing C.4: Kernel Machine

```
from cvxopt import matrix
from math import exp
from math import tanh

class KernelMachine:

    def __init__(self):
        self.__linear_const = 1
        self.__params = {}

    def param(self,key,value):
        self.__params[key] = value

    def linear(self,x,y):
        return (x * y.T + self.__params['linear_constant'])[0]

    def sigmoid(self,x,y):
        g = self.__params['gamma']
        k = self.__params['k']
        return ( g * x * y.T - k)[0]

    def polynomial(self,x,y):
        degree = self.__params['degree']
        return ((x * y.T + 1)[0])**degree

    def radialbasis(self,x,y):
        g = self.__params['gamma']
        z = x - y
        return exp(- (z * z.T)[0] / (2 * g**2) )

    def wradialbasis(self,x,y):
        g = self.__params['gamma']
        w = self.__params['kw']
        o = 0.0

        for idx in xrange(len(w)):
            o += w[idx] * ((x[idx]-y[idx])*(x[idx]-y[idx]))

        return exp(-g * o)
```

Bibliography

[1] Marius Bulacu, Rutger van Koert, Lambert Schomaker, and Tijn van der Zant. Layout analysis of handwritten historical documents for searching the archive of the cabinet of the dutch queen. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pages 357–361, Washington, DC, USA, 2007. IEEE Computer Society.

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

[3] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[5] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods).* Cambridge University Press, Cambridge, United Kingdom, 2000.

[6] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[7] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[8] Scott Doniger, Thomas Hofmann, and Miao-Hui Joanne Yeh. Predicting cns permeability of drug molecules: Comparison of neural network and support vector machine algorithms. *Journal of Computational Biology*, 9(6):849, 2002.

[9] R. O. Duda and P. E. Hart. *Pattern Classification.* John Wiley and Sons, 2000.

[10] Robert P.W. Duin. A note on comparing classifiers, 1995.

[11] Simon Haykin. *Neural Networks, A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, New Jersey, 1999.

[12] R. V. Hogg and E. A. Tanis. *Probability and Statistical Inference.* Prentice Hall, 6 edition, 2001.

[13] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2000.

[14] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[15] Edson J.R. Justino, Flvio Bortolozzi, and Robert Sabourin. A comparison of svm and hmm classifiers in the off-line signature verification. *Pattern Recognition Letters*, 26(9):1377 – 1385, 2005.

[16] Erwin Kreyszig. *Advanced Engineering Mathematics, 9th Edition.* John Wiley, December 2005.

[17] Y. LeCun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and H. S. Baird. Handwritten zip code recognition with multilayer networks. In IAPR, editor, *Proc. of the International Conference on Pattern Recognition*, volume II, pages 35–40, Atlantic City, 1990. IEEE. invited paper.

[18] T. Mitchell. *Machine Learning.* McGraw-Hill Education (ISE Editions), October 1997.

[19] David S. Moore, George P. McCabe, and Michael J. Evans. *Introduction to the Practice of Statistics Minitab Manual and Minitab Version 14.* W. H. Freeman & Co., New York, NY, USA, 2005.

[20] John C. Platt. Fast training of support vector machines using sequential minimal optimization. pages 185–208, 1999.

[21] Sergio Rojas-Galeano, Emily Hsieh, Dan Agranoff, Sanjeev Krishna, and Delmiro Fernandez-Reyes. Estimation of relevant variables on high-dimensional biological patterns using iterated weighted kernel functions. *PLoS ONE*, 3:e1806, 03 2008.

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[23] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317–327, 1997.

[24] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[25] V. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, New York, 1982.

[26] Bin Yu and Baozong Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, 26(6):883 – 889, 1993.