Contents

Chapter 1

Introduction

Support Vector Machines (SVMs) are extended methods of machine learning used to classify objects from different classes whose features can be represented as vectors $\mathbf{x}$ in a Hilbert space $\mathcal{H}$. The proposed solution by this classifier is a hyperplane in a transformed space –induced by the use of a kernel function– which has the biggest distance to both classes. By using the Lagrange function with the primal problem, a dual problem can be induced with the Lagrange multipliers as the only features. The training of this classifier through the dual problem leads to a quadratic programming (QP) problem. The resolution method for the quadratic function has been extensively analyzed and it can be handled even for very big problems by the used of SMO[1] .

On the other side, work on evaluation still has to be done to improve the classification time after the training. This work is focused on an improvement of this evaluation time.

If a non-linear kernel was used to solve the problem, normally we would expect to obtain a large number of support vectors depending on the problem complexity. These support vectors are the basis of the classification step. This step consists on the Kernel function (particularly, for the linear kernel, the dot product) of each support vector with the new sample data and then a weighted sum of these results.

One of these machines was applied for the classification of 3D reconstruction of cell nuclei in blood. Normally, an image can contain, for example, 3 125 000 voxels ($250 \times 250 \times 50$), and from each voxel, more than 30 features can be obtained. A trained machine for this problem has between 400 to 1 000 support vectors which makes the classification of a whole new image very slow, since $O(25\,600)$ multiplications are needed to classify 1 voxel. There are some works already done to enhance the SVM evaluation speed like for example:

Direct reduction of number of SVs. Burges and Schölkopf in [BS97] proposed a method tu approximate $\mathbf{w}$ by a $\mathbf{w}'$ which can be also expresed by a list of vectors associated with corresponding coefficients $alpha_i$. However, the method for determining the reduced set is computationally very expensive. Later, Downs, Gates and Masteres [DGM01] a method to identify and discard unnecessary SVs –those SVs who lineary depend on other SVs–

[1] As implemented by Chih-Chung Chang and Chih-Jen Lin in [CL05]

while leaving the SVM decision unchanged. A reduction in SVs as high as 40.96% was reported.

Indirect reduction of number of SVs by reducing the size of the QP problem. This method called *RSVM* (Reduced Support Vector Machines) was proposed in [LM04]. In [LL03] it was shown where this method was suitable.

Reduction of the number of vector components. In [LHL05] a reduction of the feature space is proposed.

The aim of this work is the reduction of the classification time by ***linear approximation*** of the function in the transformed feature space. This approximation is done by a *decision tree* whose nodes are conformed by support vector machines with linear kernels. The advantage of using only linear kernels is that the feature space is known since it is the original one and the classification can be done only by making the single dot product with the vector orthogonal to the hyperplane –which can be easily calculated– and the comparison with the intersection value of the calculated hyperplane, that is, if we have 80 straight lines, we need $O(2560)$ multiplications. An important feature in this approach is that the new classifier generalize nearly as good sa a SVM with Radial Basis Function (RBF) kernel. In many cases the evaluation speed is significantly improved because the number of nodes (evaluating hyperplanes) is considerably smaller than the number of needed support vectors. Another important feature of this new proposed method is that the constrained optimization problem is written in a different way as usual such that no parameters need to be tunned like in the classical problem with the kernel trick.

The Chapter 2 gives an introduction to the necessary basic theory of support vector machines (SVMs), constrained optimization theory, and presents decision trees. The Chapter 3 introduces some theoretical and numerical problems to substantiate the construction of the new classifier, and at the end of the chapter the new proposed algorithm is described. Chapter 4 makes a detailed description of the implementation with the most fundamental changes to previous implementations. Chapter Exp shows the results that were obtained after using this new methodology for many different classification problems.

Several issues arise from this proposed method, these are overview in Chapter **??**. Finally, Chapter **??** presents the Conclusions of the work.

Chapter 2

Background

Consider a machine (or living organism) which receives a set of inputs $\{x_1, x_2, ..., x_m\}$. This input, which will be called the *samples* or *data*, could correspong to an image on the retina, the pixels in a camera, or a sound waveform. It could also correspond to less obviously sensory data, for example the words in a news stoy, of the list of items in a supermarket shopping basket. All this data will be represented in vectors $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\} \in \mathbb{R}^n$.

These objects can be subgrouped (or an output parameter can be assign) accordingly to particular characteristics. The aim is to build a machine able to distinguish to which group belongs each sample (or to assign an appropiate parameter to the input).

2.1 Classification

One can distinguish a particular set of machines that belong to the kind of *supervised learning*. Here, a sequence of outputs $y_1, y_2, ..., y_m$ is also given that represent the corresponding labels to each sample (in classification) or a real number (in regression). The goal of the machine is to learn to produce the correct output given a new input.

Our interest will be focused in the classification theory, particularly, in a method called *Support Vector Machines* (SVMs). This machine has become very popular because, in general, satisfactory results can be obtained as with other machines and it has a robust mathematical background.

The theory developed in this chapter is focused on support vectors machines and decision trees, nevertheless, this approach leads to several other interesting theoretical issues that will be analyzed in later chapters.

2.1.1 Support Vector Machines

The first interest will be focused in a two-class problem with samples $\mathbf{x}_i$ and labels $y_i = \pm 1$ for $i = 1, ..., m$. A standard definition for these two classes will be handled through this document.

In a classification problem, we can start by differentiating two classes, but it is possible to extend a two class classifier to a multi-class problem by using techniques like

One vs. One If there are n classes, $\binom{n}{2}$ two-class classifiers are pairwise trained for this problem. For classification, vectors are tested in all models giving a probability (points) of belonging to a class, finally, it will be labeled as the class that has more points.

One vs. Rest If there are n classes, n two-class classifiers are trained, where one class is differentiated from all the others. New samples are tested in all models and the results are compared.

There are many different methodologies, Multiclass gives a comparison and resume about them.

The following discussion will be centered in a two class problem.

It will be assumed that the set of features of each sample $\mathbf{x}$ belongs to a Hilbert space denoted by $\mathcal{H}$, that is, this is a vector space with a dot product $\langle x, y\rangle$, with $x, y \in \mathcal{H}$ such that a *norm* can be induced by $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x}\rangle}$.

Two-Class SVM

In the next paragraph, a general notation for a two class problem will be described. This will be used along the whole document to make easier the description of the problem and implementation of the classifier.

Definition 2.1 (Positive and Negative Class) *Let m_1 and m_2 be two natural numbers that fulfill $m = m_1 + m_2$, $m_1 > 0$, $m_2 > 0$ and $\mathscr{C} = \{1, ..., m\}$, without loss of generality we can define:*

Class 1 (Positive Class) *of size m_1, with index $\mathscr{C}_1 = \{1, ..., m_1\}$, conformed by the set $\{\mathbf{x}_i\}, i \in \mathscr{C}_1$, gravity center $\mathbf{s}_1 = \frac{1}{m_1}\sum_{i\in\mathscr{C}_1} \mathbf{x}_i$, $y_i = 1$ for all $i \in \mathscr{C}_1$, and for some later applications, a penalization value D_1 is defined and $C_i = D_1$ $\forall i \in \mathscr{C}_1$.*

Class 2 (Negative Class) *of size m_2, with index $\mathscr{C}_2 = \{m_1+1, ..., m_1+m_2\}$, conformed by the set $\{\mathbf{x}_i\}, i \in \mathscr{C}_2$, gravity center $\mathbf{s}_2 = \frac{1}{m_2}\sum_{i\in\mathscr{C}_2} \mathbf{x}_i$, $y_i = -1$ for all $i \in \mathscr{C}_2$, and for some later application, a penalization value D_2 is assigned to this class and $C_i = D_2$ $\forall i \in \mathscr{C}_2$.*

Having two classes, we say that they are *linearly separable* if there is a hyperplane of the form $\mathcal{P} : \{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x}\rangle + b = 0\}$, $\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}$ that can perfectly divide the two classes. The vector $\mathbf{w}$ is a vector orthogonal to the hyperplane P and $\langle \mathbf{w}, \mathbf{x}\rangle$ is the length of $\mathbf{x}$ along the direction of $\mathbf{w}$.

We will be interested in finding the *canonical hyperplane* with respect to $\mathbf{x}_i$, $i \in \mathscr{C}$ defined as the hyperplane with the pair $(\mathbf{w}, b) \in \mathcal{H} \times \mathbb{R}$ if it is scaled such that

$$\min_{i=1,...,m} \|\langle \mathbf{w}, \mathbf{x}\rangle + b\| = 1. \tag{2.1}$$

That is, the *canonical hyperplane* is the one whose minimal distance to the samples equals $\frac{1}{\|\mathbf{w}\|}$.

To illustrate that, let's consider the following two class example depicted in Figure 2.1:

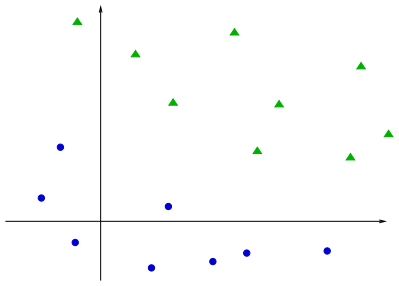

Figure 2.1: Example of a two class problem

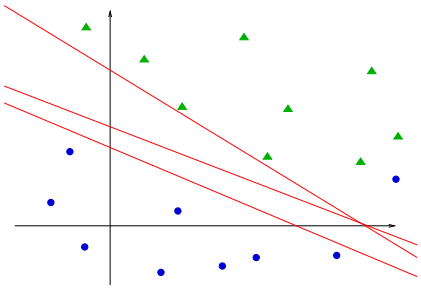

Figure 2.2: Possible dividing hyperplanes for a two class problem

Without loss of generality, let the green triangles represent the class 1 ($\mathscr{C}_1$) and the blue circles represent class 2 ($\mathscr{C}_2$). The following hyperplanes in Figure 2.2 are all valid functions to divide them:

If the objective is to divide the two classes with a plane, we would like to do it with the plane that has maximum distance (margin) to both classes, i.e., a maximum distance between the two classes. In the shown example in Figure 2.1, the desired hyperplane would look as in Figure 2.3.

It has to be noticed that for $\mathbf{x}_i, i \in \mathscr{C}_1$ and $\mathbf{x}_j, j \in \mathscr{C}_2$, such that $\langle \mathbf{w}, \mathbf{x}_i \rangle = +1$ and $\langle \mathbf{w}, \mathbf{x}_j \rangle = -1$, we have $\langle \mathbf{w}, (\mathbf{x}_i - \mathbf{x}_j) \rangle = 2$ and therefore

$$\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, (\mathbf{x}_i - \mathbf{x}_j) \rangle = \frac{2}{\|\mathbf{w}\|} \tag{2.2}$$

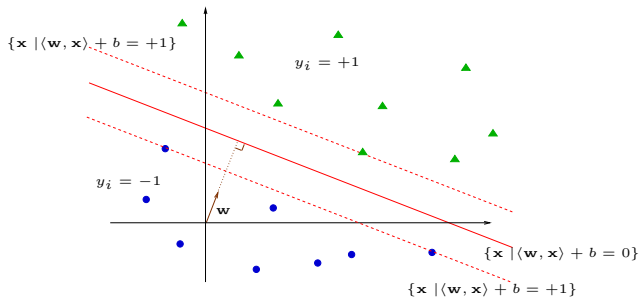

Figure 2.3: Hyperplane with maximal margin for a two class problem

With the previous equation, we can conclude that the distance of the closest vector to the hyperplane is $\frac{1}{\|\mathbf{w}\|}$, then, finding the hyperplane with the maximum distance is equivalent to maximize the norm of the orthogonal vector $\mathbf{w}$ that corresponds to the hyperplane that can divide the two classes.

Constrained Optimization Theory

A general constrained optimization problem is defined as follows [chapter]

Problem 2.2 (Constrained Optimization Problem)

$$\underset{\mathbf{x}\in\mathcal{H}}{\text{minimize}} \quad f(\mathbf{x}), \tag{2.3}$$

$$\text{subject to} \quad c_i(\mathbf{x}) = 0,\ i \in \mathcal{E}, \tag{2.4}$$

$$c_i(\mathbf{x}) \geq 0,\ i \in \mathcal{I} \tag{2.5}$$

where $\mathcal{E}$ comprehends the indexes for the equality constraints and $\mathcal{I}$ are the indices for the inequality constraints.

By analyzing the first-order Taylor series to the objective and constraint functions, some conditions can be derived.

Starting at a feasible point $\mathbf{x}$, if we move in direction $\mathbf{d}$ to retain feasibility with respect to the function $c_i(\mathbf{x}) = 0$, we require that $c_i(\mathbf{x} + \mathbf{d}) = 0$; that is,

$$0 = c_i(\mathbf{x} + \mathbf{d}) \approx c_i(\mathbf{x}) + \nabla c_i(\mathbf{x})^T \mathbf{d} = \nabla c_i(\mathbf{x})^T \mathbf{d}. \tag{2.6}$$

Hence, the direction $\mathbf{d}$ retains feasibility with respect to c_i, in first order, when it satisfies

$$\nabla c_i(\mathbf{x})^T \mathbf{d} = 0. \tag{2.7}$$

Similarly, a direction of improvement must produce a decrease in f, so that

$$0 > f(\mathbf{x} + \mathbf{d}) - f(\mathbf{x}) \approx \nabla f(\mathbf{x})\mathbf{d},$$

or, in firsts order,

$$\nabla f(\mathbf{x})^T \mathbf{d} < 0. \tag{2.8}$$

If there exists a direction $\mathbf{d}$ that satisfies both 2.7 and 2.8, we conclude that improvement on our current point $\mathbf{x}$ is possible. It follows that a *necessary condition* for optimality for the problem 2.2 is that there exist *no direction* $\mathbf{d}$ *satisfying both 2.7 and 2.8.*

The only way that such a direction cannot exist is if $\nabla f(\mathbf{x})$ and $\nabla c_1(\mathbf{x})$ are parallel[1], that is, if the condition $\nabla f(\mathbf{x}) = \alpha_i \nabla c_i(\mathbf{x})$ holds at $\mathbf{x}$, for some scalar α_i. If this condition is *not* satisfied, the direction defined by

$$\mathbf{d} = -\left(I - \frac{\nabla c_i(\mathbf{x}) \nabla c_i(\mathbf{x})^T}{\|\nabla c_i(\mathbf{x}\|^2}\right) \nabla f(x)$$

satisfies both conditions 2.7 and 2.8. By introducing the *Lagrangian function*

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) - \sum_i \alpha_i c_i(\mathbf{x}). \tag{2.9}$$

[1] see [NW99] for a detailed explanation

and noting that $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = \nabla f(\mathbf{x}) - \sum_i \alpha_i \nabla c_i(\mathbf{x})$, we can state the necessary condition as: At the solution $\mathbf{x}^*$, there is a scalar α_i^* such that

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}^*) = 0. \tag{2.10}$$

This observation suggest that we can search for solutions of the constrained problem 2.2 by searching for stationary points of the Lagrangian function. The scalar quantity α_i in 2.9 is called a *Lagrange multiplier* for the constraint $c_i(\mathbf{x}) = 0$.

An inequality is said to be active at point $\mathbf{x}$ if its evaluation at this point reaches the equality. The ***active set*** $\mathcal{A}(\mathbf{x})$ in a constrained problem at any feasible $\mathbf{x}$ is the union of the indices of the equality constrains $\mathcal{E}$ with the indices of the active inequality constraints, that is,

$$\mathcal{A}(\mathbf{x}) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(\mathbf{x}) = 0\} \tag{2.11}$$

Attention has to be given to the properties of the constraint gradients. The vector $\nabla c_i(\mathbf{x})$ is often called the *normal* to the constraint c_i at the point $\mathbf{x}$, since it is usually a vector that is perpendicular to the contours of the constraint c_i at $\mathbf{x}$, and in the case of an equality constraint, it points toward the feasible side of this constraint. However, it can happen that ∇c_i vanishes due to the algebraic representation of c_i, so that the term $\alpha_i \nabla c_i(\mathbf{x})$ equals zero for all values of α_i and thus it plays no role in the Lagrangian gradient $\nabla_{\mathbf{x}}\mathcal{L}$.

Therefore, an assumption called a *constraint qualification* is done to ensure that such degenerate behavior does not occur at the value of $\mathbf{x}$ in question. One constraint qualification is the following:

Definition 2.3 (LICQ) *Given the point $\mathbf{x}^*$ and the active set $\mathcal{A}(\mathbf{x}^*)$ defined by 2.11, we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(\mathbf{x}^*), i \in \mathcal{A}(\mathbf{x}^*)\}$ is linearly independent.*

If the above condition holds, none of the active constraint gradients can be zero.

With this, the optimality conditions for a general nonlinear programming problem concerning the properties of the gradients (first-derivative vector) 2.4 can be provided.

[chapter]

Theorem 2.4 (First-Order-Necessary Conditions (KKT)) *Suppose that $\mathbf{x}^*$ is a local solution of 2.2 and that the LICQ holds at $\mathbf{x}^*$. Then there is a Lagrange multiplier vector $\boldsymbol{\alpha}^*$, with components $\alpha_i, i \in \mathcal{E} \cap \mathcal{I}$, such that the following conditions are satisfied at $(\mathbf{x}^*, \alpha^*)$:*

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}^*) = 0, \tag{2.12}$$
$$c_i(\mathbf{x}^*) = 0, \forall i \in \mathcal{E}, \tag{2.13}$$
$$c_i(\mathbf{x}^*) \geq 0, \forall i \in \mathcal{I}, \tag{2.14}$$
$$\boldsymbol{\alpha}_i^* \geq 0, \forall i \in \mathcal{I}, \tag{2.15}$$
$$\boldsymbol{\alpha}_i^* c_i(\mathbf{x}^*) = 0, \forall i \in \mathcal{E} \cup \mathcal{I}. \tag{2.16}$$

The above conditions are often known as *Karush-Kuhn-Tucker conditions*, or *KKT conditions* for short, and they have to be satisfied in order to find an optimum for problem 2.2.

SVM Constrained Optimization Problem

The following problem, is a formal definition of the maximal margin hyperplane problem that wants to be solved.

Problem 2.5 (SVM-Primal Optimization Problem) *Let a class 1 and a class 2 be defined as in Definition 2.1, the optimal margin hyperplane primal problem is defined as follows*

$$\underset{\mathbf{w}\in\mathcal{H},b\in\mathbb{R}}{\text{minimize}} \qquad \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2, \tag{2.17}$$

$$\text{subject to} \quad y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1, \ i = 1, .., m, \tag{2.18}$$

And the corresponding decision function would look like

$$f(x) = sign(\langle \mathbf{w}, \mathbf{x}\rangle + b) \tag{2.19}$$

In problem 2.5, $f(x) = \tau(\mathbf{w})$, $\mathcal{E} = \emptyset$ and $\mathcal{I} = \mathscr{C}$. Following [NW99] and as in [SS02], the *Lagrangian* function can be defined together with the objective function τ and the constraints in 2.18 as follows

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1). \tag{2.20}$$

One of the KKT conditions states that the gradient of the Lagrangian function must equal zero, that is,

$$\frac{\partial}{\partial b}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \qquad \text{and} \tag{2.21}$$

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = 0. \tag{2.22}$$

this leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \qquad \text{and} \tag{2.23}$$

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i}. \tag{2.24}$$

The solution vector thus has an expansion 2.24 in terms of a subset of the training patterns, namely those patterns with non-zero α_i, called *Support Vectors* (*SVs*). By the KKT conditions,

$$\alpha_i[y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1] = 0 \text{ for all } i = 1, ..., m, \tag{2.25}$$

the SVs lie on the margin. All remaining training examples $(\mathbf{x}, y_i)$ are irrelevant: their constraint $y_j(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1$ could just as well be left out, and they do not appear in the expansion. Thus, the hyperplane is completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting 2.23 and 2.24 into the Lagrangian 2.20, one eliminates the primal variables $\mathbf{w}$ and b, getting the following problem which is in practice solved since it depends only on the sample vectors.

Problem 2.6 (SVM-Dual Optimization Problem) *Let class 1 and class 2 are defined as in Definition 2.1, the optimal margin hyperplane dual problem is defined as follows*

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle, \tag{2.26}$$

$$\text{subject to} \quad \alpha_i \geq 0,\ i = 1, ..., m, \tag{2.27}$$

$$\sum_{i=1}^m \alpha_i y_i = 0. \tag{2.28}$$

Using 2.24, the hyperplane decision function 2.19 can thus be written as

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^m y_i\alpha_i\langle \mathbf{x_i}, \mathbf{x}\rangle + b\right) \tag{2.29}$$

Figure 2.4 shows a solution for the example in Figure 2.1, the yellow area shows the points in the space that will be labeled as class 1 and the cyan area shows the points that will be labeled as class 2. The distance from a point to the dividing hyperplane is shown by the shadow, that is, if a point is closer to the hyperplane, it has a lighter tone of yellow/cyan.

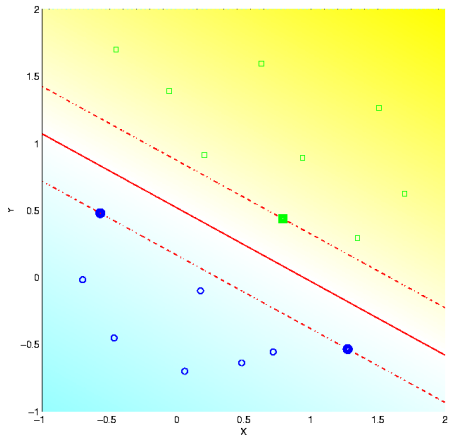

Figure 2.4: Two class classification problem with linear solution

Soft Margin SVM

Often, the problem can be an unfeasible problem because there does not exist any hyperplane that can separate perfectly both classes. For such problems, the C-SV classifier was introduced allowing some mistakes through slack variables with a penalization in the objective function, leading to the following problem:

Problem 2.7 (C-SV Classifier Primal Problem) *For a two class problem, the primal optimization problem with slack variables is defined as:*

$$\underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimize}} \quad \tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^m C_i\xi_i, \tag{2.30}$$

$$\text{subject to} \quad y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1 - \xi_i,\ i = 1, .., m, \tag{2.31}$$

$$\xi_i \geq 0,\ i = 1, .., m. \tag{2.32}$$

The definition here given can varies depending on the source it is taken. Using again 2.23 and 2.24, the last problem can be converted into the following dual problem

Problem 2.8 (C-SV Classifier Dual Problem) *In a two class problem, the optimal margin hyperplane dual problem with slack variables is defined as follows*

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle, \tag{2.33}$$

$$\text{subject to} \quad 0 \le \alpha_i \le C_i,\ i = 1, ..., m, \tag{2.34}$$

$$\sum_{i=1}^m \alpha_i y_i = 0. \tag{2.35}$$

For this problem, the decision function remains as in 2.29. We notice that there is still missing the value of the threshold b, if there exist a solution for problem 2.6, the hyperplane qould be placed in the middle of the two closest points to each class. Nevertheless, for problem 2.8, the value of the α must be taken into account; the calculation of b can be done as proposed in [KSBM99] as following

$$b = \frac{1}{2}\left(\underset{i\in I_0\cup I_1\cup I_2}{min}\{\langle \mathbf{x_i}, \mathbf{w}\rangle\} + \underset{i\in I_0\cup I_3\cup I_3}{max}\{\langle \mathbf{x_i}, \mathbf{w}\rangle\}\right), \tag{2.36}$$

where,

$$I_0 = \{i|0 < \alpha_i < C_i\}, \tag{2.37}$$

$$I_1 = \{i|y_i = 1, \alpha_i = 0\}; \quad I_2 = \{i|y_i = -1, \alpha_i = C_i\}, \tag{2.38}$$

$$I_3 = \{i|y_i = 1, \alpha_i = C_i\}; \quad I_4 = \{i|y_i = -1, \alpha_i = 0\}. \tag{2.39}$$

Non-Linear SVM

If the sample vectors are not linear separable, there exists no hyperplane that can perfectly divide them. To overcome this problem, feature-vectors are mapped into higher dimensional space $mathcal H$ by the use of some non-linear function

$$\Phi(\mathbf{x}) : \mathbb{R}^d \to \mathcal{H} \tag{2.40}$$

Φ is chosen in a way such that the classes can be separated in $\mathcal{H}$ by the trivial SVM decision function.

The linear case was developed in a Hilbert space $\mathcal{H}$. In order to make generalizations of this method, the dot product $\langle \mathbf{x}, \mathbf{x}'\rangle$ can be expressed in terms of the kernel k evaluated on input patterns x, x' in a transformed space induced by $\Phi(\mathbf{x}) = x$,

$$k(\mathbf{x}, \mathbf{x}') = \langle x, x'\rangle = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}')\rangle. \tag{2.41}$$

This substitution, which is referred to as the ***kernel trick***, is used to extend the method to transformed spaces with nonlinear Support Vector Machines in a new space $\mathcal{H}$ called the *linearization space* because in the new space, the samples are divided with an hyperplane (i.e. a linear function).

The kernel trick can be applied since all feature vectors in 2.19 and 2.26 only occurred in dot products. The vector $\mathbf{w}$ then becomes an expansion in feature

space, and therefore will typically no longer correspond to the $\Phi - image$ of a single input space vector. We obtain a decision function of the form

$$f(x) = sign\left(\sum_{i=1}^{m} y_i \alpha_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i)\rangle + b\right) \tag{2.42}$$

$$= sign\left(\sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\right) \tag{2.43}$$

with the threshold b calculated similarly as in 2.36, but considering that now we have applied Φ to the original samples,

$$b = \frac{1}{2}\left(\min_{i \in I_0 \cup I_1 \cup I_2} \{k(\mathbf{x_i}, \mathbf{w})\} + \max_{i \in I_0 \cup I_3 \cup I_3} \{k(\mathbf{x_i}, \mathbf{w})\}\right), \tag{2.44}$$

where the index I_k are defined as in 2.37, 2.38 and 2.39.

The following quadratic problem is the one formulated with the kernel trick

Problem 2.9 (C-SV Kernel Trick in Classifier) *In a two class problem, the optimal margin hyperplane dual problem in the transformed Hilbert space induced by $k(\mathbf{x}, \mathbf{x}')$ (with slack variables) is defined as follows*

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{2.45}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C_i, \ i = 1, ..., m, \tag{2.46}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{2.47}$$

To illustrate how this trick is used, we'll consider a one-dimensional example that looks like in Figure 2.5. This has no linear solution.

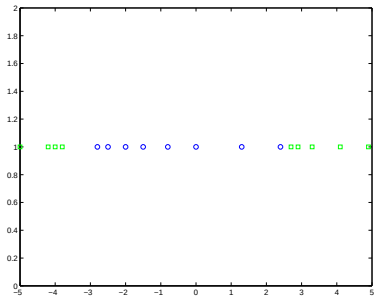

Figure 2.5: One-dimensional two class problem with no linear solution

By making a simple quadratic transformation: $x_i \rightarrow (x_i, x_i^2)$, the samples can be represented in a bigger space (in this case 2D-space) as in Figure 2.6.

In this new space, a linear solution *does* exist and looks like in Figure 2.7.

This leads to a generalization of the optimization problem by changing the *kernel* $k(\mathbf{x}_i, \mathbf{x}_j)$, and in means of calculation, this is reduced to stored this ordered pairs in a kernel matrix with the dot product in the transformed space, since the last one can be calculated in the original one.

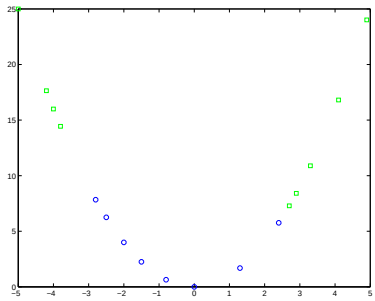

Figure 2.6: Quadratic transformation

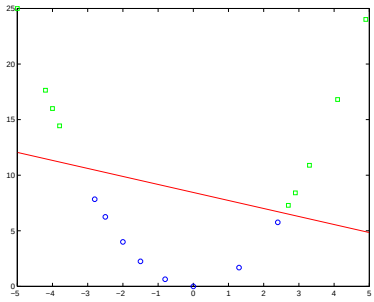

Figure 2.7: Dividing hyperplane in the new space

Kernel Design

Starting from a finite space $X = \{x_1, x_2, ...x_r\}$, a kernel-function is represented by a symmetric $r \times r$ matrix which holds $K_{ij} = k(x_i, x_j)$.

Using eigenvalue decomposition K can be rewritten as[2] $K = UVU^T$ with the $r \times n$ matrix $U = (\mathbf{u}_1^T), \mathbf{u}_2^T, ..., \mathbf{u}_r^T$, satisfying $UU^T = I_n$ and $V = diag(\lambda_1, ..., \lambda_n)$, $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n \geq 0$.

The kernel function $\Phi : X \to k \subseteq \mathbb{R}^n$ is now expressed as

$$\Phi(\mathbf{x}_i) = V^{\frac{1}{2}} \tag{2.48}$$

$$mathbfu_i \tag{2.49}$$

This results in the Gram-Matrix:

$$G_{ij} = (\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j))_K = V^{\frac{1}{2}} \tag{2.50}$$

$$mathbfu_i V^{\frac{1}{2}} \tag{2.51}$$

$$mathbfu_j = \mathbf{u}_i^T V \mathbf{u}_j = K_{ij} \tag{2.52}$$

this shows that in fact, the only restriction on kernels is that theis eigenvalues have to satisfy $\lambda \geq 0$. Thus, K has to be *semi positive definite*. In general, a K is a kernel iff K holds $\sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$.

This is stated by Mercer theorem:

[2] where here U^T denotes the transpose of U

Theorem 2.10 (Mercer Theorem) *$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ iff for arbritrary $g(\mathbf{x})$ with $\int g(\mathbf{x})^2 d\mathbf{x} < \infty$ holds:*

$$\int K(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_i) g(\mathbf{x}_j) d\mathbf{x}_i d\mathbf{x}_j \geq 0. \tag{2.53}$$

If $\mathbf{x} \in \mathbb{R}^n$ there were several proposed kernels, which details can be read in [SS02]:

Homogeneous

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle \tag{2.54}$$

Polynomial

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^d \tag{2.55}$$

Gaussian

$$k(\mathbf{x}, \mathbf{x}') = exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2} \right) \tag{2.56}$$

Sigmoid

$$k(\mathbf{x}, \mathbf{x}') = tanh(\kappa \langle \mathbf{x}, \mathbf{x}' \rangle + \vartheta) \tag{2.57}$$

All these kernel functions are transformation of the original space to another one of maybe different dimension (possibly infinite), where a linear solution exist. The classifying function can still be represented in the original space, however, this will not be anymore an hyperplane, but a function with a curvature.

The Gaussian function, also known as Radial Basis Function (RBF) is normally the first choice because it combines good performance with strong theoretical foundation. In [SS02] it is proven that the RBF-kernel is equivalent to the dot product of elements belonging to an infinite demensional space. To show the capacity of this trick, let us illustrate it by adding some more negative samples to the Figure 2.1, as it is shown in Figure 2.8. As it can be seen, there does not exist any hyperplane that can perfectly classify all training samples.

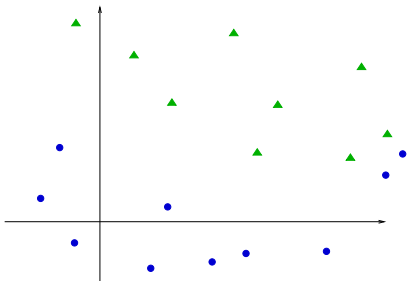

Figure 2.8: Example of a two class problem with no linear solution

If we try to adjust a straight line to separate this problem, the result seen in Figure 2.9 would be obtained.

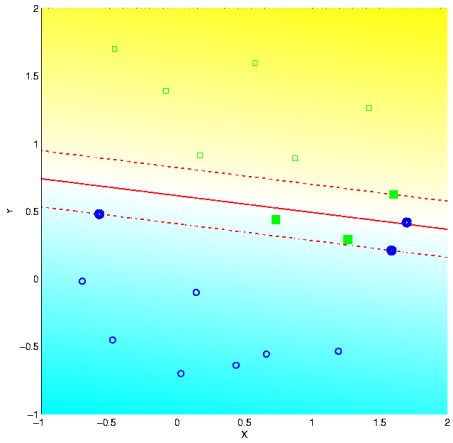

Figure 2.9: Two class problem with the best adjusted hyperplane

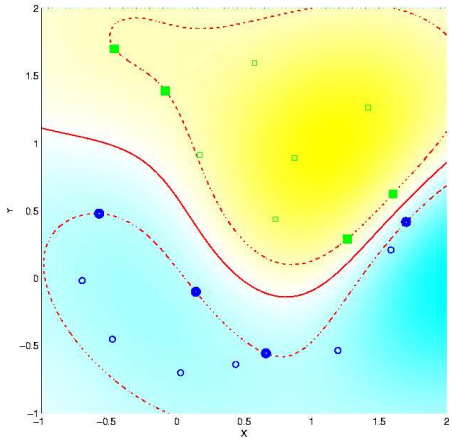

Figure 2.10: Solution for a two class problem with Gaussian Kernel

The solution to the problem in Figure 2.8 with a Gaussian kernel would look like in Figure 2.10.

The samples that are filled out represent the support vectors. The solid red line represents the decision hyperplane and the margin between both classes is marked with the spoted red lines.

2.1.2 Data Preprocessing

In general the data cannot be used as it was gotten originally. It can happen that the values of one vector dimension differs significantly in magnitude with other dimension. This can mislead the classifier causing that the training of the machine is basically based in the dimension with bigger values. Therefore, some prepsocessing is needed before feeding the data to the machine.

Normalizing the Data

Normalization is a process of scaling the numbers in a data set to improve the accuracy of the subsequent numeric computations. For a sample $\mathbf{x}_{ij}$ in a set $\{\mathbf{x}_i\}$, with $\mathbf{x}_i \in \mathbb{R}^n$, there are several ways of doing this:

Standard deviation . The dataset is center at zero mean and scaled it to have unit standard deviation,

$$\mathbf{x}_{ij} = \frac{\mathbf{x}_{ij} - \mu_{\mathbf{x}_i}}{\sigma_{\mathbf{x}_i}} \tag{2.58}$$

where $\mu_{\mathbf{x}_i}$ and $\sigma_{\mathbf{x}_i}$ is the mean and the standard deviation of the vector $\mathbf{x}_i$.

Min-max . The dataset is linearly scaled to a specific minimum min^* and maximum max^* value,

$$\mathbf{x}_{ij} = \frac{max^* - min^*}{max\{\mathbf{x}_i\} - min\{\mathbf{x}_i\}}(\mathbf{x}_{ij} - min\{\mathbf{x}_i\}) + min^* \tag{2.59}$$

2.1.3 Model Selection

The results of a classifier depend on the training samples and a set of training parameters. In the case of the SVMs, these parameters are the cost C_i for outliers and parameters like γ in the Gaussian or the degree in the polynomial kernel.

Since these parameters can be changed to adjust the model, the task of finding the best model is called *model selection*.

The first concern is how to measure a model. In an intuitive way, this question is easy to answer: a good model always gives a correct classification result even with samples that have never been seen. This ability is called the *generalization* ability. If the training error is driven to a very small value, but when the new data is presented to the network the error is large, then the model has *overfitting*. A theoretical approach to the generalization ability was introduced with the Vapnik-Chervonenkis dimension (VC dimension) [VC79] as a measure of it.

2.2 Decision Trees

Chapter 3

Support Vector Machines in a Decision Tree

Support vector machines are used to classify feature vectors. The usual choice is the RBF kernel since in terms of generalization capacity, when a RBF kernel is used to train a SVM, the results can be very satisfactory, this method was run with several databases which contents differs from each other substantially[1]. More over, the training of this machine can be done with reasonable computer time, with the SMO technique.

Unfortunately, as explained before, the classification time depends on the number of resulting support vectors. It can occur that many of the training sample vectors contribute to the representation of the function in the original space. Several examples where run, and it was observed that this set was relative big in respect to the number of training samples.

3.1 Theoretical Approach

In this thesis a linear approximation of a continuous function to classify is proposed. The space will be divided in regions defined by linear inequalities (hyperplanes). This gives many advantages, one of these is that the transformed space is known since it is the original one; the tuning of parameters can be avoided and the number of hyperplanes needed to linearly approximate the classification function is far less than the number of needed support vectors.

With this aim, a decision tree was built. Each node corresponds to a hyperplane which can classify a specific region.

3.1.1 Zero Solution in SVM

Usually, a SVM is trained so that it will make the least possible mistakes in both classes. With the classical approach, the importance of errors in each class can be tuned by adjusting the values of D_1 and D_2 which are the weights for class $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively, according to Definition 2.1.

[1]For details about the classification capacity in these exampels, some data is resumed in the section of experiments

So, each hyperplane in the tree is trained in dependence of the previous hyperplane, to be able to identify a region of the feature space where only samples of one class lie. To achieve this, a first constrained problem is solved in order to found the SVM that classifies perfectly one chosen class (say class $\mathscr{C}_1$) and make the least errors in the other.

This would lead to propose a big value for D_1 and a very small value for D_2. A problem that can be faced is that with this proposal, the optimal solution can be the zero solution is D_1 and if the center of gravity if class $\mathscr{C}2$ lies in the convex hull of class $\mathscr{C}_1$ is enough big as is proved in the following theorem.

Theorem 3.1 (Zero Solution) *Let class 1 and class 2 be defined as in Definition 2.5. If the convex hull of one class (say, class $\mathscr{C}_k$) intersects the convex hull of the other (say, class $\mathscr{C}_{\bar{k}}$), then $\mathbf{w} = \mathbf{0}$ is a feasible solution for the primal problem 2.5 if $D_2 \geq \max_{i\in\mathscr{C}_1}\{\lambda_i\} \cdot D_1$, where λ_i are such that:*

$$\mathbf{p}_1 = \sum_{i\in\mathscr{C}_1} \lambda_i \mathbf{x}_i$$

for a point $\mathbf{p}_1$ that belongs to both convex hulls.

Proof

It has to be noticed that, if $i \in \mathscr{C}_k$, $j \in \mathscr{C}_{\bar{k}} \Rightarrow y_i \cdot y_j = -1$ and $i \in \mathscr{C}_k$, $j \in \mathscr{C}_k \Rightarrow y_i \cdot y_j = 1$, then the dual problem can be written as follows

$$\begin{array}{ll} \text{maximize} & \sum_{i\in\mathscr{C}_1} \alpha_i + \sum_{i\in\mathscr{C}_2} \alpha_i + \sum_{i\in\mathscr{C}_1, j\in\mathscr{C}_2} \alpha_i\alpha_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle \\ & -\frac{1}{2}\sum_{i,j\in\mathscr{C}_1} \alpha_i\alpha_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle - \frac{1}{2}\sum_{i,j\in\mathscr{C}_2} \alpha_i\alpha_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle \\ \\ \text{subject to} & \sum_{i\in\mathscr{C}_1} \alpha_i y_i + \sum_{i\in\mathscr{C}_2} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq D_1 \text{ for all } i \in \mathscr{C}_1 \\ & 0 \leq \alpha_j \leq D_2 \text{ for all } j \in \mathscr{C}_2 \end{array}$$

If $\mathbf{p}_1$ belongs to the convex hull of both classes, then, it can be written as follows

$$\mathbf{p}_1 = \sum_{i\in\mathscr{C}_1} \lambda_i \mathbf{x}_i \quad \text{and} \quad \mathbf{p}_1 = \sum_{j\in\mathscr{C}_2} \lambda_j \mathbf{x}_j$$

with $\lambda_i \geq 0$ for all $i \in \mathscr{C}_1$, $\sum_{i\in\mathscr{C}_1} \lambda_i = 1$ and $\lambda_j \geq 0$ for all $j \in \mathscr{C}_2$, $\sum_{j\in\mathscr{C}_2} \lambda_j = 1$.

Let $\alpha_i = \lambda_i D_1 \leq D_1$ for all $i \in \mathscr{C}_1$ and $\alpha_j = \lambda_j D_1 \leq \max_{j\in\mathscr{C}_1}\{\lambda_j\} D_1 \leq D_2$ for all $j \in \mathscr{C}_2$, then

$$\begin{array}{rcl} \sum_{i\in\mathscr{C}_1} \alpha_i y_i + \sum_{j\in\mathscr{C}_2} \alpha_j y_j & = & \sum_{i\in\mathscr{C}_1} \lambda_i D_1 - \sum_{j\in\mathscr{C}_2} \lambda_j D_1 \\ & = & D_1 \sum_{i\in\mathscr{C}_1} \lambda_i - D_1 \sum_{j\in\mathscr{C}_2} \lambda_j \\ & = & D_1 - D_1 \\ & = & 0 \end{array}$$

Therefore $\alpha_i = \lambda_i D_1$ for all $i \in \mathscr{C}_1$ and $\alpha_j = \lambda_j D_1$ for all $j \in \mathscr{C}_2$ is a feasible solution for the dual problem.

If we calculate the vector $\mathbf{w}$ with these values, we obtain:

$$\begin{aligned}
\mathbf{w} &= \textstyle\sum_{i\in\mathscr{C}_1} \alpha_i \mathbf{x}_i y_i + \sum_{j\in\mathscr{C}_2} \alpha_j \mathbf{x}_j y_j \\
&= \textstyle\sum_{i\in\mathscr{C}_1} \lambda_i D_1 \mathbf{x}_i - \sum_{j\in\mathscr{C}_2} \lambda_j D_1 \mathbf{x}_j \\
&= D_1 \textstyle\sum_{i\in\mathscr{C}_1} \lambda_i \mathbf{x}_i - D_1 \sum_{j\in\mathscr{C}_2} \lambda_j \mathbf{x}_j \\
&= D_1 \mathbf{p}_1 - D_1 \mathbf{p}_1 \\
&= \mathbf{0}
\end{aligned}$$

and we conclude that $\mathbf{w} = \mathbf{0}$ is a feasible solution for the primal problem 2.5.

□

In [BCSTW00], Bennett and Bredensteiner proved that problem 2.7 has also another dual problem that can be seen in terms of the convex hull. When a solution exist, a geometric interpretation of problem 2.5 can be reduced to the problem of finding the closest points of the two convex hulls; then, constructing the line segment between the two points. The plane, orthogonal to the line segment that bisects the line segment, is chosen to be the separating plane.

Under this approach, problem 2.7 leads to the geometric interpretation of finding the closest points of the two reduced convex hulls according to D_1 and D_2. This reduced convex hull will still be around the center of gravity of the original convex hull. And in the case that the center of gravity of class 2 is inside the convex hull of class 1, if D_1 is big enough, the zero solution will still be a feasible solution no matter what value D_2 has. This is resumed in the following Corollary.

Corollary 3.2 *[Zero Solution with Gravity Center] If the center of gravity of class 2, $\mathbf{s}_2$, is inside of the convex hull of class 1, it can be represented as*

$$\mathbf{s}_2 = \sum_{i\in\mathscr{C}_1} \lambda_i \mathbf{x}_i \quad \textit{and} \quad \mathbf{s}_2 = \sum_{j\in\mathscr{C}_2} \frac{1}{m_2} \mathbf{x}_j$$

with $\lambda_i \geq 0$ for all $i \in \mathscr{C}_1$ and $\sum_{i\in\mathscr{C}_1} \lambda_i = 1$.

If additionally $D_1 \geq \lambda_{max} D_2 m_2$, where $\lambda_{max} = \max_{i\in\mathscr{C}_1}\{\lambda_i\}$, then $\mathbf{w} = \mathbf{0}$ is a feasible solution for the primal problem 2.5,

Proof

Let class 1 and class 2 be as in Definition 2.1, then the dual problem can be written as follows

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{i\in\mathscr{C}_1} \alpha_i + \sum_{i\in\mathscr{C}_2} \alpha_i + \sum_{i\in\mathscr{C}_1, j\in\mathscr{C}_2} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\
& -\tfrac{1}{2} \textstyle\sum_{i,j\in\mathscr{C}_1} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \tfrac{1}{2} \sum_{i,j\in\mathscr{C}_2} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\
\\
\text{subject to} \quad & \textstyle\sum_{i\in\mathscr{C}_1} \alpha_i y_i + \sum_{i\in\mathscr{C}_2} \alpha_i y_i = 0 \\
& 0 \leq \alpha_i \leq D_1 \text{ for all } i \in \mathscr{C}_1 \\
& 0 \leq \alpha_j \leq D_2 \text{ for all } j \in \mathscr{C}_2
\end{aligned}$$

Let $\alpha_i = \lambda_i D_2 m_2 \leq \lambda_{max} D_2 m_2 \leq D_1$ for all $i \in \mathscr{C}_1$ and $\alpha_j = D_2$ for all $j \in \mathscr{C}_2$, then,

$$\begin{aligned}\sum_{i\in\mathscr{C}_1}\alpha_i y_i + \sum_{j\in\mathscr{C}_2}\alpha_j y_j &= \sum_{i\in\mathscr{C}_1}\lambda_i D_2 m_2 - \sum_{j\in\mathscr{C}_2} D_2 \\ &= D_2 m_2 \sum_{i\in\mathscr{C}_1}\lambda_i - D_2 m_2 \\ &= D_2 m_2 - D_2 m_2 \\ &= 0\end{aligned}$$

Therefore $\alpha_i = \lambda_i D_2 m_2$ for all $i \in \mathscr{C}_1$ and $\alpha_j = D_2$ for all $j \in \mathscr{C}_2$ is a feasible solution for the dual problem.
If we calculate the vector $\mathbf{w}$ with these values, we obtain:

$$\begin{aligned}\mathbf{w} &= \sum_{i\in\mathscr{C}_1}\alpha_i \mathbf{x}_i y_i + \sum_{j\in\mathscr{C}_2}\alpha_j \mathbf{x}_j y_j \\ &= \sum_{i\in\mathscr{C}_1}\lambda_i D_2 m_2 \mathbf{x}_i - \sum_{j\in\mathscr{C}_2} D_2 \mathbf{x}_j \\ &= D_2 m_2 \sum_{i\in\mathscr{C}_1}\lambda_i \mathbf{x}_i - D_2 m_2 \mathbf{s}_2 \\ &= D_2 m_2 \mathbf{s}_2 - D_2 m_2 \mathbf{s}_2 \\ &= \mathbf{0}\end{aligned}$$

and we conclude that $\mathbf{w} = \mathbf{0}$ is a feasible solution for the primal problem 2.5.

□

Any reduced convex hull produced by a C-SV problem definition[2] will still contain the gravity center of the class. Therefore, for the case where the gravity center of one class is in the convex hull of the other class and vice-versa, if a feasible solution for the C-SV problem wants to be found, the convex hull of both classes has to be reduced enough, giving some mistake in both classes during the training. Numerically, the kernel matrix is not always positive definite, therefore, the trivial solution ($\mathbf{w} = \mathbf{0}$) can be found with other values of $\alpha \neq 0$.

3.1.2 Reduction of Possibility of the Zero Solution

In order to reduce the classification time, the used support vector machine with non-linear kernel will be substitute with a decision tree of linear support vector machines. The tree will first target an area in the feature space that can be clearly assigned to class $\bar{k}, \bar{k} = 2, 1$ by a linear classifier. This will be achieved by finding the hyperplane with the widest margin that made no mistakes in class $k, k = 1, 2$ and that makes the least possible mistakes in class $\bar{k}$.

For this, and to decrease the number of trivial solutions that are in the approach, the following new problem will be introduced:

Problem 3.3 (Hard Margin for 1 class (Primal Problem)) *Let 2 classes be defined as in Definition 2.1, we will be interested on solving the following problem:*

$$\underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_{i\in\mathscr{C}_k} y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b), \tag{3.1}$$

$$\text{subject to} \qquad y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1 \text{ for all } i \in \mathscr{C}_k, \tag{3.2}$$

where $k = 1$ and $\bar{k} = 2$, or $k = 2$ and $\bar{k} = 1$.

[2] for more details on this type of SVM, please see [SS02], [Cri] and [BCSTW00]

Analyzing more precisely this problem, it can be seen that the feasible solution of this optimization problem is that one that classifies correctly all the samples in class k (because $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)$) for all $i \in \mathscr{C}_k$ is a condition with no slack variables. On the other side, from all the vectors that satisfy this condition, the search vector is the one that has a balance between the size of the margin and the number of misclassified samples of class $\mathscr{C}_\mathbf{k}$. As before, this problem can be transformed into the following dual problem which is a special case of the original problem where all the α_k for one class are equal to zero.

Problem 3.4 (Hard Margin for one class (Dual Problem)) *Let 2 classes be defined as in Definition 2.1, we will be interested on solving the following problem:*

$$\begin{aligned}
\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \textstyle\sum_{i\in\mathscr{C}_k} \alpha_i - \sum_{i\in\mathscr{C}_k, j\in\mathscr{C}_{\bar{k}}} \alpha_i y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle && (3.3)\\
& -\tfrac{1}{2}\textstyle\sum_{i,j\in\mathscr{C}_{\bar{k}}} y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \tfrac{1}{2}\sum_{i,j\in\mathscr{C}_k} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, && (3.4)\\
\text{subject to} \quad & 0 \le \alpha_i, \ i \in \mathscr{C}_k && (3.5)\\
& \text{and } \textstyle\sum_{i\in\mathscr{C}_k} \alpha_i y_i + \sum_{i\in\mathscr{C}_{\bar{k}}} y_i = 0. && (3.6)
\end{aligned}$$

Or

$$\begin{aligned}
\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, && (3.7)\\
\text{subject to} \quad & 0 \le \alpha_i \le C_i, \ i \in \mathscr{C}_k && (3.8)\\
& \alpha_j = 1, \ j \in \mathscr{C}_{\bar{k}} && (3.9)\\
& \textstyle\sum_{i=1}^m \alpha_i y_i = 0. && (3.10)
\end{aligned}$$

where $k = 1$ and $\bar{k} = 2$, or $k = 2$ and $\bar{k} = 1$.

3.2 Description of the Algorithm

The aim is to build a tree which nodes are SVMs. At each step, a region defined by a hyperplane is labeled with a class until the whole space is labeled. To illustrate this and the further description of the algorithm, let us consider the example in Figure 3.1.

This example can be found at the web-page of the LIBSVM[3] c++ library under the name of *Fourclass*. The problem has 2 features and therefore it can be represented in 2D. Class 1 is represented with green triangles and class 2 is represented with blue circles. This example has clearly a non-linear solution, so a SVM with Gaussian Kernel was used. The graphical representation of the found solution is depicted in Figure 3.2.

The classification function corresponding to the found hyperplane in the transformed space is marked with a solid red line, the existing margin between the two classes can be seen with the spotted red lines. The thicker points are the needed support vectors for the classification. As it can be seen, these are a big percent of the training data, therefore a large evaluation time for evaluating new points is needed.

[3]Which Internet address is http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

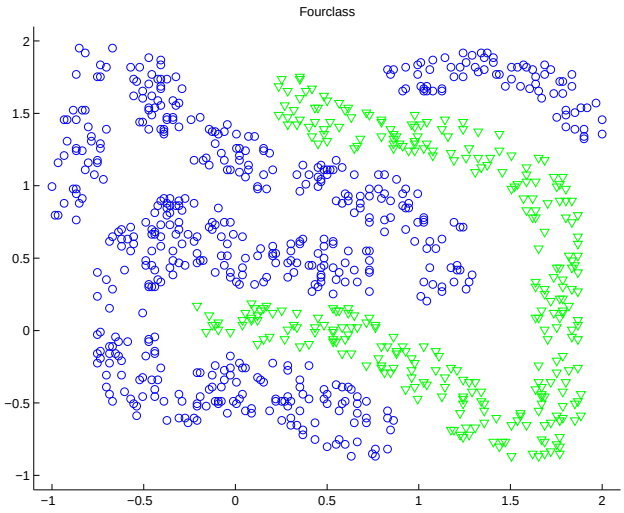

Figure 3.1: Fourclass example

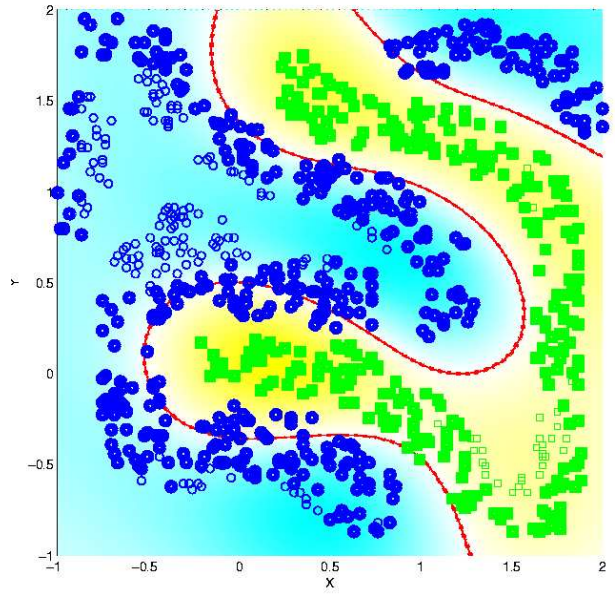

Figure 3.2: Solution for the fourclass with a SVM (Gaussian kernel)

This solution can be trained in order to get a perfect classification in both classes accordingly with the sample vectors. Due to limitations in the matlab package, it cannot be obtained with this tool.

3.2.1 Decision Tree with Linear SVM Nodes

The speedup of the classification of a dataset is done by the construction of a decision tree which nodes are hyperplanes obtained with the training of a support vector machine with linear kernel.

To obtain the decision tree, at each step a ***hard class*** is chosen –say class 1– then a SVM is trained so that the resulting hyperplane will correctly distinguish all points belonging to class 1, that is, all the samples with label $y_i = 1$ will lie on one side of the hyperplane and all points on the other side of the plane will be labeled as class 2. The number of samples is then reduced by leaving out the training samples of class 2 that were correctly classified with this SVM. This process is repeated with the reduced problem until the samples left, belong all to the same class.

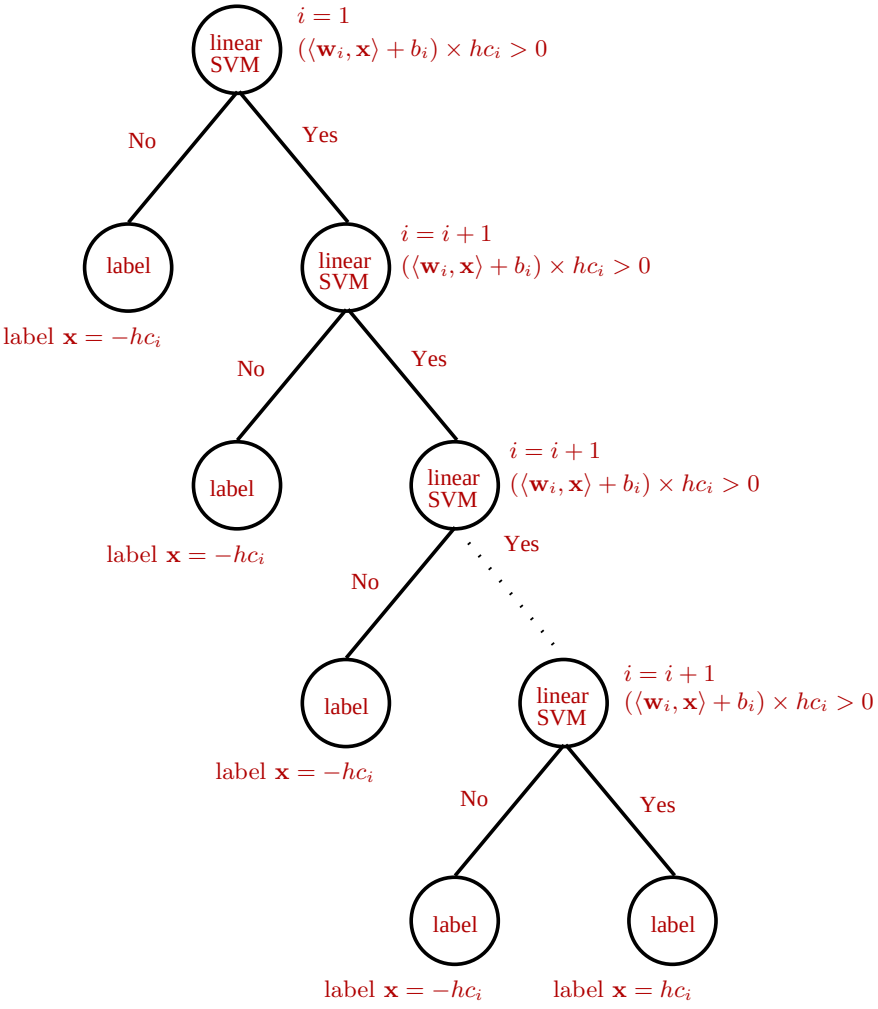

Figure 3.3: Decision tree with linear SVM

The classification then takes places by identifying at each node if the sample belongs to the non-hard class 2 being labeled with it, or keeping with the

evaluation to the next node. This is depicted in the diagram 3.3.

In the fourclass example, the class 1 (green triangles) is the hard class at the first step, the line (hyperplane) obtained by solving problem 3.4 with hard class 1 will look like in Figure 3.4.

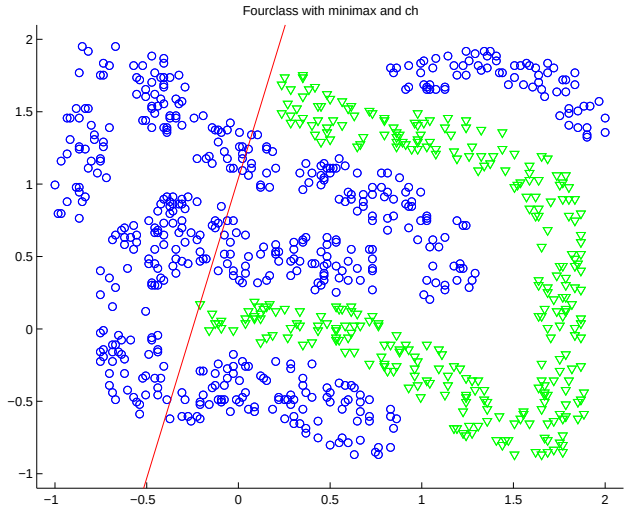

Figure 3.4: First hyperplane for problem 3.4 for *fourclass* ($\mathscr{C}_1$ = hard class)

On Figure 3.5 the region which has exclusively samples belonging to the non-hard class is depicted with a shadow in cyan and it all will be labeled as class 2 with different scale (accordingly to the shadow). The darkness indicates that is is more probable fora sample in that area to belong to class 2.

Next, the problem is reduced by leaving out the samples that lie in the previously marked region. For the here analyzed *fourclass* example, the new problem to solve is the one in Figure 3.6.

This procedure is stepwise repeated with the new sample-space marking the "safe" areas (i.e. areas where samples of only 1 class was found) as non-hard class.

The previous Figure shows which hyperplane found the algorithm at each step of the tree. A region labeled with a cyan shadows represents the solution of a QP with the positive class (green triangles) as the hard class, that is, all the elements in the cyan region are labeled as negative samples. Every time that a hyperplane was chosen, the samples belonging the the non-hard class were removed and the QP for the rest was solve. The space can be step-wise labeled by considering the region that is on the side of the hyperplane where only samples belonging to the non-hard class were found.

At each step, the algorithm choose the plane that can reduce the most the problem, therefore, it can happen that the same class is chosen as the hard class for consecutive nodes in the decision tree.

By repeating this procedure, new regions are labeled until the left samples belong all to the same class. The algorithm will label the whole left region containing these samples with the class they belong.

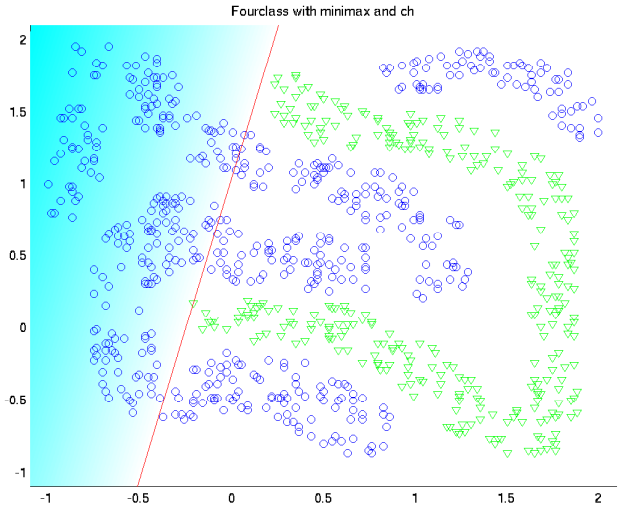

Figure 3.5: First labeling after resolution of problem 3.4 for *fourclass*

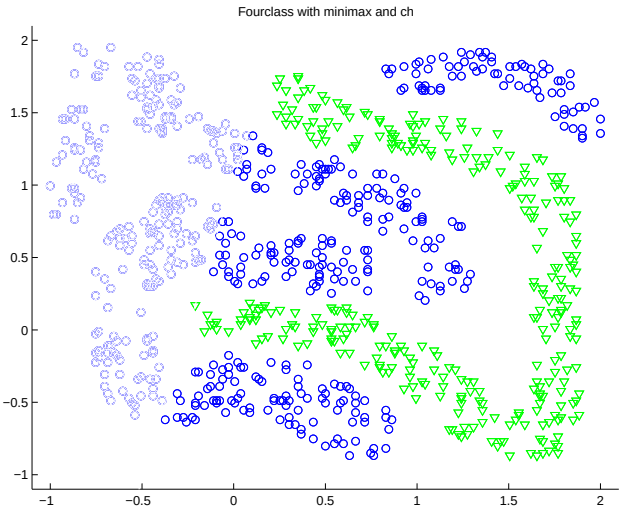

Figure 3.6: Reduced problem for next classification step

Picture 3.7 depicts the final solution of the algorithm for the fourclass problem and how the space was divided according to the decision tree.

3.2.2 Search of the Best Hyperplane

As seen in the theoretical approach, two problems are consider. One is the original approach given by Vapnik, and the second one is the new approach

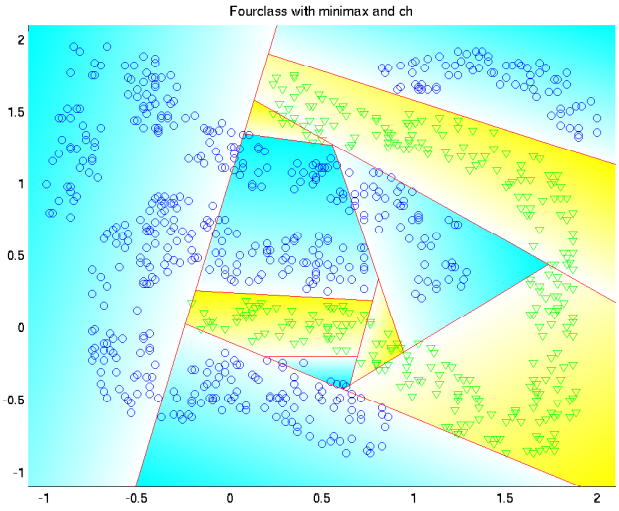

Figure 3.7: Final solution for the fourclass problem

where a hard class is defined and the objective is to find a hyperplane with the maximum margin and the least possible mistakes on the non-hard class.

As in [SS02], both problems can be reduced to a quadratic constrained problem and since there are several existing methods to solve it efficiently, an optimal hyperplane can be found.

A general sequential quadratic programming (QP) problem can be set out for problem 2.8 and also applied for problem 3.4, which has the form:

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \quad \tfrac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \tag{3.11}$$

$$\text{subject to} \quad L_i \leq \alpha_i \leq C_i \tag{3.12}$$

$$\mathbf{y}^T\boldsymbol{\alpha} = 0, \tag{3.13}$$

where $\mathbf{e}$ is the vector of all ones, $C_i > 0$ is the upper bound, Q is an $m \times m$ positive semidefinite matrix, $Q_{ij} = y_i y_j k(x_i, x_j) = \langle \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j) \rangle$ is the kernel. As explain in the previous chapter, vector $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function Φ. The algorithm used to solve this quadratic problem is basically the same to the one developed and implemented by Chih-Chung Chang and Chih-Jen Lin in [CL05], with some adjustments so that this new algorithm can solve QP problems with general bounds.

This QP problem involves a matrix Q that has a number of elements equal to the square of the number of training samples m. If there are many training samples (more than 5000, for example), it can occur that Q will not be able to fit in the memory. Thus, the QP problem becomes intractable via standard QP techniques. Several techniques were developed to have a better approach to this problem, like in [OFG97] a decomposition algorithm that is guaranteed to solve the QP problem is presented. A special case of this algorithm is the Sequential Minimal Optimization (SMO), presented in [Pla99], where subproblems with only two variables are solved.

After the resolution of this problem, the direction of the orthogonal vector is found. The rest is to find the threshold b of this hyperplane, equivalent to the intersection with the axes of it.

It has to be taken into consideration that the searched hyperplane is one that makes no mistakes in the hard class and makes the less possible mistakes in the other class. The usual way of calculating the threshold in 2.36, cannot be longer used for define the hyperplane. Instead, the threshold is calculated by assigning to b_1 the minimum (maximum) value of $\langle \mathbf{w}, \mathbf{x}_i \rangle$ for all $i \in \mathscr{C}_1$ ($i \in \mathscr{C}_2$) and then, for those samples belonging to the non-hard class that are correctly classified, the maximum (minimum) value of $\langle \mathbf{w}, \mathbf{x}_i \rangle$ for all $i \in \mathscr{C}_2$ ($i \in \mathscr{C}_1$) is assigned to b_2 and the threshold is set to $b = \frac{1}{2}(b_1 + b_2)$. That is,

for hard class = 1

$$b = \frac{min_{i \in \mathscr{C}_1} \langle \mathbf{w}, \mathbf{x}_i \rangle + max_{\{j \in \mathscr{C}_2 \wedge \langle \mathbf{w}, \mathbf{x}_j \rangle < 0\}} \langle \mathbf{w}, \mathbf{x}_j \rangle}{2} \tag{3.14}$$

for hard class = -1

$$b = \frac{min_{j \in \mathscr{C}_2} \langle \mathbf{w}, \mathbf{x}_j \rangle + max_{\{i \in \mathscr{C}_1 \wedge \langle \mathbf{w}, \mathbf{x}_i \rangle < 0\}} \langle \mathbf{w}, \mathbf{x}_i \rangle}{2} \tag{3.15}$$

In Figure 3.8 the calculation of the threshold is depicted. For this example, the hard class is the positive class (the green triangles). An orthogonal vector $\mathbf{w}$ is given, the green hyperplane es the one with b_1 as threshold; this has the characteristic that all samples in class 1 are correctly classified, except for the ones such that $\langle \mathbf{w}, \mathbf{x_i} \rangle = b_1$, $i \in \mathscr{C}_1$ and from this threshold, the nearest hyperplane is search such that the least possible errors in class 2 in sone, represented with the blue hyperplane with threshold b_2. Finally, b is calculated as the average of these two values.

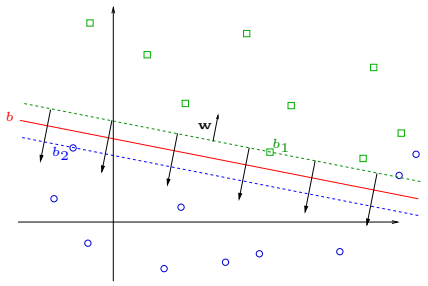

Figure 3.8: Search of threshold b for problem in Figure 2.8

The proposed solution for this problem can be seen in Figure 3.9, as usual, the yellow area represent the positive class and the cyan area represents the negative class with the assigned probability accordingly to the hyperplane that is classifying that area.

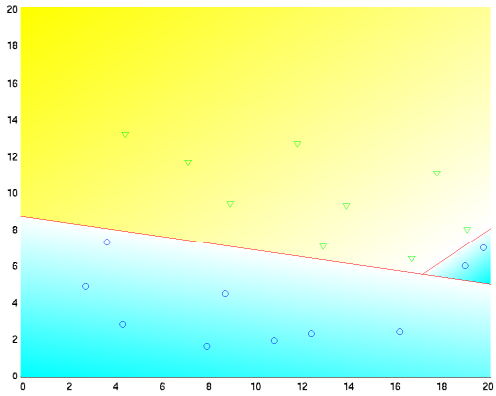

Figure 3.9: Final solution for problem in Figure 2.8

Chapter 4

Implementation Details

This chapter summarize specific details about the implemented algorithms.

The construction of the decision tree is straightforward, therefore, the main interest is to explain the resolution of the optimization problem and the obtaining of the hyperplane in each node of the tree.

Section 4.1 deals with the QP problem and its resolution. Section 4.2 is focused in obtaining the parameters to define the hyperplane, $\mathbf{w}$ and b. Section 4.3 shows the additional heuristics that were used in order to find always a hyperplane that can detach some samples in each iteration.

4.1 Quadratic Problem

Each node in the decision tree is formed by a hyperplane. Each of these hyperplanes can be defined with a vector $\mathbf{w}$ that is orthogonal to the hyperplane and an offset b. The problem of finding the hyperplane with maximum margin and no errors in one class leads to an optimization Problem (2.7 or 3.3) which dual Problem (2.8 or 3.4) results to be a QP problem with the form:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & f(\boldsymbol{\alpha}) = \tfrac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \quad , \\ \text{subject to} \quad & L_i \leq \alpha_i \leq C_i, \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0. \end{aligned} \tag{4.1}$$

If the problem has m samples, Q is a $m \times m$ matrix containing the kernel function applied to each pair of samples, that is, $Q_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$; $\mathbf{e}$ is the vector of ones with lenght m; L_i and C_i are the lower and upper bound, respectively, for α_i. Finally, $\mathbf{y} = (y_1, ..., y_m)^T$ is the vector containing the labels for samples $\mathbf{x}_i$.

Finding an optimum in the dual space, is equivalent to finding an optimum in the primal space. With the help of the KKT conditions, the vector $\mathbf{w}$ can be later calculated using

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{4.2}$$

while the bias b can be fixed accordingly to the stated problem.

4.1.1 Sequential Minimal Optimization (SMO)

The difficulty of solving the previous problem, is the density of Q because Q_{ij} is in general not zero. The methods proposed by [OFG97], [Pla99] and [Joa98] modifies only a subset of $\boldsymbol{\alpha}$ per iteration. This subset –denoted as the working set B– leads to a small sub-problem to be minimized in each iteration. The SMO method as used in [FCL05] is a particular (extreme) case of it, where the subset is restricted to have size 2. Then, in each iteration a simple two-variable problem is solved.

With the original problem, where $L_i = 0$, several simplifications to the resolution method (solver) could be done to speed it up. Therefore, some small adjustements had to be done to the code of the LIBSVM in order to solve problems with $L_i \neq 0$. This algorithm can be found in [CL05] under the name of ***Algorithm 1***.

At each iteration, SMO identifies a pair $\{\alpha_i, \alpha_j\}$ to solve a subproblem with only two variables. This pair is chosen such that it leads to the maximum decrease in the objective function. The algorithm to select the working set can be consulted in [FCL05]. The new subproblem is the one defined in [CL05] like follows

Problem 4.1 (Two-variable QP subproblem) *For iteration t, if $B = \{i, j\}$ and $Q_{ii} + Q_{jj} - 2Q_{ij} > 0$, then the following problem has to be solved*

$$\begin{aligned} \underset{\boldsymbol{\alpha}_B^t = \{\alpha_i, \alpha_j\}}{\text{minimize}} \quad & \tfrac{1}{2}[\alpha_i \;\; \alpha_j] \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (-\mathbf{e}_B + Q_{BN}\boldsymbol{\alpha}_N^t)^T \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix}, \\ \text{subject to} \quad & L_i \leq \alpha_i \leq C_i, \\ & L_j \leq \alpha_j \leq C_j, \\ & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^t, \end{aligned}$$

where $B = i, j$, $N = \{1, ..., m\} \setminus B$, $\boldsymbol{\alpha}_B^t$ and $\boldsymbol{\alpha}_N^t$ are the subvectors of $\boldsymbol{\alpha}$ at iteration t corresponding to B and N respectively; $\mathbf{y}_N$ is the subvector of $\mathbf{y}$ (vector containing the labels) corresponding to N; $Q_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$; $\mathbf{e}_B$ is the vector of ones of size $|B|$ and $\Delta = \mathbf{y}^T\boldsymbol{\alpha}$.

For the resolution of the two variable problem, the next algorithm was implemented. This consist only on slights variations of the one implemented by Chang and Lin [CL05], now it is considered the lower bound different to zero.

Algorithm 4.2 (SMO-iteration solution) *The following algorithm, developed and implemented by Fan [FCL05] and Chang [CL05] in the LIBSVM library was also used in this project in order to update the values of $\alpha_i and \alpha_j$ in the QP problem,*

```
Q[i][j] = kernel evaluation of sample i and sample j
y[i] = label for sample x[i]
alpha = array of size m
G[i] = i-th element of the gradient of the objective function
L_i = lower bound for alpha[i]
C_i = upper bound for alpha[i]

if(y[i]!=y[j]) {
    delta = (-G[i]-G[j])/max{Q[i][i]+Q[j][j]+2*Q[i][j],0}
```

```
    diff = alpha[i] - alpha[j]
    alpha[i] += delta
    alpha[j] += delta

    if(diff > 0) {
        if(alpha[j] < L_j) {
            alpha[j] = L_j
            alpha[i] = diff + L_j
       }
    }
else {
        if(alpha[i] < L_i) {
            alpha[i] = L_i
            alpha[j] = -diff + L_i
       }
    }
    if(diff > C_i - C_j) {
        if(alpha[i] > C_i) {
            alpha[i] = C_i
            alpha[j] = C_i - diff
       }
    }
    else {
        if(alpha[j] > C_j) {
            alpha[j] = C_j
            alpha[i] = C_j + diff
        }
    }
}
else {
    delta = (G[i]-G[j])/max{Q[i][i]+Q[j][j]-2*Q[i][j],0}
    sum = alpha[i] + alpha[j]
    alpha[i] -= delta
    alpha[j] += delta
    if(sum > C_i) {
        if(alpha[i] > C_i) {
        alpha[i] = C_i
        alpha[j] = sum - C_i
    }
}
else {
        if(alpha[j] < L_j) {
            alpha[j] = L_j
            alpha[i] = sum - L_j
        }
    }
if(sum > C_j) {
        if(alpha[j] > C_j) {
            alpha[j] = C_j
            alpha[i] = sum - C_j
```

```
        }
    }
    else {
        if(alpha[i] < L_i) {
            alpha[i] = L_i
            alpha[j] = sum - L_i
        }
    }
}
```

4.1.2 QP Speeding up Techniques

Two more techniques were used to improve the resolution time of the QP problem.

The first one, ***shrinking***, was proposed in [Joa98]. This technique is used because for many problems the number of free vectors (i.e. where $L_i < \alpha_i < C_i$) is small. The shrinking technique reduces the size of the working problem without considering some bounded variables, and near to the end of the iterative process, the possible set A, where all final free α_i may reside in, is identified.

The other method used to reduce the computational time is the ***caching***, that is, the elements of Q_{ij} are calculated as needed since Q is fully dense and may not be stored in the computer memory, but only the recently used Q_{ij} are stored. Hence, the computational cost of later iterations can be reduced.

This two methods were not modified from the original version in the LIBSVM library and are explained in detail in [Joa98] and [CL05].

4.2 Construction of the Decision Hyperplane

The aim of the quadratic programming is to go back to the primal Problem 2.7, and obtain the orthogonal vector and the bias b for the corresponding node in the decision tree. With this, a decision function with the form $sign(\langle \mathbf{w}, \mathbf{x} \rangle + b)$ can be used, since the found hyperplane is in the original space.

4.2.1 Orthogonal Vector

If the hard Class is denoted as $\mathscr{C}_k$ and the non-hard Class is denoted as $\mathscr{C}_{\bar{k}}$. The two following problems are solved with $(k = 1, \bar{k} = 2)$ and later with $(k = 2, \bar{k} = 1)$.

The first problem solved is based directly on Problem 2.7, where a very large cost is given to the errors on the hard class and a standard low cost is given to the non-hard class, in this way, the hyperplane will avoid misclassifications in the hard class:

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1,000,000,000,000, \ i \in \mathscr{C}_k \\ & 0 \leq \alpha_j \leq 1, \ j \in \mathscr{C}_{\bar{k}} \\ & \textstyle\sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{4.3}$$

The initial solution for this problem is set to $\alpha_i = 0$ for all $i \in \mathscr{C}$.

The next problem is based on the new approach proposed in Problem 3.4. This problem is explicitly formulated so that the feasible solutions are the ones that does not allow any misclassifications in the hard class (in numerical terms, this is equivalent to assign a very large cost on the erros in the hard class) and the hyperplane is then adjusted to do the least possible mistakes on the other class. The quadratic problem takes the next form:

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle, \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1,000,000,000,000,\ i \in \mathscr{C}_k \\ & \alpha_j = 1,\ j \in \mathscr{C}_{\bar{k}} \\ & \textstyle\sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{4.4}$$

The initial solution for this problem is $\alpha_i = 0$ for all $i \in \mathscr{C}_k$ and $\alpha_j = 1$ for all $j \in \mathscr{C}_{\bar{k}}$.

The solution of these problems will give the values for $\boldsymbol{\alpha}$ on the optimal point. To obtain the orthogonal vector $\mathbf{w}$, the KKT Condition 2.24 is used:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i}.$$

After solving these problems, four possible solutions for $\mathbf{w}$ are obtained, two are the solutions of Problem 4.3 after solving the problem with $k = 1$ and then for $k = 2$ and the other two are the results obtained from Problem 4.4, again after solving the problem for $k = 1$ and then for $k = 2$.

With one of these hyperplanes, the problem will be reduced by removing the samples of class $\bar{k}$ that were correctly classified with the hyperplane. The solution that can reduce the problem more is finally assigned to the next node on the decision tree with the appropiate threshold b.

New hyperplanes are searched and placed in the succesive child nodes.

4.2.2 Threshold

The resolution of the QP problem proposes as solution an orthogonal vector $\mathbf{w}$ to a hyperplane, there is still missing the calculation of the threshold b, which as mentioned before, is usually calculated as in Equation 2.36:

$$b = \frac{1}{2}\left(\underset{i\in I_0\cup I_1\cup I_2}{min} \{\langle \mathbf{x_i}, \mathbf{w}\rangle\} + \underset{i\in I_0\cup I_3\cup I_3}{max} \{\langle \mathbf{x_i}, \mathbf{w}\rangle\} \right), \tag{4.5}$$

with $I_0 = \{i|0 < \alpha_i < C_i\}$; $I_1 = \{i|y_i = 1, \alpha_i = 0\}$; $I_2 = \{i|y_i = -1, \alpha_i = C_i\}$; $I_3 = \{i|y_i = 1, \alpha_i = C_i\}$; $I_4 = \{i|y_i = -1, \alpha_i = 0\}$.

This cannot be used for the new implementation since this is calculated considering a margin of error in **both** classes [KSBM99] and [KG02]. Our aim is to find such a threshold, that when it is used to classify, all the samples in the actual hard class remain correctly classified and the least possible from the samples belonging to the non-hard class are misclassified. The following algorithm was done to calculate b:

Algorithm 4.3 (Pseudocode for Calculation of threshold) *Code used to calculate the threshold b for the problem in Equations 4.3 or 4.4*

```
m = number of samples
n = feature space size
x[i] = feature vector of sample i
y[i] = label of sample i
hc = hard class (1 or -1)
w = orthogonal vector to the found hyperplane

ub = INF
lb = -INF

for  i=0 to m {
    if ((y[i])*hc > 0) {
        yG = x[i]' * w

        if(y[i] > 0)
            ub = min{ub,yG}
        else
            lb = max{lb,yG}
    }
}

if (ub != INF)
    r1 = ub
else
    r1 = lb

ub = INF
lb = -INF

for  i=0 to m  {
    if ((y[i])*hc < 0) {
        yG = (x[i]' * w) - r1

        if ( (y[i]*yG) > 0) {
            yG = yG + r1
            if(y[i] > 0)
                ub = min{ub,yG}
            else
                lb = max{lb,yG}
        }
    }
}

if (ub != INF)
    r2 = ub
else if (lb !=-INF)
    r2 = lb
```

```
else
    r2 = r1

r=(r1+r2)/2

return r
```

4.3 Heuristics Used

Even though that the optimization function is a quadratic function, numeric problems, speeding up techniques and semi-positive definite matrix can mislead the algorithm so that a global optimum can not always be found. Several heuristics were implemented to assure the convergence of the tree.

4.3.1 Greedy Technique

At each step, Problem 2.8 and Problem 3.4 are solved for both cases: making class 1 as the hard class and also making class 2 as the hard class. The number of vectors that can be left out by using each of these 4 solutions is counted and the assigned hyperplane to the next child node is the one that can reduced more the problem. These made the algorithm a greedy algorithm.

4.3.2 Avoiding the Zero Solution

As seen in Corollary 3.2, the zero solution can outcome if the cost of the hard class is bigger enough than the cost of the non-hard class. One method to avoid obtaining trivial solutions, the upper bound for the alphas in the hard class (C_k), is reduced until a solution different to the trivial one is found, that is, if $\|\mathbf{w}\| < tol$, then C_k is adjusted as follows:

$$C_k = C_k/10, \tag{4.6}$$

where *tol* is a number close to zero that states the tolerance for the norm of vector $\mathbf{w}$.

The other method to avercome the problem of the degenerated solution, is the new approach of the QP problem as in Problem 3.4.

4.3.3 Change of Sign of w

In several cases, the found hyperplane in not able to reduce the problem. In this case, if none sample could be left out, then, $-\mathbf{w}$ is used instead. This is equivalent to change the inequality sense for the classification. This is illustred in Figure 4.1

4.3.4 Perpendicular Hyperplanes

In several cases, the found hyperplane is not able to reduce the problem. It was observed that, nerverless, the hyperplane was oriented in the direction of the

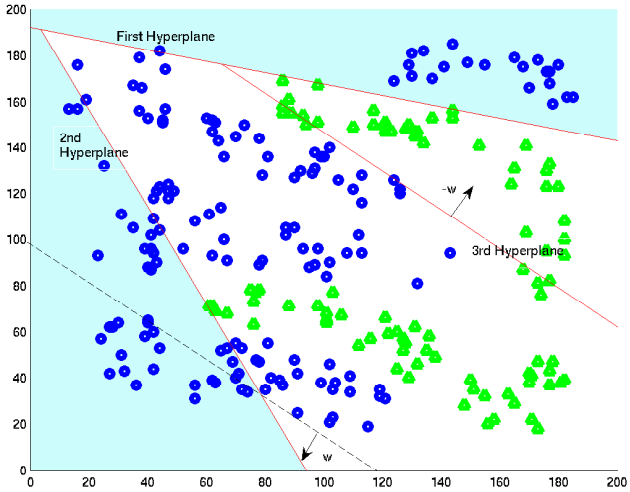

Figure 4.1: Change of sign of $\mathbf{w}$. The direction of vector $\mathbf{w}$ points towards the positive class. The third hyperplane (dashed line) can not classify correctly any sample in the non-hard class (positive class, represented with triangles). If the inequality is changed, this hyperplane can reduce the problem for the next iteration.

distribution of the samples, and some of the orthogonal hyperplanes could do a successful classification.

The use of these perpendicular hyperplanes in the decision tree –in a greedy way– increased the classification rate and the generalization ability. Experimentally, it was observed that this heuristic was not frequently used. The algorithm implemented it, only to change the morphology of the problem to go further.

An additionally degree of greedy technique was implemented with this heuristic. This consist on having the option of considering also the orthogonal hyperplanes together with the original hyperplane that results from the QP problem. Again the chosen hyperplane is the one that can reduced more the problem.

This heuristic is illustrated in Figure 4.2.

4.3.5 Reduction of Usless Hyperplanes (Pruning)

The algorithm stops building the tree after all the samples have been left out (that is, at the moment that the left samples belong all to the same class). At the end of the algorithm, several hyperplanes are of no use in the classification since later hyperplanes were more general than these. This means that if some hyperplanes deeper in the decision tree is used before, some others could be left aside and then reduce the classification time. Figure 4.3 shows an example of this case.

An algorithm was implemented to "clean" the set $\{\mathbf{w}_i\}$, where each $\mathbf{w}_i$ corresponds to the node i in the decision tree. Each hyperplane was eliminated from the decision tree and if no more errors were generated than with the original

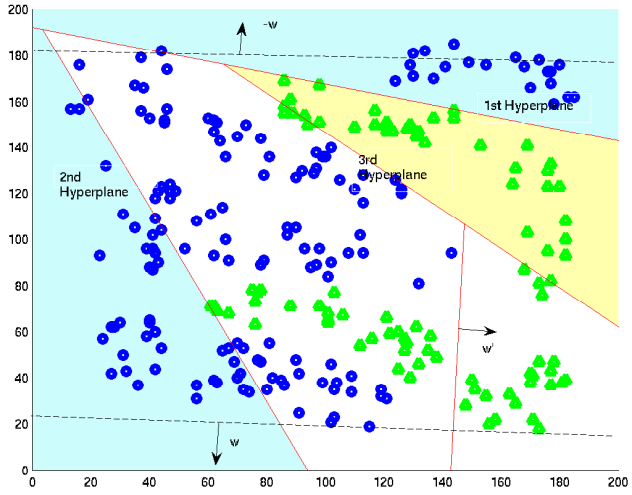

Figure 4.2: Searching perpendicular hyperplanes to $\mathbf{w}$. The direction of vector $\mathbf{w}$ points towards the positive class. The fourth hyperplane (dashed line) can not classify correctly any sample (both equalities) in the non-hard class (positive class, represented with triangles). Instead, a perpendicular hyperplane (with $\mathbf{w}'$) is used.

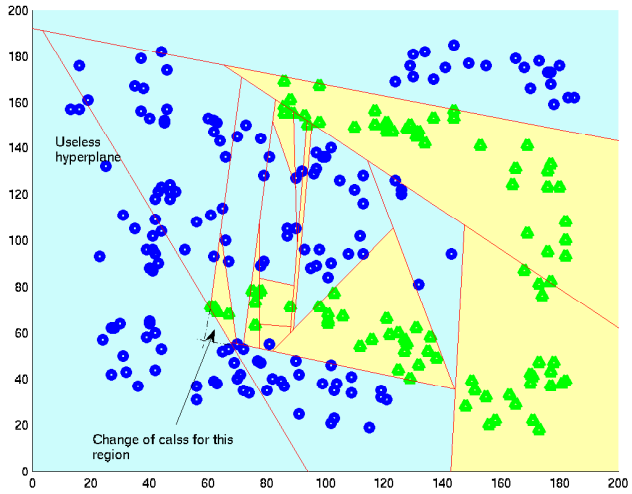

Figure 4.3: Prunning, useless hyperplanes are removed from the tree.

tree, the node with this hyperplane was eliminated.

```
w = two-dimensional array, w[i] contains the hyperplane in node i
rho[i] = threshold for node i
```

```
hard_class_vector[i] = hard class for node i
reached_errors = number of errors with the original decision
                 plane, without reduction
x[j] = sample j
y[j] = label for sample j
Classify(w, rho, hard_class_vector, x) = function that classifies
    sample x with the decision tree form with the w, rho and
    hard_class_vector elements
function erase(i) = erase element i of the vector
function insert(i,obj) = insert obj at position i

w_temp= w[0]
rho_temp = 0
hcv_temp = 0

for (int i=w.size - 1; i>=0 ; i--) {
    errors = 0

    w_temp = w[i]
    rho_temp = rho[i]
    hcv_temp = hard_class_vector[i]

    w.erase(i);
    rho.erase(i);
    hard_class_vector.erase(i);

    for (int j=0; j<prob->l ; j++) {
      x_class = Classify(w,rho,hard_class_vector,x[j]);
      if ( y[j]*x_class <=0 )
          errors++
}
    if ( errors > (reached_errors) ) {
        w.insert(i,w_temp);
        rho.insert(i,rho_temp);
        hard_class_vector.insert(i,hcv_temp);
    }
    if (w.size()==1)
        break
}
```

For the *fourclass* example, at the end of the construction of the tree, the following hyperplanes were taken with the corresponding heuristics:

Line Number	Solver used	Hard Class	Sign of **w**	Use of a perpendicular hyperplane
1	c	-1	1	no
2	h	1	1	yes
3	h	1	1	no
4	h	1	-1	yes
5	h	1	1	no
6	h	-1	-1	yes
7	c	-1	1	no
8	h	1	1	no
9	h	1	1	no
10	h	-1	-1	no
11	h	-1	1	no
12	h	-1	1	no

Chapter 5

Experiments and Comparisons

5.1 Description of the Datasets

5.2 Results and Comparisons

The datasets were trained and classified with a RBF-Kernel SVM and with the new implemented method. Additionally, two different normalizations methods were used (using the standard deviation and also the minimax method), already implemented in the LIBSVMtl library.

It has to be mentioned that the results obtained with the SVM with RBF-Kernel were not tunned and the results were obtained with the standard values. The aim of this section is mainly to compare the number or SVs against the number of require hyperplanes in the new method together with the training and classification time.

DNA (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	180	180	180		
Nr. Train Samples	1330	1330	1330		
Nr. SVs or Hyperplanes	881	3	3	293.67	293.67
Training Time	00:02.62	00:02.11	00:03.82	1.24	0.69
Nr. Test Samples	1446	1446	1446		
Training accuracy	1351	1315	1315	1.03	1.03
Classification Time	00:06.78	00:01.85	00:01.71	***3.66***	***3.96***
Train correctness %	93.43 %	90.94 %	90.94 %	1.03	1.03

DNA (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	180	180	180		
Nr. Train Samples	1330	1330	1330		
Nr. SVs or Hyperplanes	798	3	3	266	266
Training Time	00:02.35	00:02.01	00:03.74	1.17	0.63
Nr. Test Samples	1446	1446	1446		
Training accuracy	1354	1305	1305	1.04	1.04
Classification Time	00:06.70	00:01.93	00:01.81	***3.47***	***3.7***
Train correctness %	93.64 %	90.25 %	90.25 %	1.04	1.04

Faces (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	576	576	576		
Nr. Train Samples	9172	9172	9172		
Nr. SVs or Hyperplanes	1902	9	9	211.33	211.33
Training Time	31:53.67	00:43:59.77	47:46.43	0.43	0.67
Nr. Test Samples	4262	4262	4262		
Training accuracy	4148	3926	3926	1.06	1.06
Classification Time	03:05.80	00:13.55	00:14.51	***13.71***	***12.8***
Train correctness %	97.33 %	92.12 %	92.12 %	1.06	1.06

Faces (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	576	576	576		
Nr. Train Samples	9172	9172	9172		
Nr. SVs or Hyperplanes	2206	4	4	551.5	551.5
Training Time	14:55.23	10:55.70	14:21.99	1.37	1.04
Nr. Test Samples	4262	4262	4262		
Training accuracy	4082	3879	3879	1.05	1.05
Classification Time	03:13.60	00:14.73	00:14.63	***13.14***	***13.23***
Train correctness %	95.78 %	91.01 %	91.01 %	1.05	1.05

Fourclass

The test and the training samples were randomly generated 1/3 of the population was training samples

Fourclass (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	2	2	2		
Nr. Train Samples	87	87	87		
Nr. SVs or Hyperplanes	135	7	5	19.29	27
Training Time	00:00.30	00:00.09	00:00.11	3.33	2.73
Nr. Test Samples	618	618	618		
Training accuracy	538	600	593	0.9	0.91
Classification Time	00:00.18	00:00.05	00:00.07	***3.6***	***2.57***
Train correctness %	87.06 %	97.09 %	95.95 %	0.9	0.91

Fourclass (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	2	2	2		
Nr. Train Samples	287	287	287		
Nr. SVs or Hyperplanes	150	16	8	9.38	18.75
Training Time	00:00.10	00:00.18	00:00.11	0.56	0.91
Nr. Test Samples	618	618	618		
Training accuracy	498	573	596	0.87	0.84
Classification Time	00:00.08	00:00.05	00:00.05	***1.6***	***1.6***
Train correctness %	80.58 %	92.72 %	96.44 %	0.87	0.84

Isolet

The test and the training samples were randomly generated 2/3 of the population was testing samples

Isolet (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	617	617	617		
Nr. Train Samples	155950	155950	155950		
Nr. SVs or Hyperplanes	35340	344	344	102.73	102.73
Training Time	07:13.75	18:51.98	01:04:11.38	0.38	0.11
Nr. Test Samples	1559	1559	1559		
Training accuracy	1499	1472	1472	1.02	1.02
Classification Time	03:01.99	00:32.85	00:36.43	***5.54***	***5***
Train correctness %	96.15 %	94.42 %	94.42 %	1.02	1.02

Isolet (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	617	617	617		
Nr. Train Samples	155950	155950	155950		
Nr. SVs or Hyperplanes	22932	325	325	70.56	70.56
Training Time	12:46.70	06:14.40	52:47.43	2.05	0.24
Nr. Test Samples	1559	1559	1559		
Training accuracy	1493	1496	1496	1	1
Classification Time	03:16.56	00:39.92	00:24.37	***4.92***	***8.07***
Train correctness %	95.77 %	95.96 %	95.96 %	1	1

5.2.1 USPS Data

The USPS data is a database for handwritten text recognition research [Hul94], the training set contains 7291 examples and the test set contains 2007 examples as provided [].

Usps (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	256	256	256		
Nr. Train Samples	18063	18063	18063		
Nr. SVs or Hyperplanes	4522	102	99	44.33	45.68
Training Time	00:27.52	00:35.26	00:02:37.48	0.78	0.17
Nr. Test Samples	7291	7291	7291		
Training accuracy	7030	6798	6816	1.03	1.03
Classification Time	02:07.23	00:29.75	00:17.22	***4.28***	***7.39***
Train correctness %	96.42 %	93.24 %	93.49 %	1.03	1.03

Usps (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	256	256	256		
Nr. Train Samples	18063	18063	18063		
Nr. SVs or Hyperplanes	3597	49	49	73.41	73.41
Training Time	00:44.74	00:22.70	02:09.58	1.97	0.35
Nr. Test Samples	7291	7291	7291		
Training accuracy	6986	6836	6836	1.02	1.02
Classification Time	01:58.59	00:19.99	00:20.07	***5.93***	***5.91***
Train correctness %	95.82 %	93.76 %	93.76 %	1.02	1.02

Worm2 (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	231	231	231		
Nr. Train Samples	493	493	493		
Nr. SVs or Hyperplanes	348	9	9	38.67	38.67
Training Time	00:05.59	00:08.57	00:17.75	0.65	0.31
Nr. Test Samples	1055	1055	1055		
Training accuracy	896	813	813	1.1	1.1
Classification Time	00:13.34	00:05.16	00:05.42	***2.59***	***2.46***
Train correctness %	84.93 %	77.06 %	77.06 %	1.1	1.1

Worm2 (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	231	231	231		
Nr. Train Samples	493	493	493		
Nr. SVs or Hyperplanes	346	4	4	86.5	86.5
Training Time	00:05.44	00:05.08	00:09.88	1.07	0.55
Nr. Test Samples	1055	1055	1055		
Training accuracy	728	811	811	0.9	0.9
Classification Time	00:14.28	00:05.40	00:05.54	***2.64***	***2.58***
Train correctness %	69 %	76.87 %	76.87 %	0.9	0.9

Nuclei (Std-Dev)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	32	32	32		
Nr. Train Samples	3372	3372	3372		
Nr. SVs or Hyperplanes	980	122	84	8.03	11.67
Training Time	00:00.98	00:03.03	00:02.99	0.32	0.33
Nr. Test Samples	65536	65536	65536		
Training accuracy	64021	61480	62265	1.04	1.03
Classification Time	01:01.70	00:23.44	00:15.18	***2.63***	***4.06***
Train correctness %	97.69 %	93.81 %	95.01 %	1.04	1.03

0	6073/6887 (88.18%)	1009/5.865e+04 (1.72%)
2	498/539 (92.39%)	1010/6.5e+04 (1.554%)
4	437/565 (77.35%)	786/6.497e+04 (1.21%)
6	673/896 (75.11%)	342/6.464e+04 (0.5291%)
10	54584/56649 (96.35%)	124/8887 (1.395%)

Nuclei (Min-Max)	RBF Kernel	Hard g1=0	Hard g1=1	RBF/Hard g1=0	RBF/Hard g1=1
Nr. Features	32	32	32		
Nr. Train Samples	3372	3372	3372		
Nr. SVs or Hyperplanes	1619	121	95	13.38	17.04
Training Time	00:00.96	00:04.59	00:03.28	0.21	0.29
Nr. Test Samples	65536	65536	65536		
Training accuracy	63065	61167	61635	1.03	1.02
Classification Time	01:30.16	00:33.47	00:33.86	***2.69***	***2.66***
Train correctness %	96.23 %	93.33 %	94.05 %	1.03	1.02

Bibliography

[BCSTW00] Kristin P. Bennett, Nello Cristianini, John Shawe-Taylor, and Donghui Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3):295–313, 2000.

[BS97] Chris J. C. Burges and Bernhard Schölkopf. Improving speed and accuracy of support vector learning machines. In M. Jordan M. Mozer and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381. MIT Press, Cambridge, MA, 1997.

[CL05] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines, 2005.

[Cri] David J. Crisp. A geometric interpretation of nu-svm classifiers.

[DGM01] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *Machine Learning*, 2:293–297, 2001.

[FCL05] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using the second order information for training svm. Technical report, National Taiwan University, 2005.

[Hul94] J. J. Hull. A database for handwritten text recognition research, 1994.

[Joa98] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[KG02] S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1-3):351–360, 2002.

[KSBM99] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to platt's smo algorithm for svm classifier design, 1999.

[LHL05] Stine R. Lin H. and Auslender L., editors. *Speeding Up Multi-class SVM Evaluation by PCA and Feature Selection*. 2005 SIAM Workshop, Newport Beach, CA, 2005.

[LL03] K. Lin and C. Lin. A study on reduced support vector machines, 2003.

[LM04] Y. Lee and O. Mangasarian. Segmentation and classification of cell nuclei in tissue slices using voxel-wise gray-scale invariants, 2004.

[NW99] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag New York, Inc., 1999.

[OFG97] E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines, 1997.

[Pla99] J.C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[SS02] B. Schölkopf and A. Smola. *Learning with Kernels.* The MIT Press, Cambridge, MA, USA, 2002.

[VC79] V. Vapnik and A. Cervonenkis. *Theorie der Zeichenerkennung.* Akademie Verlag, Berlin, 1979.