MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY
and
CENTER FOR BIOLOGICAL AND COMPUTATIONAL LEARNING
DEPARTMENT OF BRAIN AND COGNITIVE SCIENCES

A.I. Memo No. 1602
C.B.C.L Paper No. 144

March, 1997

Support Vector Machines: Training and Applications

Edgar E. Osuna, Robert Freund and Federico Girosi

Abstract

The Support Vector Machine (SVM) is a new and very promising classification technique developed by Vapnik and his group at AT&T Bell Laboratories [3, 6, 8, 24]. This new learning algorithm can be seen as an alternative training technique for Polynomial, Radial Basis Function and Multi-Layer Perceptron classifiers. The main idea behind the technique is to separate the classes with a surface that maximizes the margin between them. An interesting property of this approach is that it is an approximate implementation of the Structural Risk Minimization (SRM) induction principle [23]. The derivation of Support Vector Machines, its relationship with SRM, and its geometrical insight, are discussed in this paper.

Since Structural Risk Minimization is an inductive principle that aims at minimizing a bound on the generalization error of a model, rather than minimizing the Mean Square Error over the data set (as Empirical Risk Minimization methods do), training a SVM to obtain the maximum margin classifier requires a different objective function. This objective function is then optimized by solving a large-scale quadratic programming problem with linear and box constraints. The problem is considered challenging, because the quadratic form is completely dense, so the memory needed to store the problem grows with the square of the number of data points. Therefore, training problems arising in some real applications with large data sets are impossible to load into memory, and cannot be solved using standard non-linear constrained optimization algorithms.

We present a decomposition algorithm that can be used to train SVM's over large data sets. The main idea behind the decomposition is the iterative solution of sub-problems and the evaluation of, and also establish the stopping criteria for the algorithm. We present previous approaches, as well as results and important details of our implementation of the algorithm using a second-order variant of the Reduced Gradient Method as the solver of the sub-problems.

As an application of SVM's, we present preliminary results in Frontal Human Face Detection in images. This application opens many interesting questions and future research opportunities, both in the context of faster and better optimization algorithms, and in the use of SVM's in other pattern classification, recognition, and detection applications.

This report describes research done within the Center for Biological and Computational Learning in the Department of Brain and Cognitive Sciences and the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. This research is sponsored by MURI grant N00014-95-1-0600; by a grant from ONR/ARPA under contract N00014-92-J-1879 and by the National Science Foundation under contract ASC-9217041 (this award includes funds from ARPA provided under the HPCC program). **Edgar Osuna** was supported by Fundación Gran Mariscal de Ayacucho and Daimler Benz. Additional support is provided by Daimler-Benz, Eastman Kodak Company, Siemens Corporate Research, Inc. and AT&T.

Contents

1 Introduction

In this report we address the problem of implementing a new pattern classification technique recently developed by Vladimir Vapnik and his team at AT&T Bell Laboratories [3, 6, 8, 24] and known as Support Vector Machine (SVM). SVM can be thought as an alternative training technique for Polynomial, Radial Basis Function and Multi-Layer Perceptron classifiers, in which the weights of the network are found by solving a Quadratic Programming (QP) problem with linear inequality and equality constraints, rather than by solving a non-convex, unconstrained minimization problem, as in standard neural network training techniques. Since the number of variables in the QP problem is equal to the number of data points, when the data set is large this optimization problem becomes very challenging, because the quadratic form is completely dense and the memory requirements grow with the square of the number of data points. We present a decomposition algorithm that guarantees global optimality, and can be used to train SVM's over very large data sets (say 50,000 data points). The main idea behind the decomposition is the iterative solution of sub-problems and the evaluation of optimality conditions which are used both to generate improved iterative values, and also establish the stopping criteria for the algorithm.

We demonstrate the effectiveness of our approach applying SVM to the problem of detecting frontal faces in images, which involves the solution of a pattern classification problem (face versus non-face patterns) with a large data base (50,000 examples). The reason for the choice of the face detection problem as an application of SVM is twofold: 1) the problem has many important practical applications, and received a lot of attention in recent years; 2) the difficulty in the implementation of SVM arises only when the data base is large, say larger than 2,000, and this problems does involve a large data base.

The paper is therefore divided in two main parts. In the first part, consisting of sections 2 and 3 we describe the SVM approach to pattern classification and our solution of the corresponding QP problem in the case of large data bases. In the second part (section 4) we describe a face detection system, in which the SVM is one of the main components. In particular, section 2 reviews the theory and derivation of SVM's, together with some extensions and geometrical interpretations. Section 3 starts by reviewing the work done by Vapnik *et al.* [5] in solving the training problem for the SVM. Section 3.1.1 gives a brief description and references of the initial approaches we took in order to solve this problem. Section 3.2 contains the main result of this paper, since it presents the new approach that we have developed to solve Large Database Training problems of Support Vector Machines.

Section 4 presents a Frontal Human Face Detection System that we have developed as an application of SVM's to computer vision object detection problems.

2 Support Vector Machines

In this section we introduce the SVM classification technique, and show how it leads to the formulation of a QP programming problem in a number of variables that is equal to the number of data points. We will start by reviewing the classical Empirical Risk Minimization approach, and showing how it naturally leads, through the theory of VC bounds, to the idea of Structural Risk Minimization (SRM), which is a better induction principle, and how SRM is implemented by SVM.

2.1 Empirical Risk Minimization

In the case of two-class pattern recognition, the task of learning from examples can be formulated in the following way: given a set of decision functions

$$\{f_\lambda(\mathbf{x}) : \lambda \in \Lambda\}, \qquad f_\lambda : \Re^N \to \{-1, 1\}$$

where Λ is a set of abstract parameters, and a set of examples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \qquad \mathbf{x}_i \in \Re^N, y_i \in \{-1, 1\}$$

drawn from an unknown distribution $P(\mathbf{x}, y)$, we want to find a function f_{λ^*} which provides the smallest possible value for the *expected risk*:

$$R(\lambda) = \int |f_\lambda(\mathbf{x}) - y| \, P(\mathbf{x}, y) d\mathbf{x} dy$$

The functions f_λ are usually called *hypothesis*, and the set $\{f_\lambda(\mathbf{x}) : \lambda \in \Lambda\}$ is called the *hypothesis space* and denoted by $\mathcal{H}$. The expected risk is therefore a measure of how good an hypothesis is at predicting the correct label y for a point $\mathbf{x}$. The set of functions f_λ could be for example, a set of Radial Basis Functions or a Multi-Layer Perceptron with a certain number of hidden units. In this case, the set Λ is the set of weights of the network.

Since the probability distribution $P(\mathbf{x}, y)$ is unknown, we are unable to compute, and therefore to minimize, the expected risk $R(\lambda)$. However, since we have access to a sampling of $P(\mathbf{x}, y)$, we can compute a stochastic approximation of $R(\lambda)$, the so called *empirical risk*:

$$R_{\mathrm{emp}}(\lambda) = \frac{1}{\ell} \sum_{i=1}^{\ell} |f_\lambda(\mathbf{x}_i) - y_i|$$

Since the law of large numbers guarantees that the empirical risk converges in probability to the expected risk, a common approach consists in minimizing the empirical risk rather than the expected risk. The intuition underlying this approach (the *Empirical Risk Minimization Principle*) is that if R_{emp} converges to R, the minimum of R_{emp} may converge to the minimum of R. If convergence of the minimum of R_{emp} to the minimum of R does not hold, the Empirical Risk Minimization Principle does not allow us to make any inference based on the data set, and it is therefore said to be *not consistent*. As shown by Vapnik and Chervonenkis [25, 26, 23] consistency takes place if and only if convergence in probability of R_{emp} to R is replaced by *uniform* convergence in probability. Vapnik and Chervonenkis [25, 26, 23] showed that necessary and sufficient condition for consistency of the Empirical Risk Minimization Principle is the finiteness of the *VC-dimension* h of the hypothesis space $\mathcal{H}$. The VC-dimension of the hypothesis space $\mathcal{H}$ (or VC-dimension of the classifier f_λ) is a natural number, possibly infinite, which is, losely speaking, the largest number of data points that can be separated in all possible ways by that set of functions f_λ. The VC-dimension is a measure of the complexity of the set $\mathcal{H}$, and it is often, but not necessarily, proportional to the number of free parameters of the classifier f_λ.

The theory of uniform convergence in probability developed by Vapnik and Chervonenkis also provides bounds on the deviation of the empirical risk from the expected risk. A typical uniform Vapnik and Chervonenkis bound, which holds with probability $1 - \eta$, has the following form:

$$R(\lambda) \leq R_{\mathrm{emp}}(\lambda) + \sqrt{\frac{h\left(\ln \frac{2l}{h} + 1\right) - \ln \frac{\eta}{4}}{l}} \qquad \forall \lambda \in \Lambda \tag{1}$$

where h is the VC-dimension of f_λ. From this bound it is clear that, in order to achieve small expected risk, that is good generalization performances, both the empirical risk and the ratio between the VC-dimension and the number of data points has to be small. Since the empirical risk is usually a decreasing function of h, it turns out that, for a given number of data points, there is an optimal value of the VC-dimension. The choice of an appropriate value for h (which in most techniques is controlled by the number of free parameters of the model) is crucial in order to get good performances, especially when the number of data points is small. When using a Multilayer Perceptron or a Radial Basis Functions network, this is equivalent to the problem of finding the appropriate number of hidden units. This problem is known to be difficult, and it is usually solved by some sort of cross-validation technique.

The bound (1) suggests that the Empirical Risk Minimization Principle can be replaced by a better induction principle, as we will see in the next section.

2.2 Structural Risk Minimization

The technique of *Structural Risk Minimization* developed by Vapnik [23] is an attempt to overcome the problem of choosing an appropriate VC-dimension. It is clear from eq. (1) that a small value of the empirical risk does not necessarily imply a small value of the expected risk. A different induction principle, called the *Structural Risk Minimization Principle*, has been proposed by Vapnik [23]. The principle is based on the observation that, in order to make the expected risk small, both sides in equation (1) should be small. Therefore, both the VC-dimension and the empirical risk should be minimized at the same time. In order to implement the SRM principle one needs then a nested structure of hypothesis spaces

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_n \subset \ldots$$

with the property that $h(n) \leq h(n+1)$ where $h(n)$ is the VC-dimension of the set $\mathcal{H}_n$. Then equation (1) suggests that, disregarding logarithmic factors, the following problem should be solved:

$$\min_{\mathcal{H}_n} \left(R_{\text{emp}}[\lambda] + \sqrt{\frac{h(n)}{l}} \right) \tag{2}$$

The SRM principle is clearly well founded mathematically, but it can be difficult to implement for the following reasons:

1. The VC-dimension of $\mathcal{H}_n$ could be difficult to compute, and there are only a small number of models for which we know how to compute the VC-dimension.

2. Even assuming that we can compute the VC-dimension of $\mathcal{H}_n$, it is not easy to solve the minimization problem (2). In most cases one will have to minimize the empirical risk for every set $\mathcal{H}_n$, and then choose the $\mathcal{H}_n$ that minimizes eq. (2).

Therefore the implementation of this principle is not easy, because it is not trivial to control the VC-dimension of a learning technique during the training phase. The SVM algorithm achieves this goal, minimizing a bound on the VC-dimension and the number of training errors at the same time. In the next section we discuss this technique in detail, and show how its implementation is related to quadratic programming.

2.3 Support Vector Machines: Mathematical Derivation

In this section we describe the mathematical derivation of the Support Vector Machine (SVM) developed by Vapnik [24]. The technique is introduced by steps: we first consider the simplest case, a linear classifier and a linearly separable problem; then a linear classifier and non-separable problem, and finally a non-linear classifier and non-separable problem, which is the most interesting and useful case.

2.3.1 Linear Classifier and Linearly Separable Problem

In this section we consider the case in which the data set is *linearly separable*, and we wish to find the "best" hyperplane that separates the data. For our purposes, linearly separable means that we can find a pair $(\mathbf{w}, b)$ such that:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq \quad 1 \qquad \forall \mathbf{x}_i \in \text{Class 1} \tag{3}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq \quad -1 \qquad \forall \mathbf{x}_i \in \text{Class 2} \tag{4}$$

The hypothesis space in this case is therefore the set of functions given by

$$f_{\mathbf{w},b} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{5}$$

Notice that if the parameters $\mathbf{w}$ and b are scaled by the same quantity, the decision surface given by (5) is unchanged. In order to remove this redundancy, and to make each decision surface correspond to one unique pair $(\mathbf{w}, b)$, the following constraint is imposed:

$$\min_{i=1,\ldots,\ell} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1 \tag{6}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ are the points in the data set. The set of hyperplanes that satisfy (6) are called *Canonical Hyperplanes.* Notice that all linear decision surfaces can be represented by *Canonical Hyperplanes*, and constraint (6) is just a normalization, which will prove to be very convenient in the following calculations. If no further constraints are imposed on the pair $(\mathbf{w}, b)$ the VC-dimension of the *Canonical Hyperplanes* is $N+1$ [24], that is, the total number of free parameters. In order to be able to apply the Structural Risk Minimization Principle we need to construct sets of hyperplanes of varying VC-dimension, and minimize both the empirical risk (the training classification error) and the VC-dimension at the same time. A structure on the set of canonical hyperplanes is defined by constraining the norm of the vector $\mathbf{w}$. In fact, Vapnik shows that, if we assume that all the points $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ lie in the unit N-dimensional sphere, the set

$$\{f_{\mathbf{w},b} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \mid ||\mathbf{w}|| \leq A\} \tag{7}$$

has a *VC-dimension* h that satisfies the following bound [24] [23]:

$$h \leq \min\{\lceil A^2 \rceil, N\} + 1 \tag{8}$$

If the data points lie inside a sphere of radius R, then (8) becomes $h \leq \min\{\lceil R^2 A^2 \rceil, N\} + 1$. The geometrical reason for which bounding the norm of $\mathbf{w}$ constraints the set of canonical hyperplanes is very simple. It can be shown that the distance from a point $\mathbf{x}$ to the hyperplane associated to the pair $(\mathbf{w}, b)$ is:

$$d(\mathbf{x}; \mathbf{w}, b) = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{||\mathbf{w}||} \tag{9}$$

According to the normalization (6) the distance between the canonical hyperplane $(\mathbf{w}, b)$ and the closest of the data points is simply $\frac{1}{||\mathbf{w}||}$. Therefore, if $||\mathbf{w}|| \leq A$ then the distance of the canonical hyperplane to the closest data point has to be larger than $\frac{1}{A}$. We can then conclude that the constrained set of canonical hyperplanes of eq. (7) is the set of hyperplanes whose distance from the data points is at least $\frac{1}{A}$. This is equivalent to placing spheres of radius $\frac{1}{A}$ around each data point, and consider only the hyperplanes that do not intersect any of the spheres, as shown in figure (1).
If the set of examples is linearly separable, the goal of the SVM is to find, among the *Canonical Hyperplanes* that correctly classify the data, the one with minimum norm, or equivalently minimum $||\mathbf{w}||^2$, because keeping this norm small will also keep the *VC-dimension* small. It is interesting to see that minimizing $||\mathbf{w}||^2$ (in this case of linear separability) is equivalent to finding the separating hyperplane for which the distance between the two convex hulls (of the two classes of training data), measured along a line perpendicular to the hyperplane, is maximized. In the rest of this paper, this distance will be referred to as the *margin.* Figure (2) gives some geometrical interpretation of why better *generalization* is expected from a separating hyperplane with large *margin.*
To construct the *maximum margin* or *optimal* separating hyperplane, we need to correctly classify the vectors $\mathbf{x}_i$ of the training set

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \qquad \mathbf{x}_i \in \Re^N$$

into two different classes $y_i \in \{-1, 1\}$, using the smallest norm of coefficients. This can be formulated as follows:

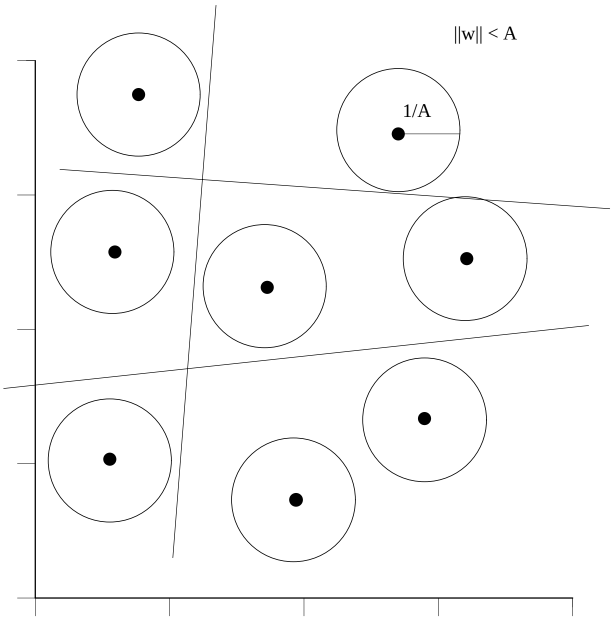


Figure 1: Bounding the norm of $\mathbf{w}$ is equivalent to constraint the hyperplanes to remain outside spheres of radius $\frac{1}{A}$ centered around the data points.

$$\begin{array}{lll} \underset{\mathbf{w},b}{\text{Minimize}} & \Phi(\mathbf{w}) & = \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} & & \\ & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) & \geq 1 \qquad i = 1\ldots\ell \end{array} \tag{10}$$

At this point, this problem can be solved using standard Quadratic Programming (QP) optimization techniques and is not very complex since the dimensionality is $N+1$. Since N is the dimension of the input space, this problem is more or less tractable for real applications. Nevertheless, in order to easily explain the extension to nonlinear decision surfaces (which will be described in section 2.3.3), we look at the dual problem, and use the technique of Lagrange Multipliers. We construct the Lagrangian

$$L(\mathbf{w}, b, \mathbf{\Lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1], \tag{11}$$

where $\mathbf{\Lambda} = (\lambda_1, \ldots, \lambda_\ell)$ is the vector of non-negative Lagrange multipliers corresponding to the constraints in (10).

The solution to this optimization problem is determined by a saddle point of this Lagrangian, which has to be minimized with respect to $\mathbf{w}$ and b, and maximized with respect to $\mathbf{\Lambda} \geq 0$. Differentiating (11) and setting the results equal to zero we obtain:

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i = 0 \tag{12}$$

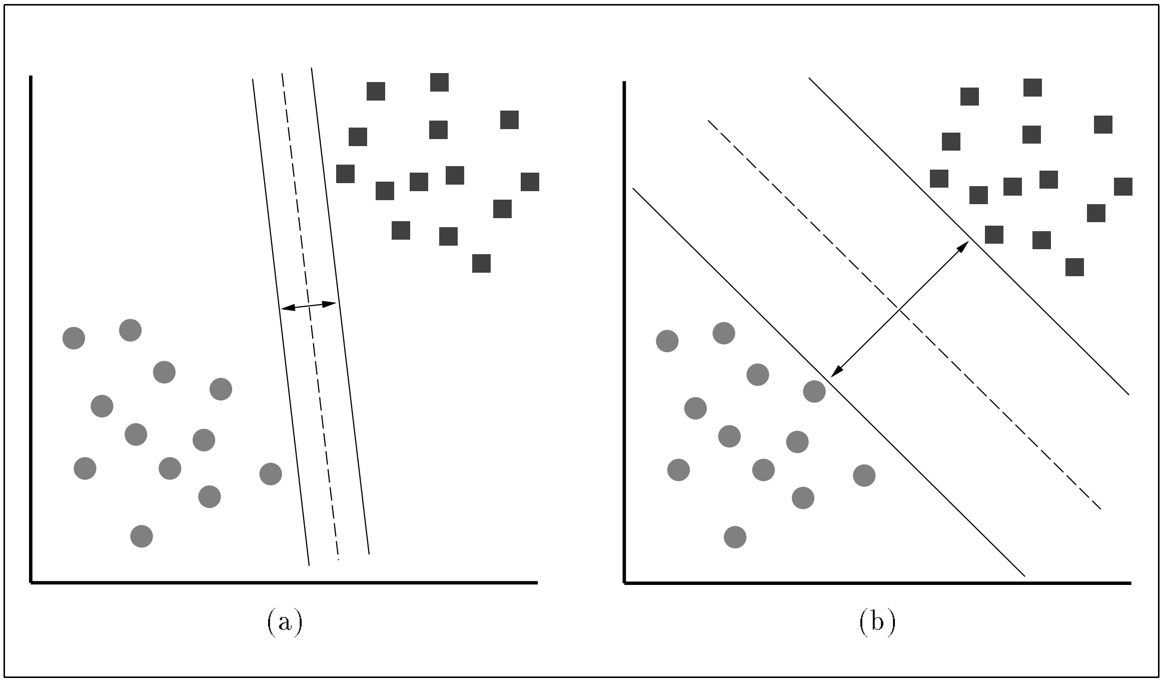


Figure 2: (a) A Separating Hyperplane with small *margin*. (b) A Separating Hyperplane with larger *margin*. A better *generalization* capability is expected from (b).

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda})}{\partial b} = \sum_{i=1}^{\ell} \lambda_i y_i = 0 \tag{13}$$

Using the superscript $*$ to denote the *optimal* values of the cost function, from equation (12) we derive:

$$\mathbf{w}^* \; = \; \sum_{i=1}^{\ell} \lambda_i^* y_i \mathbf{x}_i \tag{14}$$

which shows that the *optimal* hyperplane solution can be written as a linear combination of the training vectors. Notice that only those training vectors $\mathbf{x}_i$ with $\lambda_i > 0$ contribute in the expansion (14). Substituting (14) and (13) into (11) we obtain:

$$F(\mathbf{\Lambda}) \; = \; \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\|\mathbf{w}^*\|^2 \; = \; \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{15}$$

Writing (15) in matrix notation, incorporating non-negativity of $\mathbf{\Lambda}$ and constraint (13) we get the following dual quadratic program:

$$\begin{array}{lll} \text{Maximize} & F(\mathbf{\Lambda}) & = \mathbf{\Lambda} \cdot \mathbf{1} - \frac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ \text{subject to} & & \\ & \mathbf{\Lambda} \cdot \mathbf{y} & = 0 \\ & \mathbf{\Lambda} & \geq 0 \end{array}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$ and D is a symmetric $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$.

Notice that complementary slackness conditions of the form:

$$\lambda_i^*[y_i(\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1] = 0 \qquad i = 1, \ldots, \ell \tag{16}$$

imply that $\lambda_i > 0$ only when constraint (10) is active. The vectors for which $\lambda_i > 0$ are called *Support Vectors.* From equation (16) it follows that b^* can be computed as:

$$b^* = y_i - \mathbf{w}^* \cdot \mathbf{x}_i$$

for any support vector $\mathbf{x}_i$. By linearity of the dot product and equation (14), the decision function (5) can then be written as:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x}_i) + b^*\right) \tag{17}$$

2.3.2 The Soft Margin Hyperplane: Linearly Non-Separable Case

We now consider the case in which we still look for a linear superating surface, but a separating hyperplane does not exist, so that it is not possible to satisfy all the constraints in problem (10). In order to deal with this case one introduces a new set of variables $\{\xi_i\}_{i=1}^{\ell}$, that measure the amount of violation of the constraints. Then the margin is maximized, paying a penalty proportional to the amount of constraint violations. Formally, one solves the following problem:

$$\begin{array}{lll} \text{Minimize} & \Phi(\mathbf{w}, \Xi) & = \frac{1}{2}\|\mathbf{w}\|^2 + C(\sum_{i=1}^{\ell} \xi_i)^k \\ \mathbf{w}, b, \Xi & & \\ \text{subject to} & & \end{array} \tag{18}$$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \qquad i = 1, \ldots, \ell \tag{19}$$

$$\xi_i \geq 0 \qquad i = 1, \ldots, \ell \tag{20}$$

where C and k are parameters which have to be determined beforehand and define the cost of constraints violation. Other monotonic convex functions of the errors can be defined (see [8] for the more general case). Notice that minimizing the first term in (18) amounts to minimizing the *VC-dimension* of the learning machine, thereby minimizing the second term in the bound (1). On the other hand, minimizing the second term in (18) controls the *empirical risk*, which is the first term in the bound (1). This approach, therefore, constitutes a practical implementation of *Structural Risk Minimization* on the given set of functions. In order to solve problem (18), we construct the Lagrangian:

$$L(\mathbf{w}, b, \mathbf{\Lambda}, \Xi, \mathbf{\Gamma}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{\ell} \gamma_i \xi_i + C(\sum_{i=1}^{\ell} \xi_i)^k, \tag{21}$$

where the non-negative multipliers $\mathbf{\Lambda} = (\lambda_1, \ldots, \lambda_\ell)$ and $\mathbf{\Gamma} = (\gamma_1, \ldots, \gamma_\ell)$ are associated with constraints (19) and (20) respectively. The solution to this problem is determined by the saddle point of this Lagrangian, which has to be minimized with respect to $\mathbf{w}$, Ξ and b, and maximized with respect to $\mathbf{\Lambda} \geq 0$ and $\mathbf{\Gamma} \geq 0$. Differentiating (21) and setting the results equal to zero, we obtain:

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda}, \Xi, \mathbf{\Gamma})}{\partial \mathbf{w}} = (\mathbf{w} - \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i) = 0 \tag{22}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda}, \mathbf{\Xi}, \mathbf{\Gamma})}{\partial b} = \sum_{i=1}^{\ell} \lambda_i y_i = 0 \tag{23}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda}, \mathbf{\Xi}, \mathbf{\Gamma})}{\partial \mathbf{\Xi}} = \begin{cases} kC\left(\sum_{i=1}^{\ell} \xi_i\right)^{k-1} - \lambda_i - \gamma_i = 0 & k > 1 \\ C - \lambda_i - \gamma_i = 0 & k = 1. \end{cases} \tag{24}$$

When $k > 1$, by denoting

$$\sum_{i=1}^{\ell} \xi_i = \left(\frac{\delta}{Ck}\right)^{\frac{1}{k-1}}, \tag{25}$$

we can rewrite equation (24) as:

$$\delta - \lambda_i - \gamma_i = 0. \tag{26}$$

From equation (22) we obtain:

$$\mathbf{w}^* = \sum_{i=1}^{\ell} \lambda_i^* y_i \mathbf{x}_i \tag{27}$$

Substituting (27), (23) and (25) into (21) we obtain:

$$F(\mathbf{\Lambda}, \delta) = \sum_{i=1}^{\ell} \lambda_i - \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1 - \frac{1}{k}\right) \tag{28}$$

Therefore, in order to obtain the *Soft Margin* separating hyperplane we solve:

$$\begin{array}{lll} \text{Maximize} & F(\mathbf{\Lambda}, \delta) & = \mathbf{\Lambda} \cdot \mathbf{1} - \frac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1 - \frac{1}{k}\right) \\ \text{subject to} & & \\ & \mathbf{\Lambda} \cdot \mathbf{y} & = 0 \\ & \mathbf{\Lambda} & \leq \delta\mathbf{1} \\ & \mathbf{\Lambda} & \geq \mathbf{0} \end{array} \tag{29}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$ and D is a symmetric $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$.
When $k = 1$, that is, penalizing linearly the violations in constraint (19), the set of equations (29) simplifies to:

$$\begin{array}{lll} \text{Maximize} & F(\mathbf{\Lambda}) & = \mathbf{\Lambda} \cdot \mathbf{1} - \frac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ \text{subject to} & & \\ & \mathbf{\Lambda} \cdot \mathbf{y} & = 0 \\ & \mathbf{\Lambda} & \leq C\mathbf{1} \\ & \mathbf{\Lambda} & \geq \mathbf{0} \end{array} \tag{30}$$

The value $k = 1$ is assumed for the rest of this paper, since it simplifies the mathematical formulation and has shown very good results in practical applications. By the linearity of the dot product and equation (27), the decision function (5) can be written as:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x}_i) + b^*\right) \tag{31}$$

where $b^* = y_i - \mathbf{w}^* \cdot \mathbf{x}_i$, for any support vector $\mathbf{x}_i$ such that $0 < \lambda_i < C$ (that is a support vector which is correctly classified). In order to verify this, notice that complementary slackness in the conditions of the form:

$$\lambda_i^*[y_i(\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1 + \xi_i] = 0 \qquad i = 1, \ldots, \ell \tag{32}$$

imply that $\lambda_i > 0$ only when constraint (19) is active, establishing the need for $\lambda_i > 0$. On the other hand, (19) can be active due to $\xi_i > 0$, which is not acceptable since $\mathbf{x}_i$ would be a misclassified point. For $k = 1$ in equation (24) we have $\gamma_i = C - \lambda_i$. Since γ_i is the multiplier associated with constraint (20), $\gamma_i > 0$ implies $\xi_i = 0$, establishing the sufficiency of $\lambda_i < C$. Notice that this is a sufficient condition, since both γ_i and λ_i could be equal to zero.

Note:
Our calculation above of the threshold value b assumes the existence of some λ_i such that $0 < \lambda_i < C$. We have not found a proof yet of the existence of such λ_i, or conditions under which it does not exist. However, we think this is a very reasonable assumption, because it is equivalent to the assumption that there is at least one support vector which is correctly classified. So far our computational results indicate that this assumption is correct, and we will use it in the rest of this paper.

2.3.3 Nonlinear Decision Surfaces

Previous sections have only treated linear decision surfaces, which are definitely not appropriate for many tasks. The extension to more complex decision surfaces is conceptually quite simple, and is done by mapping the input variable $\mathbf{x}$ in a higher dimensional *feature space*, and by working with linear classification in that space. More precisely, one maps the input variable $\mathbf{x}$ into a (possibly infinite) vector of "feature" variables:

$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = (a_1\phi(\mathbf{x}), a_2\phi_2(\mathbf{x}), \ldots, a_n\phi_n(\mathbf{x}), \ldots) \tag{33}$$

where $\{a_n\}_{n=1}^{\infty}$ are some real numbers and $\{\phi_n\}_{n=1}^{\infty}$ are some real functions[1]. The Soft Margin version of SVM is then applied, substituting the variable $\mathbf{x}$ with the new "feature vector" $\boldsymbol{\phi}(\mathbf{x})$. Under the mapping (33) the solution of a SVM has the form:

$$f(\mathbf{x}) = \text{sign}\,(\phi(\mathbf{x}) \cdot \mathbf{w}^* + b^*) \Rightarrow \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x}_i) + b^*\right) \tag{34}$$

A key property of the SV machinery is that the only quantities that one needs to compute are scalar products, of the form $\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})$. It is therefore convenient to introduce the so-called *kernel function* K:

$$K(\mathbf{x}, \mathbf{y}) \equiv \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y}) = \sum_{n=1}^{\infty} a_n^2 \phi_n(\mathbf{x}) \phi_n(\mathbf{y}) \tag{35}$$

Using this quantity the solution of a SVM has the form:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* K(\mathbf{x}, \mathbf{x}_i) + b^*\right) \tag{36}$$

and the quadratic programming problem (30) becomes:

[1] The numbers $\{a_n\}_{n=1}^{\infty}$ are clearly unnecessary, and could be absorbed in the definition of the $\{\phi_n\}_{n=1}^{\infty}$, but we use them here just because they make the formulation easier.

$$\begin{array}{lll} \text{Maximize} & F(\mathbf{\Lambda}) & = \mathbf{\Lambda}\cdot\mathbf{1} - \frac{1}{2}\mathbf{\Lambda}\cdot D\mathbf{\Lambda} \\ \text{subject to} & & \\ & \mathbf{\Lambda}\cdot\mathbf{y} & = 0 \\ & \mathbf{\Lambda} & \leq C\mathbf{1} \\ & \mathbf{\Lambda} & \geq \mathbf{0} \end{array} \qquad (37)$$

where D is a symmetric, semi-positive definite, $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. Notice that the decision surface (36) is now a nonlinear function, given by linear superposition of kernel functions, one for each support vector. The idea of expanding the input space in a feature space is therefore useful only if we find some solution to the following problem: starting from the feature space or starting from the kernel.

Problem 2.1 *Find a set of coefficients $\{a_n\}_{n=1}^{\infty}$ and a set of features $\{\phi_n\}_{n=1}^{\infty}$ such that:*

1. *the scalar product $K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{y})$ is well defined (for example the series converges uniformly);*

2. *the scalar product $K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{y})$ is easy to compute as a function of $\mathbf{x}$ and $\mathbf{y}$;*

In addition to these requirements, we also should require the features ϕ_i to be such that the scalar product $K(\mathbf{x},\mathbf{y})$ defines a class of decision surfaces which is "rich" enough (for example includes some well-known approximation schemes). There are two different approaches to this problem.

Starting from the feature space

One approach consists in choosing carefully a set of features with "good" properties. For example, an obvious choice would be to take as features $\phi_i(\mathbf{x})$ monomials in the variable $\mathbf{x}$ up to a certain degree. Assuming, for simplicity, to work in a one-dimensional space, one could choose:

$$\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^d)$$

where d could be very high, and the coefficients a_i are all equal to one. In this case the decision surface is linear in the components of $\boldsymbol{\phi}$, and therefore a polynomial of degree d in x. This choice is unfortunate, however, because the scalar product

$$\boldsymbol{\phi}(x)\cdot\boldsymbol{\phi}(y) = 1 + xy + (xy)^2 + \ldots (xy)^d$$

is not particularly simple to compute when d is high. However, it is easy to see that, with a careful choice of the parameters a_i things simplify. In fact, choosing

$$a_n = \binom{d}{n}$$

it is easy to see that

$$\boldsymbol{\phi}(x)\cdot\boldsymbol{\phi}(y) = \sum_{n=0}^{d} \binom{d}{n} (xy)^n = (1+xy)^d$$

which considerably reduces the computation. A similar result, although with a more complex structure of the coefficients a_n, is true in the multivariable case, where the dimensionality of the feature space grows very quickly with the number of variables. For example, in two variables we can define:

$$\boldsymbol{\phi}(\mathbf{x}) = (1, \sqrt{2}\, x_1, \sqrt{2}\, x_2, x_1^2, x_2^2, \sqrt{2}\, x_1 x_2) \qquad (38)$$

In this case it is easy to see that:

$$K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^2 \tag{39}$$

It is straightforward to extend this example to the d-dimensional case. For example, in 3 dimensions we have:

$$\boldsymbol{\phi}(\mathbf{x}) = (1,\sqrt{2}\;x_1,\sqrt{2}\;x_2,\sqrt{2}\;x_3,x_1^2,x_2^2,x_3^2,\sqrt{2}\;x_1x_2,\sqrt{2}\;x_1x_3,\sqrt{2}\;x_2x_3)$$

and the scalar product is still of the form of eq. (39). Still in 2 dimension we can use features which are monomials of degree 3:

$$\boldsymbol{\phi}(\mathbf{x}) = (1,\sqrt{3}\;x_1,\sqrt{3}\;x_2,\sqrt{3}\;x_1^2,\sqrt{3}\;x_2^2,\sqrt{6}x_1x_2,\sqrt{3}\;x_1^2x_2,\sqrt{3}\;x_1x_2^2,x_1^3\;,x_2^3\;)$$

and it can be shown that:

$$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^3$$

It can also be shown that if the features are monomials of degree less or equal to d, it is always possible to find numbers a_n in such a way that the scalar product is

$$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^d \tag{40}$$

In the following we provide a few more examples of how one could choose the features first, and then, with a careful choice of the coefficients a_n, arrive at an analytical expression for the kernel K.

Infinite dimensional feature spaces

We consider one dimensional examples. Multidimensional kernels can be built using tensor products of one-dimensional kernels.

1. Let $x \in [0,\pi]$ and let us consider the following feature space:

$$\boldsymbol{\phi}(x) = (\sin(x),\frac{1}{\sqrt{2}}\sin(2x),\frac{1}{\sqrt{3}}\sin(3x),\ldots,\frac{1}{\sqrt{n}}\sin(nx),\ldots)$$

   Then

$$K(x,y) = \boldsymbol{\phi}(x)\cdot\boldsymbol{\phi}(x) = \sum_{n=1}^{\infty}\frac{1}{n}\sin(nx)\sin(ny) = \frac{1}{2}\log\left|\frac{\sin\frac{x+y}{2}}{\sin\frac{x-y}{2}}\right|$$

   which corresponds to the choice $a_n = \frac{1}{\sqrt{n}}$.

2. Let $x \in [0,2\pi]$, h a positive number such that $h < 1$, and let us consider the following feature space:

$$\boldsymbol{\phi}(x) = (1,h^{\frac{1}{2}}\sin(x),h^{\frac{1}{2}}\cos(x),h\sin(2x),h\cos(2x),\ldots,h^{\frac{n}{2}}\sin(nx),h^{\frac{n}{2}}\cos(nx),\ldots)$$

   Then

$$K(x,y) = \boldsymbol{\phi}(x)\cdot\boldsymbol{\phi}(x) = 1+\sum_{n=1}^{\infty}h^n\sin(nx)\sin(ny)+\sum_{n=1}^{\infty}h^n\cos(nx)\cos(ny) =$$
$$= \frac{1}{2\pi}\frac{1-h^2}{1-2h\cos(x-y)+h^2}$$

   which corresponds to the choice $a_n = h^{\frac{n}{2}}$.

3. In the two examples above we have an infinite number of features, but countable. We can also construct cases in which the number of features is infinite and uncountable. Let us consider the following map:

$$\boldsymbol{\phi}(\mathbf{x}) = \left\{ \sqrt{\tilde{G}(\mathbf{s})} e^{i\mathbf{x}\cdot\mathbf{s}} \quad | \ \mathbf{s} \in R^d \right\}$$

   where $\tilde{G}(\mathbf{s})$ is the Fourier Transform of a positive definite function, and where we work, for simplicity, with complex features. This corresponds to a kernel

$$K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y}) = \int_{R^d} d\mathbf{s} \ \tilde{G}(\mathbf{s}) e^{i(\mathbf{x}-\mathbf{y})\cdot\mathbf{s}} = G(\mathbf{x}-\mathbf{y}).$$

   which corresponds to a continuum of coefficients $a(\mathbf{s}) = \sqrt{\tilde{G}(\mathbf{s})}$.

Starting from the kernel

Another approach consists in looking for a kernel which is known to have a representation of the form (35) for some set of ϕ_i, but whose explicit analytic form may not be known.

In order to find a solution to this problem we need some preliminary facts. Let us call *positive definite kernel* any function $K(\mathbf{x},\mathbf{y})$ on $\Omega \times \Omega$, with $\Omega \subset R^d$, with the property that:

$$\sum_{i,j=1}^{n} K(\mathbf{x}_i,\mathbf{x}_j) c_i c_j \geq 0 \quad \forall \mathbf{x}_i, \mathbf{x}_j \in \Omega \ , \ \ \forall c_i, c_j \in R \tag{41}$$

In the following, we will assume that $\Omega \equiv [a,b]^d$ The kernel K defines an integral operator that is known to have a complete system of orthonormal eigenfunctions:

$$\int_{\Omega} d\mathbf{y} \ K(\mathbf{x},\mathbf{y}) \phi_n(\mathbf{y}) = \lambda_n \phi_n(\mathbf{x}) \tag{42}$$

In 1976, Stewart [20] reported that, according to a theorem of Mercer from 1909 [13] the following statements are equivalent:

1. The function $K(\mathbf{x},\mathbf{y})$ is a positive definite kernel;

2. $$\int_{[a,b]^d} d\mathbf{x} d\mathbf{y} \ K(\mathbf{x},\mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) \geq 0 \quad \forall g \in C([a,b]^d)$$

3. The eigenvalues λ_n in eq. (42) are all positive;

4. The series

$$K(\mathbf{x},\mathbf{y}) = \sum_{n=1}^{\infty} a_n^2 \phi_n(\mathbf{x}) \phi_n(\mathbf{y})$$

   (where $a_n^2 = \frac{1}{\lambda_n}$) converges absolutely and uniformly.

This leads to the following:

Statement 2.1 *Any feature vector* $\boldsymbol{\phi}(\mathbf{x}) = (a_1\phi(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_n(\mathbf{x}), \ldots)$ *such that the* $\{a_n\}_{n=1}^{\infty}$ *and the* $\{\phi_n\}_{n=1}^{\infty}$ *are respectively the eigenvalues and the eigenfunctions of a positive definite kernel* $K(\mathbf{x},\mathbf{y})$ *will solve problem (2.1), and the scalar product* $\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})$ *has the following simple expression:*

$$\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y}) = K(\mathbf{x},\mathbf{y})$$

A number of observations are in order:

- Vapnik (1995) uses the condition (2) above to characterize the kernels that can be used in SVM. Definition (41) can be used instead, and might be more practical to work with if one has to prove the "admissibility" of a certain kernel.

- There is another result similar to Mercer's one, but more general. Young (1909) proves that a kernel is positive definite if and only if

$$\int_{\Omega} d\mathbf{x} d\mathbf{y}\ K(\mathbf{x},\mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) \geq 0 \quad \forall g \in L_1(\Omega)$$

- The kernels K that can be used to represent a scalar product in the feature space are closely related to the theory of Reproducing Kernel Hilbert Spaces (RKHS) (see appendix A in (Girosi, 1997)[9]). In fact, in 1916 Moore [15] considers a more general setting for positive definite kernels, and replaces Ω in eq. (41) with any abstract set E. He calls these functions *positive Hermitian matrices* and shows that to any such K one can associate a RKHS.

In table (1) we report some commonly used kernels.

Kernel Function	Type of Classifier
$K(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2)$	Gaussian RBF
$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^d$	Polynomial of degree d
$K(\mathbf{x},\mathbf{y}) = \tanh(\mathbf{x}\cdot\mathbf{y}-\theta)$ (only for some values of θ)	Multi Layer Perceptron

Table 1: Some possible kernel functions and the type of decision surface they define.

2.4 Additional Geometrical Interpretation

Just as Figure (2) shows why better generalization is expected from maximizing the margin, one should wonder: do the support vectors have any geometrical common characteristic? Are they just scattered points used in a linear combination? It turns out that they are not.

In order to find the optimal decision surface, the support vector training algorithm tries to separate, as best as possible, the *clouds* defined by the data points from both classes.

Particularly, one would expect points closer to the *boundary* between the classes to be more important in the solution than data points that are far away, since the first are harder to classify. These data points, in some sense, help shape and define better the decision surface than other points. Therefore, the support vectors are from a geometrical point of view *border points*.

A direct consequence of the preceding argument delivers another important geometrical and algorithmic property, which is that, usually, the support vector are very few.

These ideas can be justified algebraically through the optimality conditions derived in section 3.2.1.

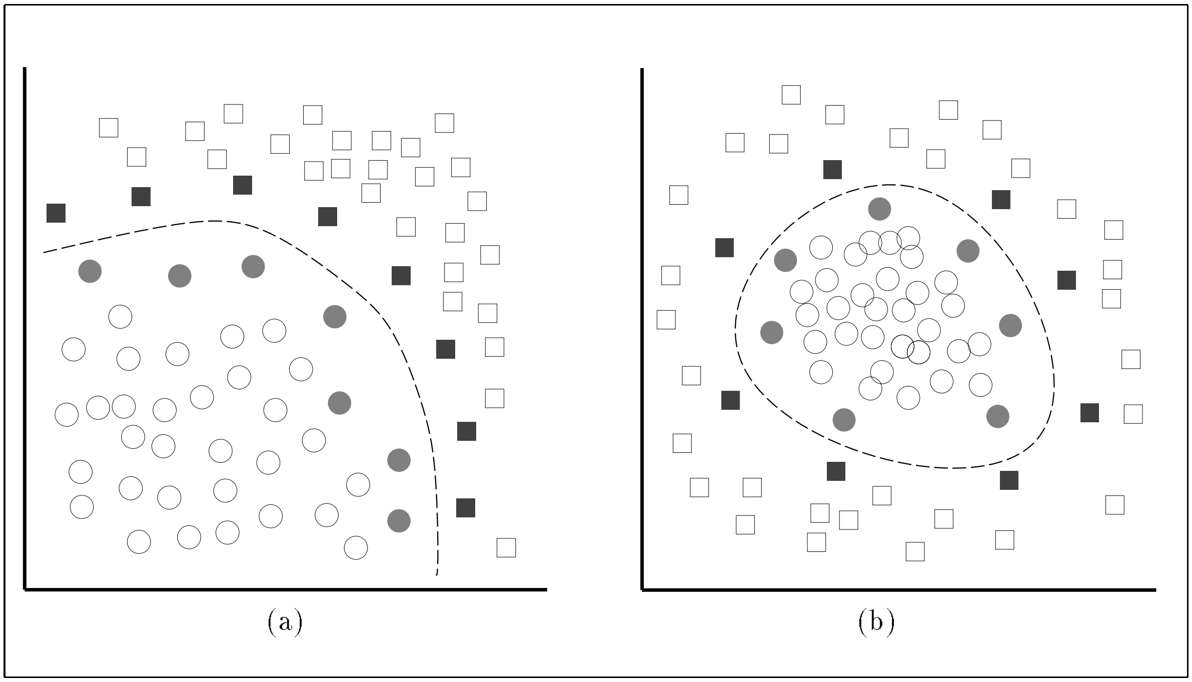
Figure (3) shows examples of the preceding geometrical interpretations with polynomial and RBF classifiers.

2.5 An Interesting Extension: A *Weighted* SVM

The original formulation of the SVM in the existing literature can be extended to handle two frequent cases in pattern classification and recognition:

- An unequal proportion of data samples between the classes.

- A need to *tilt the balance* or *weight* one class versus the other, which is very frequent when a classification error of one type is more expensive or undesirable than other.



(a) (b)

Figure 3: Decision Surfaces given in (a) by a polynomial classifier, and in (b) by a RBF, where the Support Vectors are indicated in dark fill. Notice the reduce number of them and their position close to the boundary. In (b), the Support Vectors are the RBF centers.

The way to derive this extension is to allow equation (37) to be:

$$\begin{array}{llll} \text{Maximize} & F(\mathbf{\Lambda}) & = \mathbf{\Lambda}\cdot\mathbf{1} - \frac{1}{2}\mathbf{\Lambda}\cdot D\mathbf{\Lambda} & \\ \text{subject to} & & & \\ & \mathbf{\Lambda}\cdot\mathbf{y} & = 0 & \\ & \lambda_i & \leq C^{+}\mathbf{1} & \text{for } \; y_i = +1 \\ & \lambda_i & \leq C^{-}\mathbf{1} & \text{for } \; y_i = -1 \\ & \mathbf{\Lambda} & \geq 0 & \end{array} \tag{43}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$, D is a symmetric ℓxℓ matrix with elements $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and C^+ and C^- are positive constants.

Equation (18) for $k = 1$ now becomes:

$$min \;\; \Phi(\mathbf{w}, \Xi) \; = \; \frac{1}{2}\|\mathbf{w}\|^2 + C^{+}(\sum_{i:y_i=+1} \xi_i) + C^{-}(\sum_{i:y_i=-1} \xi_i) \tag{44}$$

and equations (19) and (20) remain unchanged.

The quadratic program (43) can be interpreted as penalizing with higher penalty (C^+ or C^-) the most undesirable type of error through equation (44) . It is also important to notice that this extension has no real impact on the complexity of the problem of finding the optimal vector of multipliers $\mathbf{\Lambda}$, since only the bounding box constraints have changed.

Notice that this extension could be changed even further to allow, for example, higher values of C for highly reliable or valuable data points and lower values for data points of less confidence or value.

3 Training a Support Vector Machine

Solving the quadratic program (37) determines the value of $\mathbf{\Lambda}^*$ and therefore the desired decision surface given by equation (34).This optimization process is referred to as *Training a Support Vector Machine.* This section covers previous, current, and possible future approaches in solving this problem.

One important characteristic of (37) is that the quadratic form matrix D that appears in the objective function (even though symmetric) is completely dense and with size square in the number of data vectors. This fact implies that due to memory and computational constraints, problems with large data sets (above $\approx$ 5,000 samples) cannot be solved without some kind of data and problem decomposition.

Section 3.1 deals with approaches to solving *small* problems, both because they constitute a natural first step, and also because the decomposition algorithm described in section 3.2 iteratively solves *small* sub-problems of the type given by (37).

3.1 Previous Work

The training of a SVM with small data sets was first approached by Vapnik *et al.* [5] using a constrained conjugate gradient algorithm. Briefly described, conjugate gradient ascent was used to explore the feasible region until the step would move the solution outside of it. When that happened, the largest step along the conjugate direction was taken, while maintaining feasibility. Every time a variable λ_i reached 0, the corresponding data point was removed (therefore reducing and approximating the solution) and the conjugate gradient process was re-started.

The next approach taken by Vapnik *et al.* [5], was to adapt to this problem the algorithm for bounded large-scale quadratic programs due to Moré and Toraldo [16]. Originally, this algorithm uses conjugate gradient to explore the face of the feasible region defined by the current iterate, and gradient projection to move to a different face. The main modification was to only consider binding (and therefore *frozen*) those variables that were equal to one of the bounds and for which movement along the gradient would take them outside the feasible region.

During the process of this research, the training problem for small data sets has been approached with two different algorithms and three computer packages: Zoutendijk's method of feasible directions, (using CPLEX to solve the LP's), GAMS/MINOS (using GAMS as the modeling language and MINOS 5.4 as the solver), and a second-order variant of the reduced gradient method (algorithm implemented in MINOS 5.4). A summary of these approaches and some computational results are reported next:

3.1.1 Methods Description

Zoutendijk's Method (case of linear constraints) [29][1]:

In order to solve a nonlinear problem of the form:

$$\begin{array}{lll} \text{Maximize} & f(\mathbf{x}) & \\ \text{subject to} & & \\ & A\mathbf{x} & \leq \mathbf{b} \\ & E\mathbf{x} & = \mathbf{e} \end{array} \tag{45}$$

this method follows the following skeletal approach:

1. Find $\mathbf{x}_1$ with $A_1\mathbf{x}_1 = \mathbf{b}_1$, $A_2\mathbf{x}_1 < \mathbf{b}_2$ and $E\mathbf{x}_1 = \mathbf{e}$, partitioning $A^T = [A_1^T, A_2^T]$ and $\mathbf{b} = [\mathbf{b}_1, \mathbf{b}_2]$. Let $k = 1$.

2. Given $\mathbf{x}_k$, $A_1\mathbf{x}_k = \mathbf{b}_1$ and $A_2\mathbf{x}_k < \mathbf{b}_2$, find $\mathbf{d}_k$, the optimal solution of:

$$\begin{array}{lll} \text{Maximize} & \nabla f(\mathbf{x}_k) \cdot \mathbf{d} & \\ \text{subject to} & & \\ & A_1\mathbf{d} & \leq \mathbf{0} \\ & E\mathbf{d} & = \mathbf{0} \\ & -\mathbf{1} \leq \mathbf{d} & \leq \mathbf{1} \end{array} \tag{46}$$

3. If $\nabla f(\mathbf{x}_k) \cdot \mathbf{d}_k = 0$, Stop. Else go to 4.

4. Find a step-size δ_k, solution to:

$$\begin{array}{lll} \text{Maximize} & f(\mathbf{x}_k + \delta\mathbf{d}) & \\ \text{subject to} & & \\ & 0 \leq \delta & \leq \delta_{max} \end{array} \tag{47}$$

   where

$$\delta_{max} = \begin{cases} \min_{\hat{d}_i > 0}(\frac{\hat{b}_i}{\hat{d}_i}) & \text{if } \ \hat{\mathbf{d}} \not\leq \mathbf{0} \\ \infty & \text{if } \ \hat{\mathbf{d}} \leq \mathbf{0} \end{cases}$$

   with $\hat{\mathbf{b}} = \mathbf{b}_2 - A_2\mathbf{x}_k$ and $\hat{\mathbf{d}} = A_2\mathbf{d}_k$.

5. Let $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{d}_k$. Let $k = k + 1$. Go to step 2.

Step 2 involves solving a linear program, which is usually very easy to .In the case of training a SVM, step 2 becomes:

$$\begin{array}{llll} \text{Maximize} & f(\mathbf{d}) & = (\mathbf{1} - D\mathbf{\Lambda}_k) \cdot \mathbf{d} & \\ \text{subject to} & & & \\ & \mathbf{\Lambda} \cdot \mathbf{d} & = 0 & \\ & -1 \leq d_i & \leq 0 & \text{for } \ \lambda_i = C \\ & 0 \leq d_i & \leq 1 & \text{for } \ \lambda_i = 0 \\ & -1 \leq d_i & \leq 1 & \text{otherwise} \end{array} \tag{48}$$

and step 4 selects $\delta_k = \min(\delta_{opt}, \delta_{max})$, where:

$$\delta_{opt} = \frac{\mathbf{d} \cdot \mathbf{1} - \mathbf{d} \cdot D\mathbf{\Lambda}}{-2\mathbf{d} \cdot D\mathbf{d}} \qquad \text{and} \qquad \lambda_{max} = \min_{d_i \neq 0}\{ \min_{\lambda_i < C, d_i > 0}(\frac{C - \lambda_i}{d_i}), \min_{\lambda_i > 0, d_i < 0}(\frac{-\lambda_i}{d_i})\}$$

One interesting modification that was done to this algorithm in order to help its speed in the computer implementation, was to solve the problem several times with increasing upper bound C. The starting value of C was usually very low, and it was scaled several times until it reached the original value. The solutions were also scaled and used as a starting point for the following iteration.

From a computational point of view, this method behaved a lot better than a *naive* constrained conjugate gradient implementation, both in terms of speed and graceful degradation with the increase of C.

On the other hand, this implementation had serious difficulties in cases where most of the λ_i's were strictly between their bounds. The zigzagging and slow convergence it presented allowed GAMS/MINOS and MINOS 5.4 to outperform it by several orders of magnitude.

GAMS/MINOS:

GAMS is a modeling language that allows fast description and maintainability of optimization problems. As a language, GAMS *generates* the specified model and calls a user-specified *solver*, depending on the type of problem at hand. In the case of nonlinear programs, MINOS is one of these *solvers*.

The work done with GAMS/MINOS was very important. At the beginning, it offered a verification of the implementation of Zoutendijk's method and a point of comparison in terms of speed and accuracy, but most important, it later pointed to the idea of using MINOS 5.4 directly, without the overhead that GAMS could represent.

Another reason for considering important the work done with GAMS/MINOS was the improvement in the training speed due to a problem reformulation. The original problem given by (37) can be rewritten as:

$$
\begin{array}{lll}
\text{Maximize} & F(\mathbf{\Lambda},\mathbf{\Omega}) & = \mathbf{\Lambda}\cdot\mathbf{1} - \frac{1}{2}\mathbf{\Lambda}\cdot\mathbf{\Omega} \\
\quad \mathbf{\Lambda},\mathbf{\Omega} & & \\
\text{subject to} & & \\
 & \mathbf{\Lambda}\cdot\mathbf{y} & = 0 \\
 & D\mathbf{\Lambda} & = \mathbf{\Omega} \\
 & \mathbf{\Lambda} & \leq C\mathbf{1} \\
 & \mathbf{\Lambda} & \geq \mathbf{0}
\end{array} \tag{49}
$$

Although strange at first sight, this transformation allows a much faster function and gradient evaluation, and was responsible for an important speedup in both steps of the solution (model generation and optimization). This was enough reason to use it as the formulation when using MINOS 5.4.

MINOS 5.4:

MINOS 5.4 solves nonlinear problems with linear constraints using Wolfe's Reduced Gradient algorithm in conjunction with Davidson's quasi-Newton method. Details of its implementation are described by Murtagh and Saunders in [17], and MINOS 5.4 User's Guide [18] and Bazaraa *et al.* [1] present an overview with some heuristics and comparisons.

Wolfe's Reduced Gradient method depends upon reducing the dimensionality of the problem by representing all variables in terms of an independent set of them. Under non-degeneracy assumptions (to facilitate this brief description), a program of the form:

$$
\begin{array}{lll}
\text{Minimize} & f(\mathbf{x}) & \\
\text{subject to} & & \\
 & A\mathbf{x} & = \mathbf{b} \\
 & \mathbf{x} & \geq \mathbf{0}
\end{array}
$$

can be decomposed into $A = [B, N]$, $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ with B non-singular, $\mathbf{x}_B > \mathbf{0}$ and $\mathbf{x}_N \geq \mathbf{0}$.

By denoting the gradient $\nabla f(\mathbf{x}) = (\nabla_B f(\mathbf{x}), \nabla_N f(\mathbf{x}))$ and a direction vector $\mathbf{d} = (\mathbf{d}_B, \mathbf{d}_N)$, the system $A\mathbf{d} = \mathbf{0}$ holds for any choice of $\mathbf{d}_N$ by letting $\mathbf{d}_B = -B^{-1}\mathbf{d}_N$.

Defining $\mathbf{r} = (\mathbf{r}_B, \mathbf{r}_N) \equiv \nabla f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} A = (\mathbf{0}, \nabla_N f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} N)$ as the *reduced gradient*, it follows that $\nabla f(\mathbf{x})\cdot\mathbf{d} = \mathbf{r}_N \cdot \mathbf{d}_N$. Therefore, in order to have a feasible direction $\mathbf{d}$ to be an improving feasible direction (feasibility and $\nabla f(\mathbf{x})\cdot\mathbf{d} < 0$), a vector $\mathbf{d}_N$ must be chosen such that $\mathbf{r}_N \cdot \mathbf{d}_N < 0$ and $d_j \geq 0$ for $x_j = 0$. This can be accomplished by choosing $\mathbf{d}_B = -B^{-1}\mathbf{d}_{_N}$ and:

$$
d_{_{N_j}} = \begin{cases} -r_j & \text{if } r_j \leq 0 \\ -x_j r_j & \text{if } r_j > 0 \end{cases}
$$

for $j \in N$.After determining the improving feasible direction $\mathbf{d}$, a line-search procedure is used to determine the step-size, and an improved solution is obtained.

Reduced gradient methods allow all components of $\mathbf{d}_N$ to be non-zero. On the opposite side, for example, the simplex method for linear programming examines a similar direction-finding problem, but allow only one component of $\mathbf{d}_N$ to be non-zero at a time. It is interesting to see that although the second strategy looks too restrictive, the first one also can result in a slow progress, as sometimes only small step-sizes are possible due to the fact that many components are changing simultaneously.

In order to reach a compromise between the two strategies mentioned above, the set of non basic variables $\mathbf{x}_N$ can be further partitioned into $(\mathbf{x}_S, \mathbf{x}_{N'})$, with the corresponding decomposition of $N = [S, N']$ and $\mathbf{d}_N = (\mathbf{d}_S, \mathbf{d}_{N'})$. The variables $\mathbf{x}_s$ are called *superbasic variables*, and are intended to be the *driving force* of the iterates while $\mathbf{x}_{N'}$ is fixed and $\mathbf{x}_B$ is adjusted to maintain feasibility [17].

Notice that the direction vector $\mathbf{d}$ can be accordingly decomposed through a linear operator Z of the form:

$$
\mathbf{d} \;=\; \begin{bmatrix} \mathbf{d}_B \\ \mathbf{d}_S \\ \mathbf{d}_{N'} \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \mathbf{d}_S \;\equiv\; Z\mathbf{d}_S \tag{50}
$$

and now the search direction along the surface of the active constraints is characterized as being in the range of a matrix Z which is orthogonal to the matrix of constraint normals, i.e.,

$$AZ \;=\; [B, S, N'] \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \;=\; 0. \tag{51}$$

By expressing the *directional derivative* $\nabla f(\mathbf{x}) \cdot \mathbf{d}$ as:

$$\nabla f(\mathbf{x}) \cdot \mathbf{d} \;=\; \nabla f(\mathbf{x}) \cdot Z\mathbf{d}_S \;=\; [\nabla_S f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} S]\mathbf{d}_S \;=\; \mathbf{r}_S \cdot \mathbf{d}_S \tag{52}$$

where $\mathbf{r}_S = \nabla_S f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} S$, and the direction finding problem can therefore be reduced to:

$$\begin{array}{ll} \text{Minimize} & \mathbf{r}_S \cdot \mathbf{d}_S \\ \text{subject to} & \\ & -x_j|r_j| \le d_j \le |r_j| \quad \text{for } x_j \text{ superbasic.} \end{array} \tag{53}$$

Given that the direction finding problem described by equation (53) uses a linear approximation to the objective function, slow and zigzagging convergence is likely to happen when the contours of f are *flat* or *thin* in some directions. Therefore, we would expect faster convergence when this approach is upgraded by taking a second-order approximation to f. More formally, the goal is to minimize a second-order approximation to the direction finding problem given by:

$$f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{d} + \frac{1}{2}\mathbf{d} \cdot H(\mathbf{x})\mathbf{d} \tag{54}$$

over the linear manifold $A\mathbf{d} = 0$.
Using equations (52) and (50), (54) transforms into:

$$\min\{\mathbf{r}_S \cdot \mathbf{d}_S + \frac{1}{2}\mathbf{d}_S \cdot Z^T H(\mathbf{x}) Z \mathbf{d}_S\} \tag{55}$$

where the matrix $Z^T H(\mathbf{x}) Z$ is called the *reduced Hessian.*
Setting the gradient of (55) equal to zero results in the system of equations:

$$Z^T H(\mathbf{x}) Z \mathbf{d}_S \;=\; -\mathbf{r}_S \tag{56}$$

Once $\mathbf{d}_S$ is available, a line-search along the direction $\mathbf{d} = Z\mathbf{d}_S$ is performed and a new solution is obtained. MINOS 5.4 implements (56) with certain computational highlights [17]:

1. During the algorithm, if it appears that no more improvement can be made with the current partition $[B, S, N']$, that is, $\|\mathbf{r}_S\| < \varepsilon$, for a suitably chosen tolerance level ε, some of the non-basic variables are added to the superbasics set. Using a *Multiple Pricing* option, MINOS allows the user to specify how many of them to incorporate.

2. If at any iteration a basic or superbasic variable reaches one of its bounds, the variable is made non-basic.

3. The matrices Z, $H(\mathbf{x})$ and $Z^T H(\mathbf{x}) Z$ are never actually computed, but are used implicitly

4. The reduced Hessian matrix $Z^T H(\mathbf{x}) Z$ is approximated by $R^T R$, where R is a dense upper triangular matrix.

5. A sparse LU factorization of the basis matrix B is used.

3.1.2 Computational Results

In order to compare the relative speed between these methods, two different problems with small data-sets were solved in the same computational environment:

1. Training a SVM with a linear classifier in the Ripley data-set. This data-set consists of 250 samples in two dimensions which are not linearly separable. Table 2 shows the following points of comparison:
    - The difference between GAMS/MINOS used in the original problem and in the transformed version (49).
    - The performance degradation suffered by the conjugate gradient implementation under the increase of the upper bound C, and on the opposite hand, the negligible effect on GAMS/MINOS (modified) and MINOS 5.4.
    - A considerable advantage in performance by MINOS 5.4.
2. Training a SVM with a third degree polynomial classifier on the Sonar data-set. This data-set consists of 208 samples in 60 dimensions which are not linearly separable, but are polynomially separable. The results of this experiments are shown in Table 3 and exhibit the following points of comparison:
    - The difficulty experienced by first-order methods like Zoutendijk's method to converge when the values of the λ_i's are strictly between the bounds.
    - The clear advantage in solving the problem directly with MINOS 5.4, removing the overhead created by GAMS and incorporating the knowledge of the problem into the solution process through, for example, fast and exact gradient evaluation, use of symmetry in the constraint matrix, etc.
    - Again, a negligible effect of the upper bound C on the performance, when using MINOS.

An important computational result is the *sub-linear* dependence of the training time with the dimensionality of the input data. In order to show this dependence, Table 4 presents the training time for randomly-generated 2,000 data-points problems, with different dimensionality, separability, and upper bound C.

	Methods				
C	Conjugate Gradient	Zoutendijk	GAMS/MINOS	GAMS/MINOS Modified	MINOS 5.4
10	23.9 sec	12.4 sec	906 sec	17.6 sec	1.2 sec
100	184.1 sec	37.9 sec	1068 sec	19.7 sec	1.4 sec
10000	5762.2 sec	161.5 sec	1458 sec	22.6 sec	2.3 sec

Table 2: Training time on the Ripley data-set for different methods and upper bound C. GAMS/MINOS Modified corresponds to the reformulated version of the problem.

	Methods		
C	Zoutendijk	GAMS/MINOS Modified	MINOS 5.4
10	4381.2 sec	67.0 sec	3.3 sec
100	N/A	67.1 sec	3.3 sec
10000	N/A	67.1 sec	3.3 sec

Table 3: Training time on the Sonar Dataset for different methods and upper bound C.

3.2 A New Approach to Large Database Training

As mentioned before, training a SVM using large data sets (above $\approx$ 5,000 samples) is a very difficult problem to approach without some kind of data or problem decomposition. To give an idea of some memory requirements, an application like the one described later in section 3 involves 50,000 training samples, and this amounts to a quadratic form whose matrix D has $2.5 \cdot 10^9$ entries that would need, using an 8-byte floating point representation, 20,000 Megabytes = 20 Gigabytes of memory!

In order to solve the training problem efficiently, we take explicit advantage of the geometric interpretation introduced in Section 2.4, in particular, the expectation that the number of support vectors will be very few. If we consider the quadratic programming problem given by (37), this expectation translates into having many of the components of $\mathbf{\Lambda}$ equal to zero.

In order to decompose the original problem, one can think of solving iteratively the system given by (37), but keeping fixed at zero level, those components λ_i associated with data points that are not support vectors, and therefore only optimizing over a reduced set of variables.

To convert the previous description into an algorithm we need to specify:

1. **Optimality Conditions**: These conditions allow us to decide computationally, if the problem has been solved optimally at a particular global iteration of the original problem. Section 3.2.1 states and proves optimality conditions for the QP given by (37).

2. **Strategy for Improvement**: If a particular solution is not optimal, this strategy defines a way to improve the cost function and is frequently associated with variables that violate optimality conditions. This strategy will be stated in section 3.2.2.

After presenting optimality conditions and a strategy for improving the cost function, section 3.2.3 introduces a decomposition algorithm that can be used to solve large database training problems, and section 3.2.4 reports some computational results obtained with its implementation.

3.2.1 Optimality Conditions

In order to be consistent with common standard notation for nonlinear optimization problems, the quadratic program (37) can be rewritten in minimization form as:

$$\begin{array}{llll} \text{Minimize} & W(\mathbf{\Lambda}) & = -\mathbf{\Lambda}\cdot\mathbf{1} + \frac{1}{2}\mathbf{\Lambda}\cdot D\mathbf{\Lambda} & \\ \mathbf{\Lambda} & & & \\ \text{subject to} & & & \\ & \mathbf{\Lambda}\cdot\mathbf{y} & = 0 & (\mu) \\ & \mathbf{\Lambda} - C\mathbf{1} & \leq \mathbf{0} & (\mathbf{\Upsilon}) \\ & -\mathbf{\Lambda} & \leq \mathbf{0} & (\mathbf{\Pi}) \end{array} \tag{57}$$

where μ, $\mathbf{\Upsilon} = (\upsilon_1, \ldots, \upsilon_\ell)$ and $\mathbf{\Pi} = (\pi_1, \ldots, \pi_\ell)$ are the associated Kuhn-Tucker multipliers.

Since D is a positive semi-definite matrix (see end of section 2.3.3) and the constraints in (57) are linear, the Kuhn-Tucker (KT) conditions are necessary and sufficient for optimality, and they are:

	Dimension					
	Separable			Non-Separable		
C	4	16	256	4	16	256
10	60.7 sec	106.4 sec	613.5 sec	292.9 sec	476.0 sec	1398.2 sec
100	36.0 sec	69.2 sec	613.7 sec	313.5 sec	541.0 sec	2369.4 sec
10000	21.8 sec	56.2 sec	623.0 sec	327.4 sec	620.6 sec	3764.1 sec

Table 4: Training time on a Randomly-generated Dataset for different dimensionality and upper bound C.

$$
\begin{array}{ll}
\nabla W(\mathbf{\Lambda}) + \mathbf{\Upsilon} - \mathbf{\Pi} + \mu\mathbf{y} & = \mathbf{0} \\
\mathbf{\Upsilon} \cdot (\mathbf{\Lambda} - C\mathbf{1}) & = 0 \\
\mathbf{\Pi} \cdot \mathbf{\Lambda} & = 0 \\
& \\
\mathbf{\Upsilon} & \geq \mathbf{0} \\
\mathbf{\Pi} & \geq \mathbf{0} \\
& \\
\mathbf{\Lambda} \cdot \mathbf{y} & = 0 \\
\mathbf{\Lambda} - C\mathbf{1} & \leq \mathbf{0} \\
-\mathbf{\Lambda} & \leq \mathbf{0}
\end{array}
\tag{58}
$$

In order to derive further algebraic expressions from the optimality conditions(58) , we assume the existence of some λ_i such that $0 < \lambda_i < C$ (see end of section 2.3.2), and consider the three possible values that each component of $\mathbf{\Lambda}$ can have:

1. **Case:** $0 < \lambda_i < C$**:**

   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 + \mu y_i = 0 \tag{59}$$

   Noticing that

$$(D\mathbf{\Lambda})_i = \sum_{j=1}^{\ell} \lambda_j y_j y_i K(\mathbf{x}_i, \mathbf{x}_j) \;=\; y_i \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

   and that for $0 < \lambda_i < C$,

$$f(\mathbf{x}_i) \;=\; sign(\sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b) \;=\; \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b \;=\; y_i$$

   we obtain the following:

$$(D\mathbf{\Lambda})_i \;=\; \frac{y_i - b}{y_i} \;=\; 1 - \frac{b}{y_i} \tag{60}$$

   By substituting (60) into (59) we finally obtain that

$$\mu = b \tag{61}$$

   Therefore, at an optimal solution $\mathbf{\Lambda}^*$, the value of the multiplier μ is equal to the optimal threshold b^*.

2. **Case:** $\lambda_i = C$**:**
   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 + \upsilon_i + \mu y_i = 0 \tag{62}$$

   By defining

$$g(\mathbf{x}_i) = \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b$$

and noticing that

$$(D\mathbf{\Lambda})_i = y_i \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \;=\; y_i g(\mathbf{x}_i) - y_i b$$

equation (62) can be written as:

$$y_i g(\mathbf{x}_i) - y_i b - 1 + v_i + \mu y_i = 0$$

By combining $\mu = b$ (derived from case 1) and requiring $v_i \geq 0$ we finally obtain:

$$y_i g(\mathbf{x}_i) \;\leq\; 1 \tag{63}$$

3. **Case:** $\lambda_i = 0$**:**
   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 - \pi_i + \mu y_i = 0 \tag{64}$$

   By applying a similar algebraic manipulation as the one described for case 2, we obtain

$$y_i g(\mathbf{x}_i) \;\geq\; 1 \tag{65}$$

3.2.2 Strategy for Improvement

In order to incorporate the optimality conditions and the expectation that most λ_i's will be zero into an algorithm, we need to derive a way to improve the objective function value using this information. To do this, let us decompose $\mathbf{\Lambda}$ in two vectors $\mathbf{\Lambda}_B$ and $\mathbf{\Lambda}_N$, where $\mathbf{\Lambda}_N = 0$, and B and N partition the index set, and that the optimality conditions hold in the subproblem defined only for the variables in B. In further sections, the set B will be referred to as the *working set*. Under this decomposition the following statements are clearly true:

- We can replace $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$, $j \in N$, without changing the cost function or the feasibility of both the subproblem and the original problem.
- After such a replacement, the new subproblem is optimal if and only if $y_j g(\mathbf{x}_j) \;\geq\; 1$. This follows from equation (65) and the assumption that the subproblem was optimal before the replacement was done.

The previous statements suggest that replacing variables at zero levels in the subproblem, with variables $\lambda_j = 0$, $j \in N$ that violate the optimality condition $y_j g(\mathbf{x}_j) \;\geq\; 1$, yields a subproblem that, when optimized, improves the cost function while maintaining feasibility. The following proposition states this idea formally.

Proposition: Given an optimal solution of a subproblem defined on B, the operation of replacing $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$, $j \in N$, satisfying $y_j g(\mathbf{x}_j) \;<\; 1$ generates a new subproblem that when optimized, yields a strict improvement of the objective function $W(\mathbf{\Lambda})$.

Proof: We assume again the existence of λ_p such that $0 < \lambda_p < C$. Let us also assume without loss of generality that $y_p = y_j$ (the proof is analogous if $y_p = -y_j$). Then, there is some $\epsilon > 0$ such that $\lambda_p - \delta > 0$, for $\delta \in (0, \epsilon)$. Notice also that $g(\mathbf{x}_p) \;=\; y_p$. Now, consider $\overline{\mathbf{\Lambda}} = \mathbf{\Lambda} + \delta e_j - \delta e_p$, where e_j and e_p are the jth and pth unit vectors, and notice that the pivot operation can be handled implicitly by letting $\delta > 0$ and by holding $\lambda_i = 0$. The new cost function $W(\overline{\mathbf{\Lambda}})$ can be written as:

$$\begin{aligned}
W(\overline{\boldsymbol{\Lambda}}) &= -\overline{\boldsymbol{\Lambda}} \cdot 1 + \frac{1}{2}\overline{\boldsymbol{\Lambda}} \cdot D\overline{\boldsymbol{\Lambda}} \\
& \quad -\boldsymbol{\Lambda} \cdot 1 + \frac{1}{2}\left[\boldsymbol{\Lambda} \cdot D\boldsymbol{\Lambda} + 2\boldsymbol{\Lambda} \cdot D(\delta e_j - \delta e_p) + (\delta e_j - \delta e_p) \cdot D(\delta e_j - \delta e_p)\right] \\
& \quad W(\boldsymbol{\Lambda}) + \delta\left[\frac{g(\mathbf{x}_j) - b}{y_j} - 1 + \frac{b}{y_p}\right] + \frac{\delta^2}{2}\left[K(\mathbf{x}_j, \mathbf{x}_j) + K(\mathbf{x}_p, \mathbf{x}_p) - 2y_p y_j K(\mathbf{x}_p, \mathbf{x}_j)\right] \\
& \quad W(\boldsymbol{\Lambda}) + \delta\left[g(\mathbf{x}_j)y_j - 1\right] + \frac{\delta^2}{2}\left[K(\mathbf{x}_j, \mathbf{x}_j) + K(\mathbf{x}_p, \mathbf{x}_p) - 2y_p y_j K(\mathbf{x}_p, \mathbf{x}_j)\right]
\end{aligned}$$

Therefore, since $g(\mathbf{x}_j)y_j < 1$, by choosing δ small enough we have $W(\overline{\boldsymbol{\Lambda}}) < W(\boldsymbol{\Lambda})$.
q.e.d

3.2.3 The Decomposition Algorithm

Suppose we can define a fixed-size working set B, such that $|B| \leq \ell$, and it is big enough to contain all support vectors ($\lambda_i > 0$), but small enough such that the computer can handle it and optimize it using some solver. Then the decomposition algorithm can be stated as follows:

1. Arbitrarily choose $|B|$ points from the data set.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that $g(\mathbf{x}_j)y_j < 1$, where

   $$g(\mathbf{x}_j) = \sum_{p=1}^{\ell} \lambda_p y_p K(\mathbf{x}_j, \mathbf{x}_p) + b$$

   replace $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$ and solve the new subproblem.

Notice that this algorithm will strictly improve the objective function at each iteration and therefore will not cycle. Since the objective function is bounded ($W(\boldsymbol{\Lambda})$ is convex and quadratic, and the feasible region is bounded), the algorithm must converge to the global optimal solution in a finite number of iterations. Figure 4 gives a geometric interpretation of the way the decomposition algorithm allows the redefinition of the separating surface by adding points that violate the optimality conditions.

3.2.4 Computational Implementation and Results

We have implemented the decomposition algorithm using the transformed problem defined by equation (49) and MINOS 5.4 as the solver.
Notice that the decomposition algorithm is rather flexible about the pivoting strategy, that is, the way it decides which and how many new points to incorporate into the working set B. Our implementation uses two parameters to define the desired strategy:

- **Lookahead**: this parameter specifies the maximum number of data points the pricing subroutine should use to evaluate optimality conditions (Case 3). If Lookahead data points have been examined without finding a violating one, the subroutine continues until it finds the first one , or until all data points have been examined. In the latter case, global optimality has been obtained.
- **Newlimit**: this parameter limits the number of new points to be incorporated into the working set B.

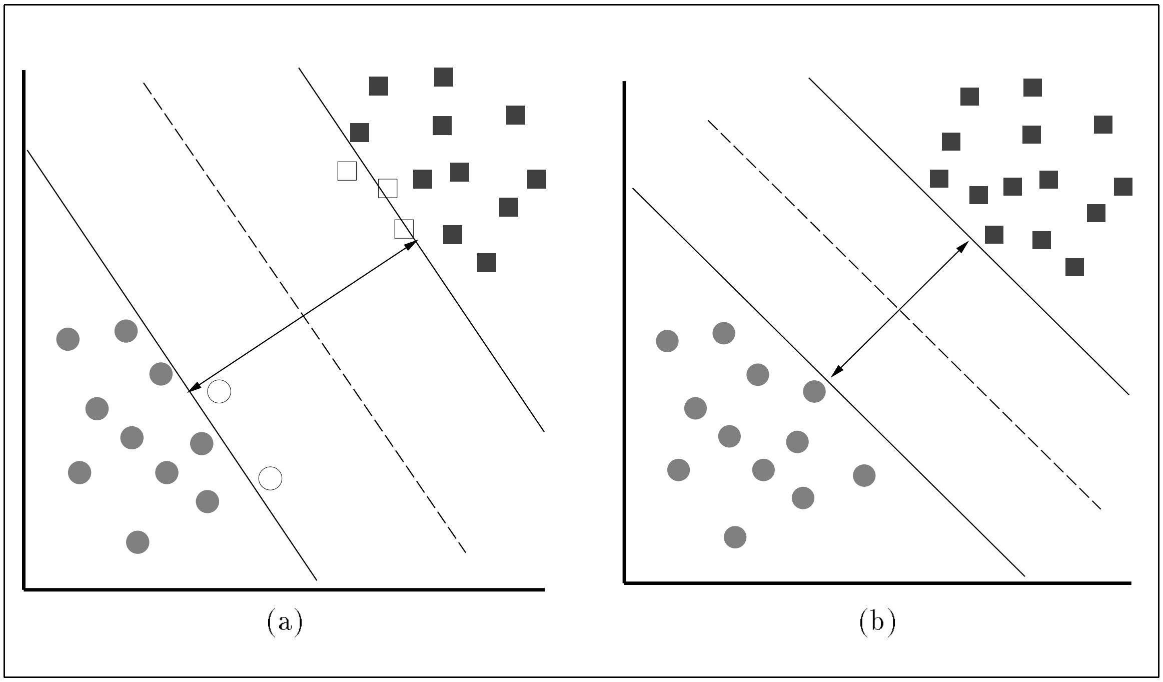


Figure 4: (a) A sub-optimal solution where the non-filled points have $\lambda = 0$ but are violating optimality conditions by being inside the ± 1 area. (b) The decision surface gets redefined. Since no points with $\lambda = 0$ are inside the ± 1 area, the solution is optimal. Notice that the size of the margin has decreased, and the *shape* of the decision surface has changed.

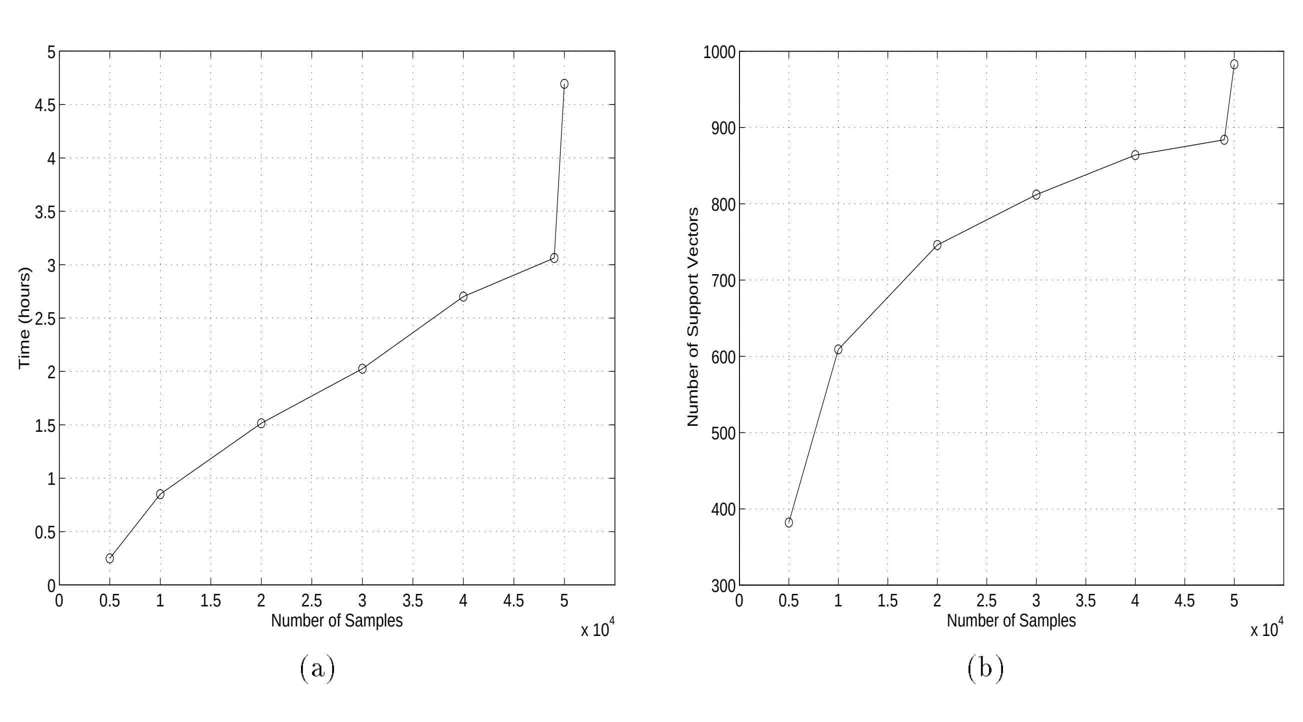
The computational results that we present in this section have been obtained using real data from our Face Detection System, which is described in Section 4.

Figure 5 shows the training time and the number of support vectors obtained when training the system with 5,000, 10,000, 20,000, 30,000, 40,000, 49,000, and 50,000 data points. We must emphasize that the last 1,000 data points were collected in the last phase of bootstrapping of the Face Detection System, and therefore make the training process *harder*, since they correspond to errors obtained with a system that was already very accurate. Figure 6 shows the relationship between the training time and the number of support vectors, as well as the number of global iterations (the number of times the decomposition algorithm calls the solver). Notice the *smooth* relation between the number of support vectors and the training time, and the jump from 11 to 15 global iterations as we go from 49,000 to 50,000 samples. This increase is responsible for the increase in the training time. The system, using a working set of 1200 variables was able to solve the 50,000 data points problem using only 25Mb of RAM.

Figure 7 shows the effect on the training time due to the parameter Newlimit and the size of the working set, using 20,000 data points. Notice the clear improvement as Newlimit is increased. This improvement suggests that in some way, the faster new violating data points are brought into the working set, the faster the decision surface is defined, and optimality is reached. Notice also that, if the working set is too small or too big compared to the number of support vectors (746 in the case of 20,000 samples), the training time increases. In the first case, this happens because the algorithm can only incorporate new points slowly, and in the second case, this happens because the solver takes longer to solve the subproblem as the size of the working set increases.

3.3 Improving the Training of SVM: Future Directions

The algorithm described in Section 3.2.3 suggests two main areas where improvements can be made through future research. These two areas are:



(a) (b)

Figure 5: (a) Training Time on a SPARCstation-20. (b) Number of Support Vectors obtained after Training

1. **The Solver**: The second-order variant of the reduced gradient method implemented by MINOS 5.4 has given very good results so far in terms of accuracy, robustness and performance. However, this method is a general nonlinear optimization method that is not designed in particular for quadratic programs, and in the case of SVM's, is not designed in particular for the special characteristics of the problem. Having as a reference the experience obtained with MINOS 5.4, new approaches to a *tailored* solver through, for example, projected Newton [2] or interior point methods [7], should be attempted.

   At this point it is not clear whether the same type of algorithm is appropriate for all stages of the solution process. To be more specific, it could happen that an algorithm performs well with few non-zero variables at early stages, and then is outperformed by others when the number of non-zero variables reaches some threshold. In particular, we learned that the number of non-zero variables that satisfy $0 < \lambda_i < C$ has an important effect on the performance of the solver.

2. **The Pivoting Strategy**: This area offers great potential for improvements through some ideas we plan to implement. The improvements are based on some qualitative characteristics of the training process that have been observed:

   - During the execution of the algorithm, as much as 40% of the computational effort is dedicated to the evaluation of the optimality conditions. At final stages, it is common to have all the data points evaluated, yet only to collect very few of them to incorporate them to the working set.
   - Only a small portion of the input data is ever brought into the working set (about 16% in the case of the face detection application).
   - Out of the samples that ever go into the working set, about 30% of them enter and exit this set at least once. These vectors are responsible for the first characteristic mentioned above.

   Possible future strategies that exploit these characteristics are:

   - Keep a list or file with all or part of the input vectors that have exited the working set. At the pricing stage, when the algorithm computes the optimality conditions, evaluate these data

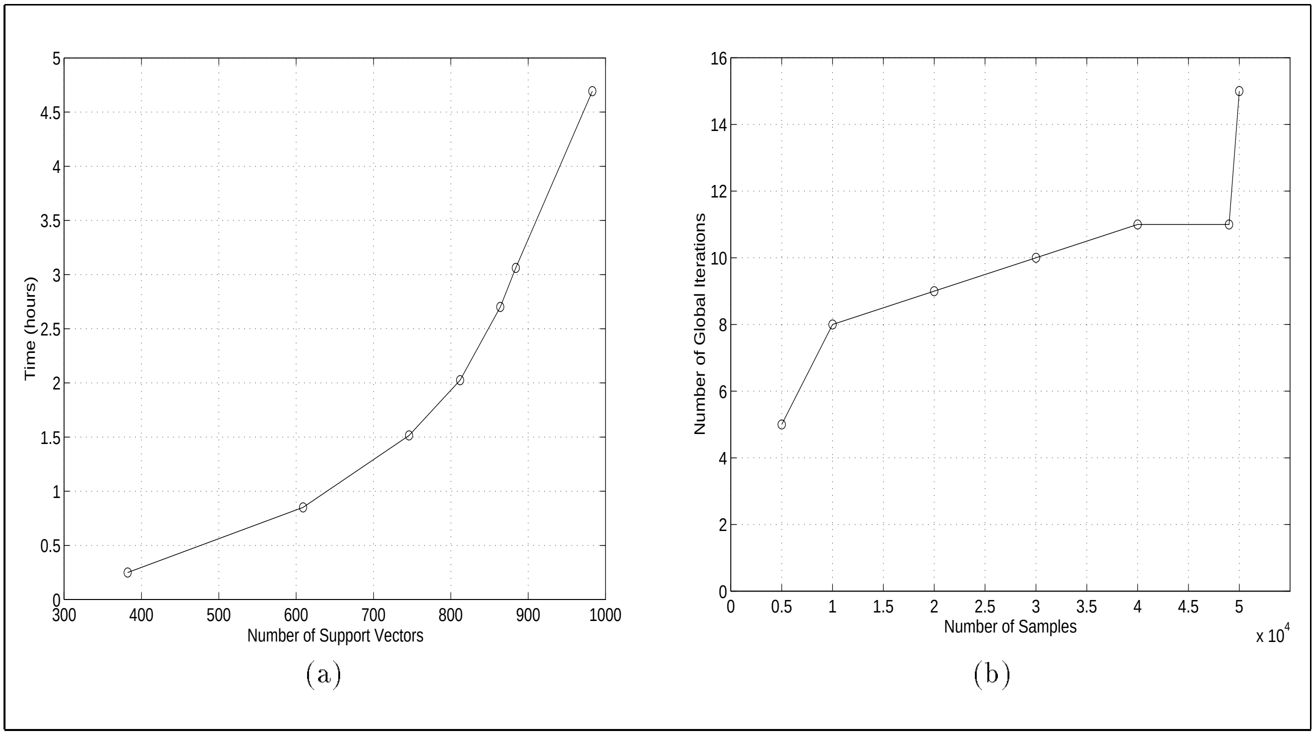


Figure 6: (a) Number of Support Vectors versus Training Time on a SPARCstation-20. Notice how the Number of support vectors is a better indicator of the increase in training time than the number of samples alone. (b) Number of global iterations performed by the algorithm. Notice the increase experimented when going from 49000 to 50000 samples. This increase in the number of iterations is responsible for the increase in the training time

poits before other data points to determine the entering vectors. This strategy is analogous to one sometimes used in the revised simplex method where the algorithm keeps track of the basic variables that have become non-basic. In the case of training of SVM's, the geometric interpretation of this heuristic is to think that if a point was a support vector at some iteration, it was more or less close to the boundary between the classes, and as this boundary is refined or *fine-tuned*, it is possible for it to switch from active to non-active several times. This heuristic could be combined with main memory and cache management policies used in computer systems.

- During the pricing stage, instead of bringing into the working set the first k points that violate optimality conditions, we could try to determine r violating data points, with $r > k$ and choose from these the k *most violating* points. This is done under the geometric idea that the *most violating* points help in defining the decision surface faster and therefore save time in future iterations.
- These last two approaches can be combined by keeping track not only of the points exiting the working set, but also of the remaining violating data points as well.

So far in the description and implementation of the decomposition algorithm, we have assumed that enough memory is available to solve a working set problem that contains all of the support vectors. However, some applications may require more support vectors than the available memory can manage. One possible approach that can be taken is to approximate the optimal solution by the best solution that can be obtained with the current working set size. The present algorithm and implementation can be easily extended to handle this situation by replacing support vectors with $0 < \lambda_i < C$ with new data points. Other more complex approaches that can be pursued to obtain an optimal solution should be the subject of future research.

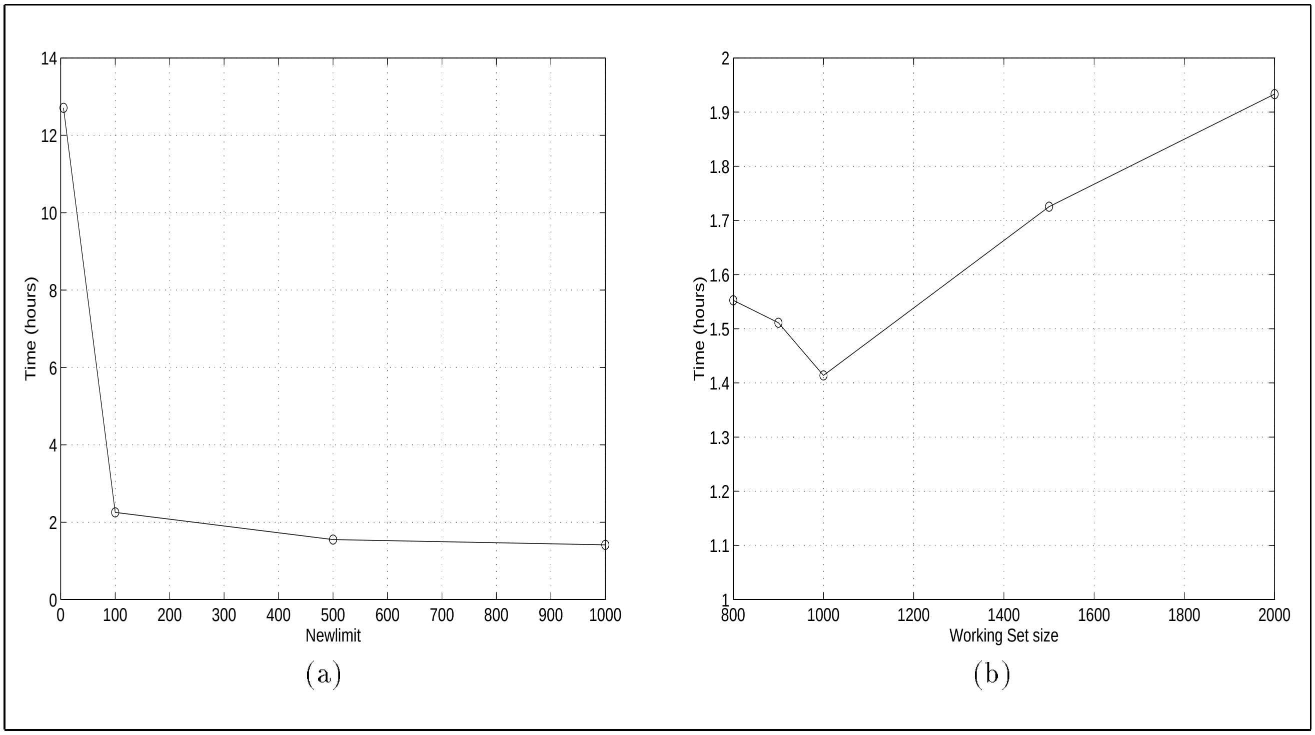


Figure 7: (a) Training time for 20,000 samples with different values of Newlimit, using a working set of size 1000 and Lookahead=10,000. (b) Training time for 20,000 samples with different sizes of the working set, using Newlimit=size of the working set, and Lookahead=10,000.

4 SVM Application: Face Detection in Images

This section introduces a Support Vector Machine application for detecting vertically oriented and unoccluded frontal views of human faces in grey level images. It handles faces over a wide range of scales and works under different lighting conditions, even with moderately strong shadows.

The face detection problem can be defined as follows: Given as input an arbitrary image, which could be a digitized video signal or a scanned photograph, determine whether or not there are any human faces in the image, and if there are, return an encoding of their location. The encoding in this system is to fit each face in a bounding box defined by the image coordinates of the corners.

Face detection as a computer vision task has many applications. It has direct relevance to the face recognition problem, because the first important step of a fully automatic human face recognizer is usually identifying and locating faces in an unknown image. Face detection also has potential application in human-computer interfaces, surveillance systems, census systems, etc.

From the standpoint of this paper, face detection is interesting because it is an example of a natural and challenging problem for demonstrating and testing the potentials of Support Vector Machines. There are many other object classes and phenomena in the real world that share similar characteristics, for example, tumor anomalies in MRI scans, structural defects in manufactured parts, etc. A successful and general methodology for finding faces using SVM's should generalize well for other spatially well-defined pattern and feature detection problems.

It is important to remark that face detection, like most object detection problems, is a difficult task due to the significant pattern variations that are hard to parameterize analytically. Some common sources of pattern variations are facial appearance, expression, presence or absence of common structural features, like glasses or a moustache, light source distribution, shadows, etc.

This system works by testing candidate image locations for local patterns that appear like faces using a classification procedure that determines whether or not a given local image pattern is a face or not.

Therefore, the face detection problem is approached as a classification problem given by examples of 2 classes: faces and non-faces.

4.1 Previous Systems

The problem of face detection has been approached with different techniques in the last few years. This techniques include Neural Networks [4] [19], detection of face features and use of geometrical constraints [27], density estimation of the training data [14], labeled graphs [12] and clustering and distribution-based modeling [22][21].

Out of all these previous works, the results of Sung and Poggio [22][21], and Rowley *et al.* [19] reflect systems with very high detection rates and low false positive detection rates.

Sung and Poggio use clustering and distance metrics to model the distribution of the face and non-face manifold, and a Neural Network to classify a new pattern given the measurements. The key of the quality of their result is the clustering and use of combined Mahalanobis and Euclidean metrics to measure the distance from a new pattern and the clusters. Other important features of their approach is the use of non-face clusters, and the use of a bootstrapping technique to collect important non-face patterns. One drawback of this technique is that it does not provide a principled way to choose some important free parameters like the number of clusters it uses.

Similarly, Rowley *et al.* have used problem information in the design of a retinally connected Neural Network that is trained to classify faces and non-faces patterns. Their approach relies on training several NN emphasizing subsets of the training data, in order to obtain different sets of weights. Then, different schemes of arbitration between them are used in order to reach a final answer.

The approach to the face detection system with a SVM uses no prior information in order to obtain the decision surface, this being an interesting property that can be exploited in using the same approach for detecting other objects in digital images.

4.2 The SVM Face Detection System

This system, as it was described before, detects faces by exhaustively scanning an image for face-like patterns at many possible scales, by dividing the original image into overlapping sub-images and classifying them using a SVM to determine the appropriate class, that is, face or non-face. Multiple scales are handled by examining windows taken from scaled versions of the original image.

Clearly, the major use of SVM's is in the classification step, and it constitutes the most critical and important part of this work. Figure 9 gives a geometrical interpretation of the way SVM's work in the context of face detection.

More specifically, this system works as follows:

1. A database of face and non-face 19×19 pixel patterns, assigned to classes +1 and -1 respectively, is trained on, using the support vector algorithm. A 2nd-degree polynomial kernel function and an upper bound $C = 200$ are used in this process obtaining a perfect training error.

2. In order to compensate for certain sources of image variation, some preprocessing of the data is performed:

   - **Masking:** A binary pixel mask is used to remove some pixels close to the boundary of the window pattern allowing a reduction in the dimensionality of the input space from 19×19=361 to 283. This step is important in the reduction of background patterns that introduce unnecessary noise in the training process.
   - **Illumination gradient correction:** A best-fit brightness plane is subtracted from the unmasked window pixel values, allowing reduction of light and heavy shadows.
   - **Histogram equalization:** A-histogram equalization is performed over the patterns in order to compensate for differences in illumination brightness, different cameras response curves, etc.

3. Once a decision surface has been obtained through training, the run-time system is used over images that do not contain faces, and misclassifications are stored so they can be used as negative examples in subsequent training phases. Images of landscapes, trees, buildings, rocks, etc., are good sources of false positives due to the many different *textured* patterns they contain. This bootstrapping step, which was successfully used by Sung and Poggio [22] is very important in the context of a face detector that learns from examples because:

   - Although negative examples are abundant, negative examples that are useful from a learning point of view are very difficult to characterize and define.
   - By approaching the problem of object detection, and in this case of face detection, by using the paradigm of binary pattern classification, the two classes, object and non-object are not equally complex since the non-object class is broader and richer, and therefore needs more examples in order to get an accurate definition that separates it from the object class. Figure 8 shows an image used for bootstrapping with some misclassifications, that were later used as negative examples.

4. After training the SVM, we incorporate it as the classifier in a run-time system very similar to the one used by Sung and Poggio [22][21] that performs the following operations:

   - Re-scale the input image several times.
   - Cut 19×19 window patterns out of the scaled image.
   - Preprocess the window using masking, light correction and histogran equalization.
   - Classify the pattern using the SVM.
   - If the class corresponds to a face, draw a regtangle aroung the face in the output image.

4.2.1 Experimental Results

To test the run-time system, we used two sets of images. The set A, contained 313 high-quality images with same number of faces. The set B, contained 23 images of mixed quality, with a total of 155 faces. Both sets were tested using our system and the one by Sung and Poggio [22][21]. In order to give true meaning to the number of false positives obtained, it is important to state that set A involved 4,669,960 pattern windows, while set B 5,383,682. Table 5 shows a comparison between the 2 systems.

	Test Set A		Test Set B	
	Detection Rate	False Detections	Detection Rate	False Detections
Ideal System	100 %	0	100%	0
SVM	97.12 %	4	74.19%	20
Sung and Poggio	94.57 %	2	74.19%	11

Table 5: Performance of the SVM face detection system

Figures 10, 11, 12, 13, 14 and 15 present some output images of our system. These images were not used during the training phase of the system.

4.3 Future Directions in Face Detection and SVM Applications

Future research in SVM application can be divided into three main categories or topics:

1. **Simplification of the SVM:** One drawback for using SVM in some real-life applications is the large number of arithmetic operations that are necessary to classify a new input vector. Usually, this number is proportional to the dimension of the input vector and the number of support vectors obtained. In the case of face detection, for example, this is ≈ 283,000 multiplications per pattern!

   The reason behind this overhead is in the roots of the technique, since SVM's define the decision surface by explicitly using data points. This situation causes a lot of redundancy in most cases, and can be solved by *relaxing* the constraint of using data points to define the decision surface. This is a topic of current research conducted by Burges [6] at Bell Laboratories, and it is of great interest for us, because in order to simplify the set of support vectors, one needs to solve highly-nonlinear constrained optimization problems.

Since a closed form solution exists for the case of kernel functions that are 2nd. degree polynomials, we are using a simplified SVM [6] in our current experimental face detection system that gains an acceleration factor of 20, without degrading the quality of the classifications.

2. **Detection of other objects:** We are interested in using SVM's to detect other objects in digital images, like cars, airplanes, pedestrians, etc. Notice that most of these objects have very different appearance, depending on the viewpoint. In the context of face detection, an interesting extension that could lead to better understanding and approach to other problems, is the detection of tilted and rotated faces. It is not clear at this point, whether these different *view* of the same object can be treated with a single classifier, or if they should be treated separately.

3. **Use of multiple classifiers:** The use of multiple classifiers offers possibilities that can be faster and/or more accurate. Rowley *et al.* [19] have successfully combined the output from different neural networks by means of different schemes of arbitration in the face detection problem. Sung and Poggio [22][21] use a first classifier that is very fast as a way to quickly discard patterns that are *clearly* non-faces. These two references are just examples of the combination of different classifiers to produce better systems.

   Our current experimental face detection system performs an initial quick-discarding step using a SVM trained to separate *clearly* non-faces from *probable* faces using just 14 averages taken from different areas of the window pattern. This classification can be done about 300 times faster and is currently discarding more than 99% of input patterns. More work will be done in the near future in this area.

   The classifiers to be combined do not have to be of the same kind. An interesting type of classifier that we will consider is Discriminant Adaptative Nearest Neighbor due to Hastie *et al.* [11][10].

5 Conclusions

In this paper we presented a novel decomposition algorithm to train Support Vector Machines. We have successfully implemented this algorithm and solved large dataset problems using acceptable amounts of computer time and memory. Work in this area can be continued, and we are currently studying new techniques to improve the performance and quality of the training process.

The Support Vector Machine is a very new technique, and, as far as we know, this paper presents the second problem-solving application to use SVM, after Vapnik *et al.* [5, 8, 24] used it in the character recognition problem, in 1995.

Our Face Detection System performs as well as other state-of-the-art systems, and has opened many interesting questions and possible future extensions. From the object detection point of view, our ultimate goal is to develop a general methodology that extends the results obtained with faces to handle other objects. From a broader point of view, we also consider interesting the use of the function approximation or regression extension that Vapnik [24] has done with SVM, in many different areas where Neural Networks are currently used.

Finally, another important contribution of this paper is the application of OR-based techniques to domains like Statistical Learning and Artificial Intelligence. We believe that in the future, other tools like duality theory, interior point methods and other optimization techniques and concepts, will be useful in obtaining better algorithms and implementations with a solid mathematical background.

References

[1] M. Bazaraa, H. Sherali, and C. Shetty. *NonLinear Programming Theory and Algorithms.* J. Wiley, New York, 2nd edition, 1993.

[2] D. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20(2):221–246, March 1982.

[3] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifier. In *Proc. 5th ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992.

[4] G. Burel and D. Carel. Detection and localization of faces on digital images. *Pattern Recognition Letters*, 15:963–967, 1994.

[5] C. Burges and V. Vapnik. A new method for constructing artificial neural networks. Technical report, AT&T Bell Laboratories, 101 Crawfords Corner Road Holmdel NJ 07733, May 1995.

[6] C.J.C. Burges. Simplified support vector decision rules, 1996.

[7] T. Carpenter and D. Shanno. An interior point method for quadratic programs based on conjugate projected gradients. *Computational Optimization and Applications*, 2(1):5–28, June 1993.

[8] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:1–25, 1995.

[9] F. Girosi. An equivalence between sparse approximation and Support Vector Machines. A.I. Memo 1606, MIT Artificial Intelligence Laboratory, 1997. (available at the URL: http://www.ai.mit.edu/people/girosi/svm.html).

[10] T. Hastie, A. Buja, and R. Tibshirani. Penalized discriminant analysis. Technical report, Statistics and Data Analysis Research Department, AT&T Bell Laboratories, Murray Hill, New Jersey, May 1994.

[11] T. Hastie and R. Tibshirani. Discriminat adaptative nearest neighbor classification. Technical report, Department of Statistics and Division of Biostatistics, Stanford University, December 1994.

[12] N. Krüger, M. Pötzsch, and C. v.d. Malsburg. Determination of face position and pose with learned representation based on labled graphs. Technical Report 96-03, Ruhr-Universität, January 1996.

[13] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[14] B. Moghaddam and A. Pentland. Probabilistic visual learning for object detection. Technical Report 326, MIT Media Laboratory, June 1995.

[15] E.H. Moore. On properly positive Hermitian matrices. *Bull. Amer. Math. Soc.*, 23(59):66–67, 1916.

[16] J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optimization*, 1(1):93–113, February 1991.

[17] B. Murtagh and M. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[18] B. Murtagh and M. Saunders. *MINOS 5.4 User's Guide.* System Optimization Laboratory,Stanford University, Feb 1995.

[19] H. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. Technical Report CMU-CS-95-158R, School of Computer Science, Carnegie Mellon University, November 1995.

[20] J. Stewart. Positive definite functions and generalizations, an historical survey. *Rocky Mountain J. Math.*, 6:409–434, 1976.

[21] K. Sung. *Learning and Example Selection for Object and Pattern Detection.* PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Bilogical and Computational Learning, December 1995.

[22] K. Sung and T. Poggio. Example-based learning for view-based human face detection. *A.I. MEMO 1521, C.B.C.L Paper 112*, December 1994.

[23] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

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

[25] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequences of events to their probabilities. *Th. Prob. and its Applications*, 17(2):264–280, 1971.

[26] V.N. Vapnik and A. Ya. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.

[27] G. Yang and T. Huang. Human face detection in a complex background. *Pattern Recognition*, 27:53–63, 1994.

[28] W.Y. Young. A note on a class of symmetric functions and on a theorem required in the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[29] G. Zoutendijk. *Methods of Feasible Directions.* D. Van Norstrand, Princeton, N.J., 1960.

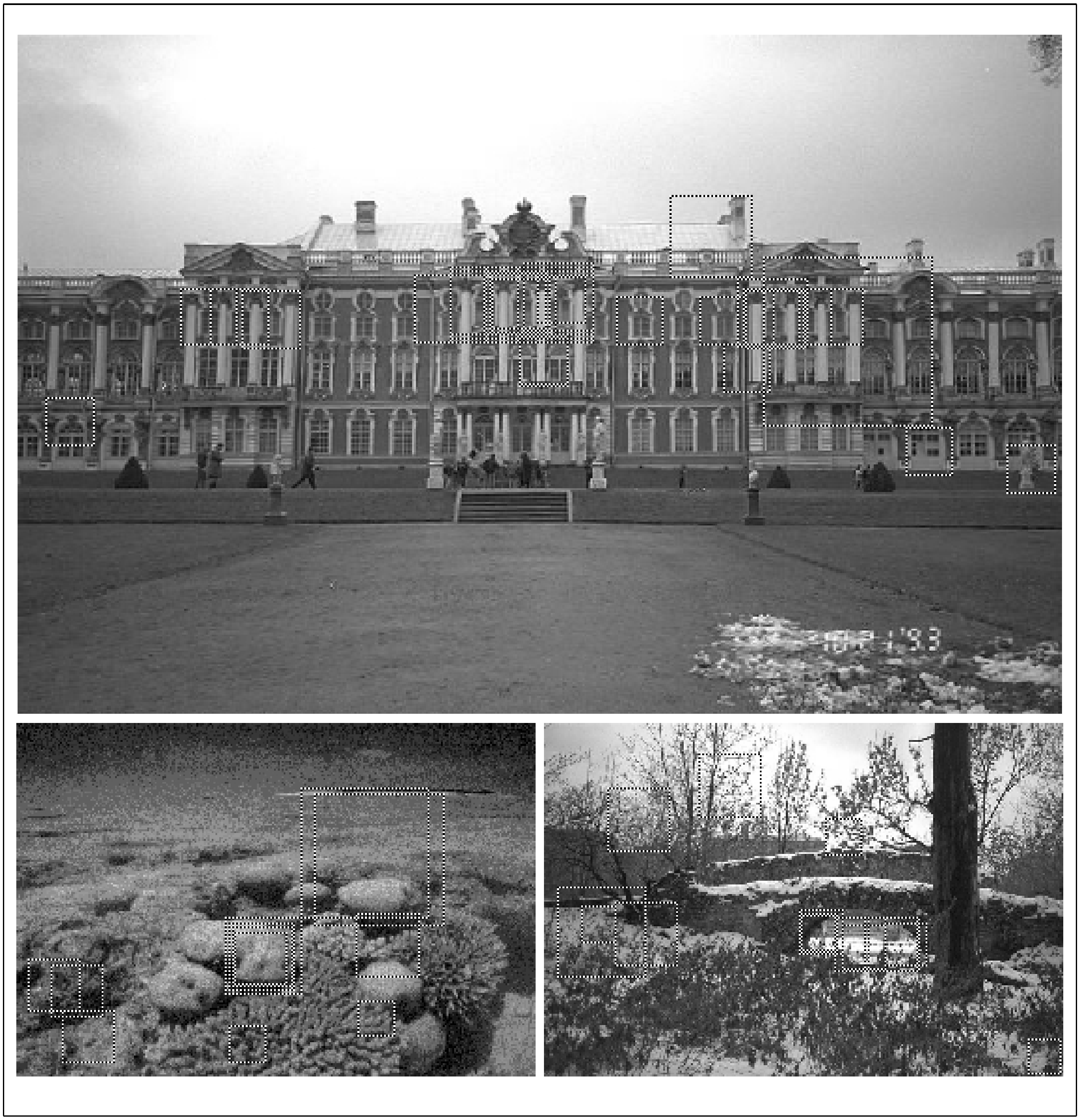


Figure 8: Some false detections obtained with the first version of the system. This false positives were later used as negative examples (class -1) in the training process

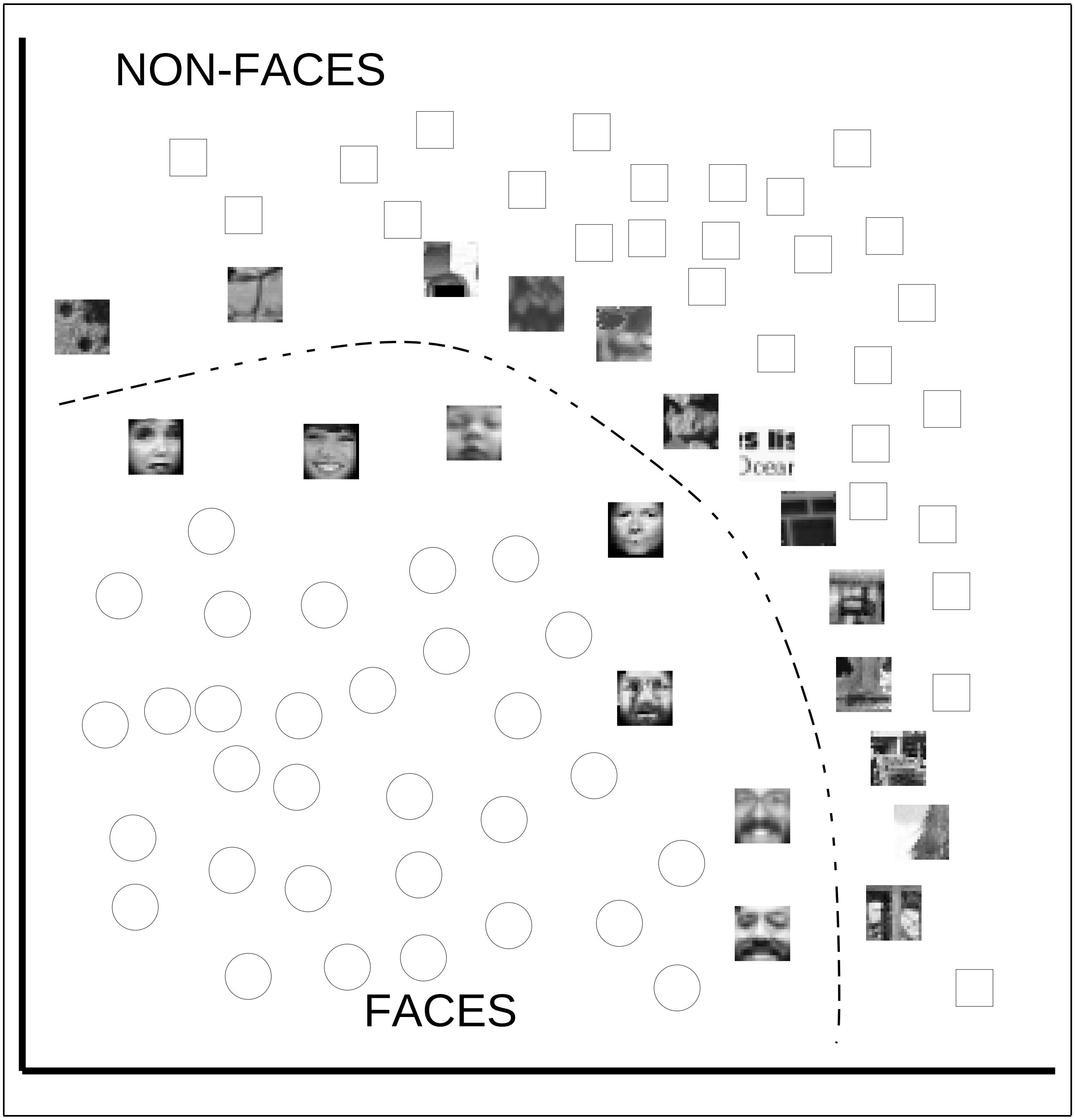


Figure 9: Geometrical Interpretation of how the SVM separates the face and non-face classes. The patterns are real support vectors obtained after training the system. Notice the small number of total support vectors and the fact that a higher proportion of them correspond to non-faces.

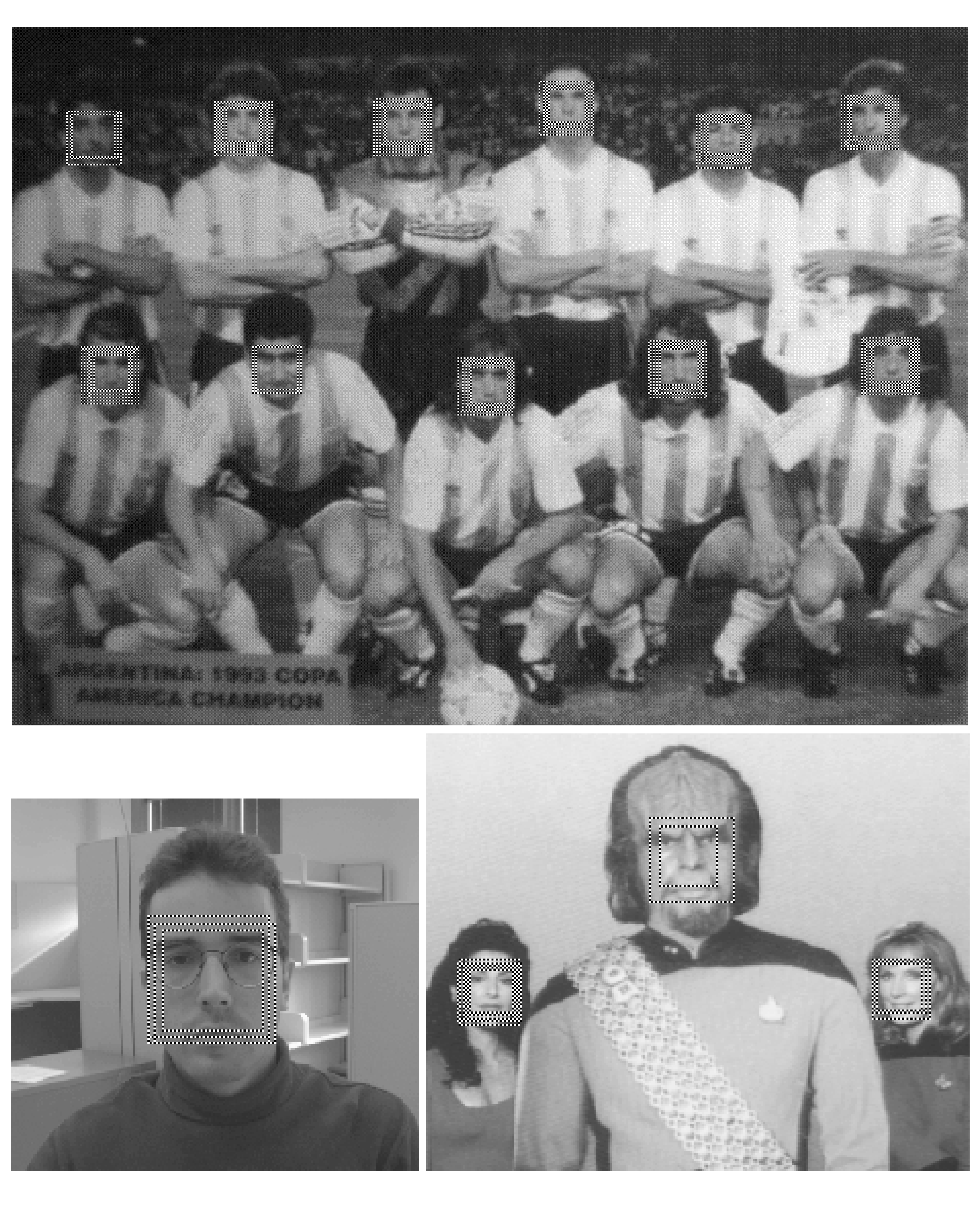


Figure 10: Faces

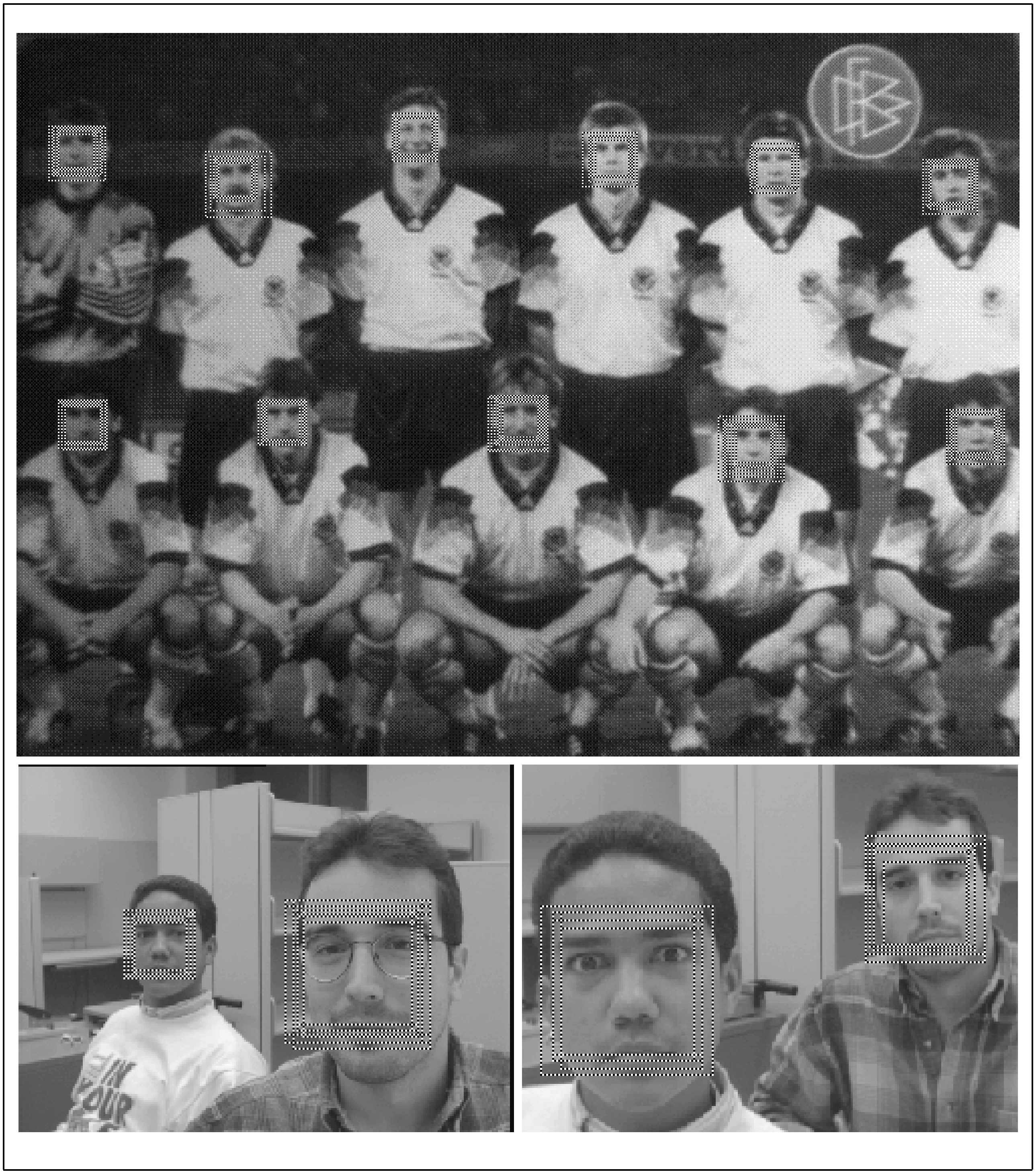


Figure 11: Faces

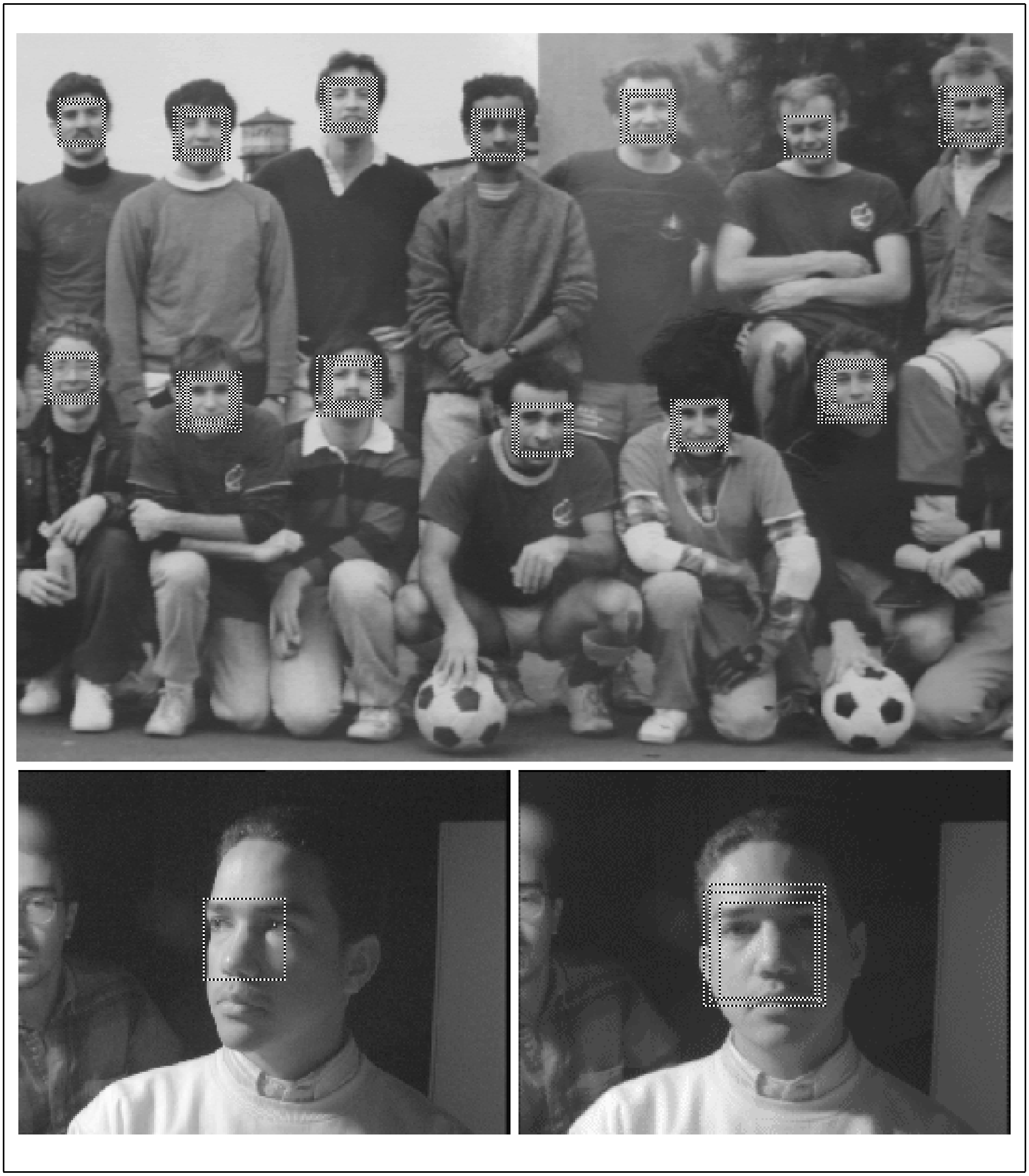


Figure 12: Faces

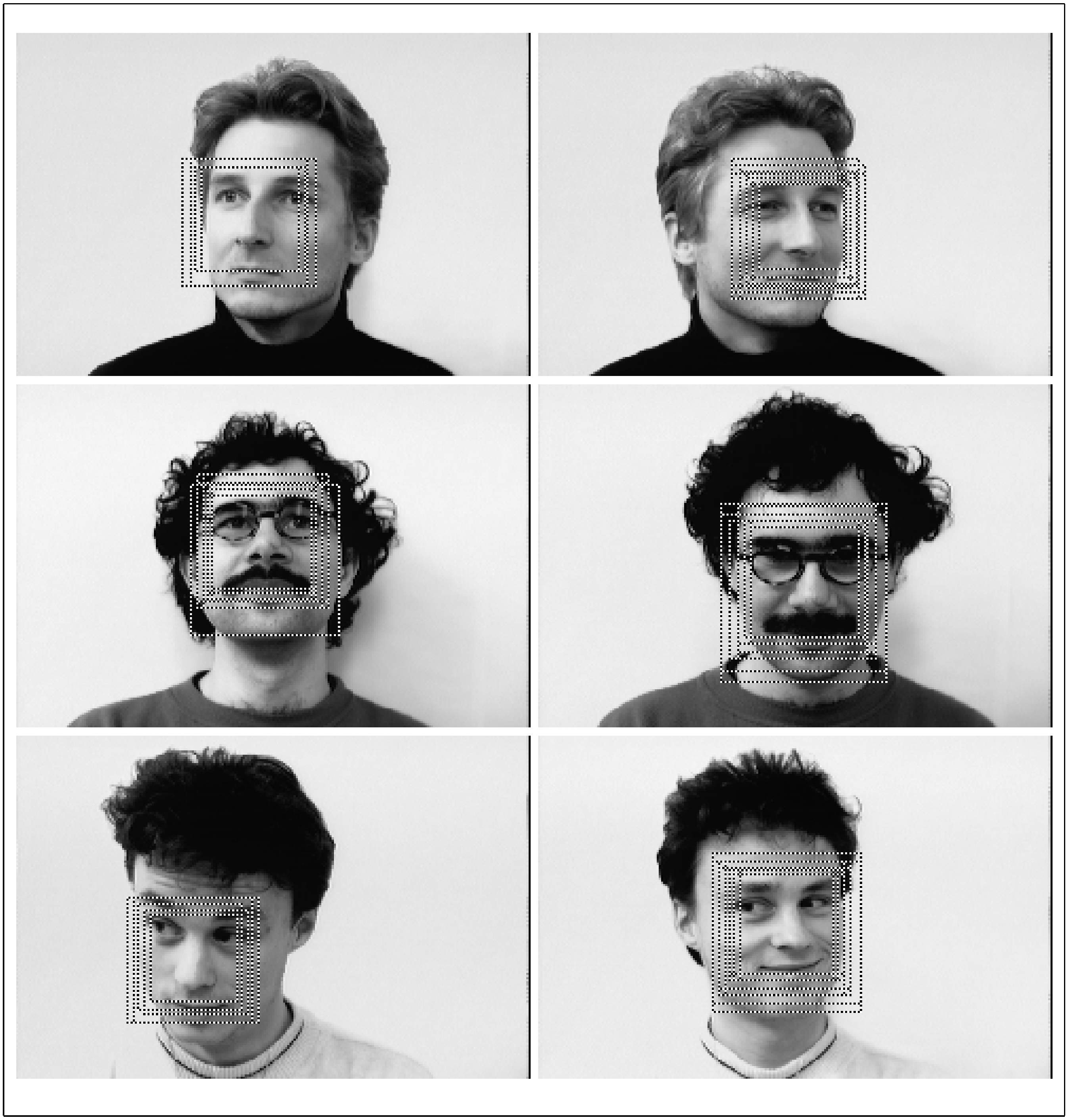


Figure 13: Faces

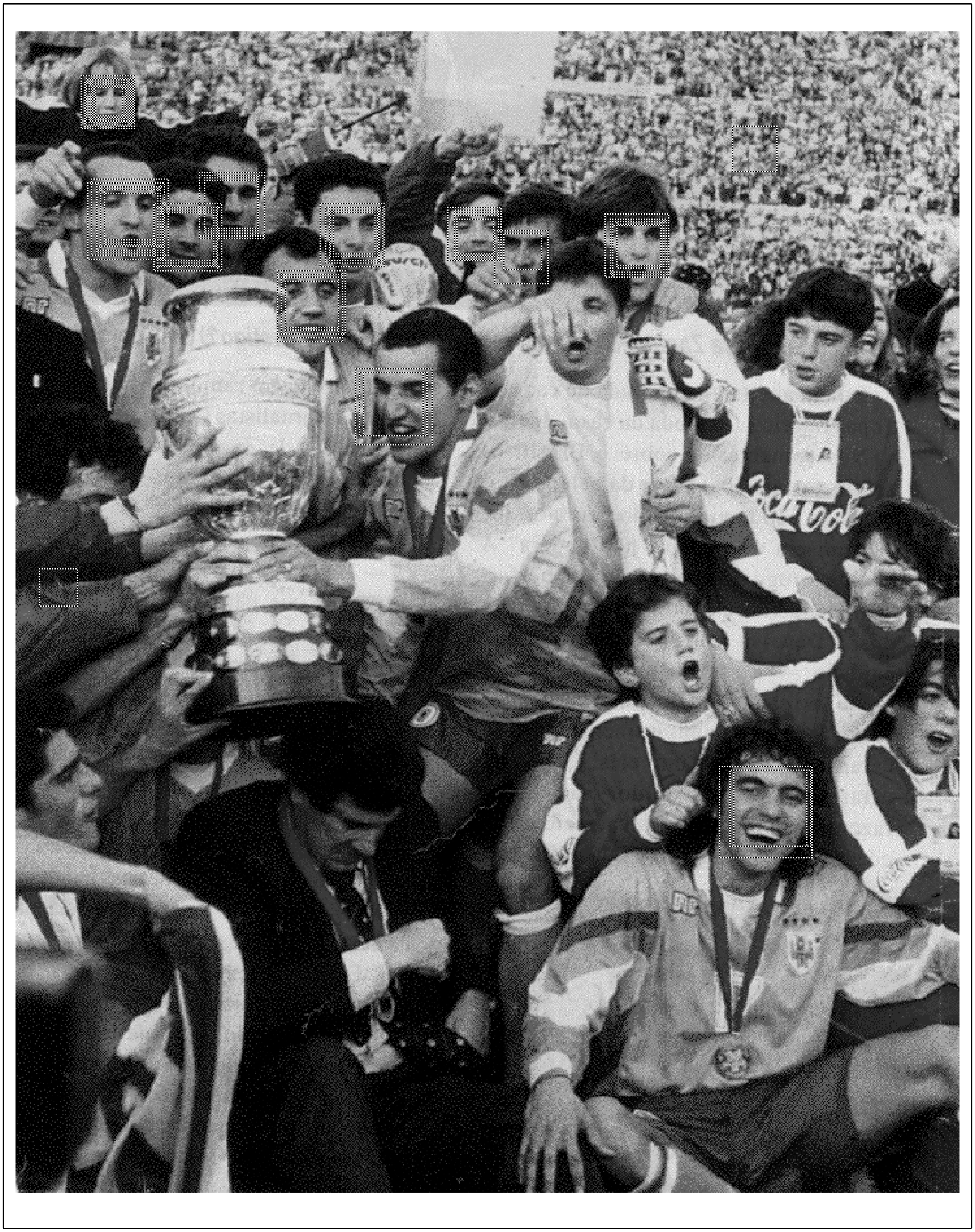


Figure 14: Faces

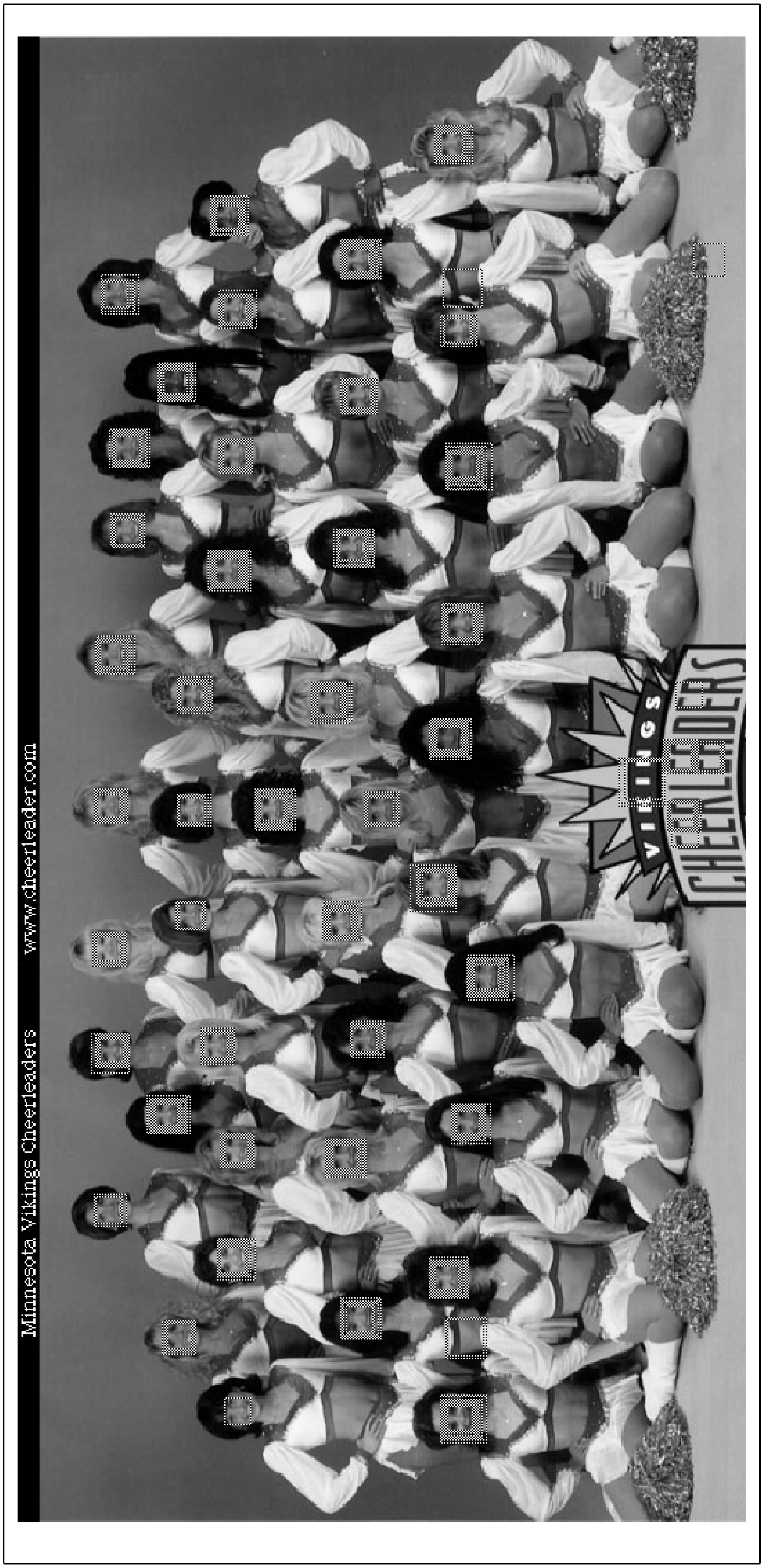


Figure 15: Faces