Learning and Nonlinear Models - Revista da Sociedade Brasileira de Redes Neurais (SBRN), Vol. 4, No. 2, pp. 54-69, 2006

T u t o r i a l

ON FAST SVM ALGORITHMS USED FOR PATTERN RECOGNITION

Felipe A. C. de Bastos[1] and Marcello L. R. de Campos[2]
[1]Brazilian Army Technological Center - CTEx
Av. das Américas 28705, D10, CEP 23020-470, Rio de Janeiro, RJ
[2]Electrical Engineering Program
COPPE/Federal University of Rio de Janeiro
P.O.Box 68504, CEP 21945-970, Rio de Janeiro, RJ, Brazil
fcaetano@ctex.eb.br and campos@lps.ufrj.br

Abstract - This tutorial on fast Support Vector Machines (SVM) presents mathematical formulations and pseudo-code Implementations of three algorithms used for fast SVM training. Traditional SVM training is a quadratic-programming (QP) minimization problem that can be solved, e.g., using the Sequential Minimization Optimization (SMO) algorithm. This algorithm solves analytically a small QP optimization problem in each iteration, drastically reducing the training time needed by conventional QP optimizers. It is important to note that traditional SVM can be of two types: L1SVM and L2SVM, depending on the way that the training error is characterized in the SVM mathematical formulation. The SMO implementation presented in this tutorial applies only for the L1SVM, but it can be adapted to the L2SVM case.

The Proximal SVM (PSVM) algorithm was also introduced as a fast alternative to traditional SVM classifiers that usually require a large amount of computation time for training. Unfortunately the PSVM algorithm may present poor performance due to biased optimal hyperplanes. The Unbiased Proximal SVM (UPSVM) algorithm uses a slightly different approach to circumvent this problem, such that an unbiased optimal hyperplane is always obtained.

The results obtained show that the UPSVM algorithm performs better than the Sequential Minimal Optimization (SMO) algorithm with respect to training time with similar or better probability of correct pattern classification. The UPSVM algorithm also performs better than the PSVM algorithm with respect to probability of correct pattern classification (especially for low values of the regularization parameter C), to training time, and to the number of floating point operations.

Keywords - Support Vector Machines – SVM, fast algorithms, Pattern Recognition

1 Introduction

Support Vector Machines (SVM) were first introduced by Vapnik based on the method of Minimal Structural Risk Minimization [1]. Since its proposal, it has been used in different applications, such as pattern classification [2, 3] and nonlinear regression [4, 5, 6]. In this work, we are particularly interested in explaining the use of SVM for pattern classification. A Support Vector Machine is basically a binary pattern classifier of *M*-dimensional data. A set of different SVMs can be used to solve the more general problem of *N*-class classification. An SVM classifier is obtained after a supervised training procedure where maximization of the distance between two classes (represented by two sets of training vectors) is done along with the minimization of the training error.

SVM training consists basically in finding two support hyperplanes, one for each class, so that the distance between classes can be measured as the distance between the support hyperplanes. The SVM classifier is represented by the optimal hyperplane located in the middle of both support hyperplanes and is defined by a small amount of training patterns called support vectors (SVs). In the case of linearly separable classes, as illustrated in Figure 1, the SVs are located at the respective support hyperplanes and correspond to the closest data of each class to the optimal hyperplane, giving the idea that the support hyperplanes really "support" each class.

In other words, an SVM classifier is a kind of linear discriminator [7], i.e., it uses a hyperplane to divide an M-dimensional space (the input or the feature space) in two regions, each one associated with a different class. What makes it different from other linear discriminators is the fact that maximization of the distance between two classes is done in the training procedure as a means to obtain good generalization capability.

As will be seen in the next section, in order to train an SVM classifier it is necessary to solve a quadratic-programming (QP) optimization problem which is usually a slow and difficult task. Besides, the required memory to solve QP problems for a large data set is sometimes prohibitive. The most common technique used to circumvent these problems is based on the solution of many smaller QP problems in each iteration. Two classical methods often found in literature are interesting cases of this training technique: The SVMlight algorithm [8] and the Sequential Minimal Optimization (SMO) algorithm [9]. The SVMlight algorithm uses a decomposition method proposed by Osuna et al. [2] to obtain a smaller QP problem in each iteration but uses conventional QP optimizers to solve it. On the other hand, the SMO algorithm solves analytically one QP problem involving only two patterns in each iteration. The SMO algorithm is detailed in this tutorial because it does not need an external QP optimizer and, consequently, it can handle large training sets in a very fast and simple way. Besides, although it is claimed in [8] that the SVMlight algorithm was two times faster than the SMO algorithm, the latter was at least one order of magnitude faster than the former in [10].

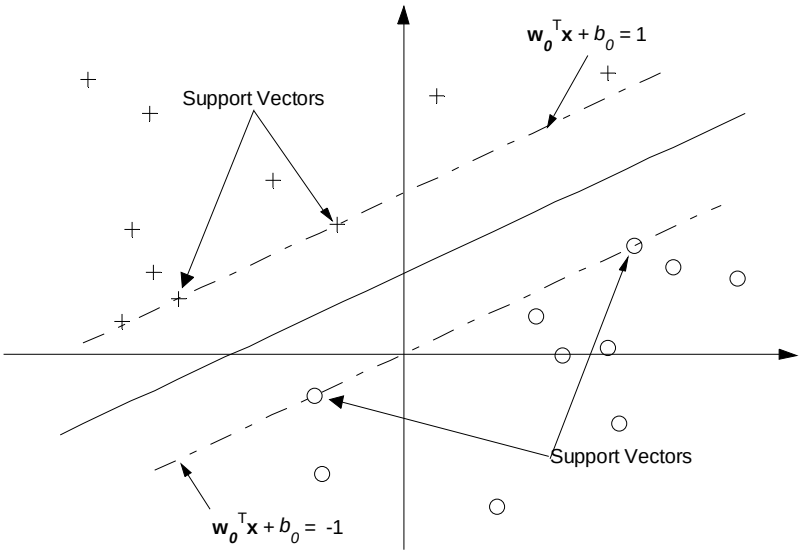

Figure 1: Example for separable classes.

Recently, Vishwanathan et al. proposed the SimpleSVM algorithm [11]. This algorithm also solves many smaller QP problems in each iteration, but its main difference to the two algorithms briefly discussed above is that the set of support-vector candidates (and the small QP problem) grows at most one training example in each iteration. This algorithm is based on the idea that the number of support vectors is small relative to the size of the data set and that the original QP problem can be solved with a smaller one that takes into account only the SVs. But it is also important to note that the number of SVs can vary with respect to the regularization parameter C (present in the SVM mathematical formulation) whose optimal value depends on the application.

It is important to note that traditional SVM can be of two types: L1SVM and L2SVM, depending on the way that the training error is characterized in the SVM mathematical formulation (see next section). The tree methods cited were derived for the L1SVM classifier but can be adapted for the L2SVM formulation.

This work also describes in detail two other algorithms used for supervised training of a modified SVM classifier. The first algorithm is called Proximal SVM (PSVM) and was introduced by Mangasarian and Fung [10]. The second algorithm is called Unbiased PSVM (UPSVM) and was introduced in [12] and always generates an unbiased optimal hyperplane, which is not the case when using the PSVM algorithm. The UPSVM algorithm is slightly faster than the PSVM algorithm and much faster than the SMO algorithm, but maintains equal or better performance [12, 13]. PSVM and UPSVM algorithms have computational complexity of $O(N)$ in the case of linear classifiers [13], and the SMO algorithm has computational complexity of $O(N^2)$ in the best case [9].

In order to present SMO, PSVM, and UPSVM algorithms and their implementations, this work is divided as follows: Section 2 summarizes the theory about SVM used for pattern classification, including linear and nonlinear classifiers for Large-Margin SVM (L1SVM) and Proximal SVM (PSVM and UPSVM) algorithms. In section 3, SMO, PSVM, and UPSVM

algorithms are described and pseudo-code implementations are given. Finally, section 4 presents the conclusions.

2 SVM and PSVM Formulations

2.1 Support Vector Machines

The classification surface is a hyperplane on the input space (linear SVMs) or on the feature space (nonlinear SVMs) obtained after applying a nonlinear transformation to the input space. For linear classifiers, the separating hyperplane in the input space is given by

$$\mathbf{x}^T \mathbf{w}_o + b_o = 0 \tag{1}$$

where, in the SVM nomenclature, $\mathbf{w} \in R^M$ and $b \in R$ denote weight vector and bias, respectively, and $\mathbf{x} \in R^M$ is a vector in the input space. An input pattern, $\mathbf{x}_i$, is said to belong to class C_1 if $\mathbf{x}_i^T \mathbf{w}_o + b_o \geq 0$, and to belong to class C_2 otherwise. For nonlinear classifiers, the separating hyperplane in the feature space is given by

$$g(\mathbf{x})^T \mathbf{w}_o + b_o = 0 \tag{2}$$

where $g(\mathbf{x})$ is a mapping function that maps the M-dimensional input space into the L-dimensional feature space ($L \geq M$).

In the case of linearly separable patterns shown in figure 1 (hard margin), the support vectors are defined as the closest data to the optimal hyperplane, i.e., without loss of generality, for a linear classifier, a support vector $\mathbf{x}_S$ must satisfy

$$\mathbf{x}_S^T \mathbf{w}_o + b_o = \begin{cases} 1 & \text{if } \mathbf{x}_S \in C_1 \\ -1 & \text{if } \mathbf{x}_S \in C_2 \end{cases} \tag{3}$$

The distance between the support hyperplanes defined by Eq. 3 is called margin of separation. Let $\mathbf{x}_1$ be a support vector belonging to class C_1 and $\mathbf{x}$ be its projection onto the optimal hyperplane. The distance between $\mathbf{x}_1$ and the optimal hyperplane is given by

$$r_1 = \frac{|(\mathbf{x}_1 - \mathbf{x})^T \mathbf{w}_o|}{\|\mathbf{w}_o\|} \tag{4}$$

where both $(\mathbf{x}_1 - \mathbf{x})$ and $\mathbf{w}_o$ are vectors perpendicular to the optimal hyperplane that can be pointing in the same, or in opposite directions. Applying Eqs.1 and 3 to Eq.4, we obtain

$$r_1 = \frac{1}{\|\mathbf{w}_o\|} \tag{5}$$

Proceeding in the same way for $\mathbf{x}_2$, a support vector belonging to class C_2, the distance between it and the optimal hyperplane is

$$r_2 = \frac{1}{\|\mathbf{w}_o\|} \tag{6}$$

So the margin of separation, calculated as $\rho = r_1 + r_2$, is inversely proportional to the magnitude of $\mathbf{w}$ and is given by

$$\rho = \frac{2}{\|\mathbf{w}_o\|} \tag{7}$$

When input patterns are not linearly separable (soft margin), the support vectors are still given by Eq. 3, but some input patterns may lay inside the margin of separation, or on the wrong side of the optimal hyperplane.

Figure 2 shows the use of a hyperplane for the classification of input vectors from two non-linearly separable classes. Let us define $\mathbf{x}_i$ the observed pattern and ξ_i the training error, which is equal to zero if and only if $\mathbf{x}_i$ is bounded by the respective support hyperplane, otherwise ξ_i is equal to the distance from $\mathbf{x}_i$ to the respective support hyperplane. As it can be seen, some vectors of both classes can be located on the wrong side of its support hyperplane, which means training error different from zero ($\xi_i > 0$), but only those located on the wrong side of the optimal hyperplane are wrongly classified ($\xi_i > 1$).

Classification error may occur when classes are non-linearly separable. In this case, the optimal hyperplane parameters are obtained by the minimization of the training error (indirectly related to the classification error) along with the maximization of the margin of separation. The latter can be understood as a way to obtain classification error minimization for unknown data.

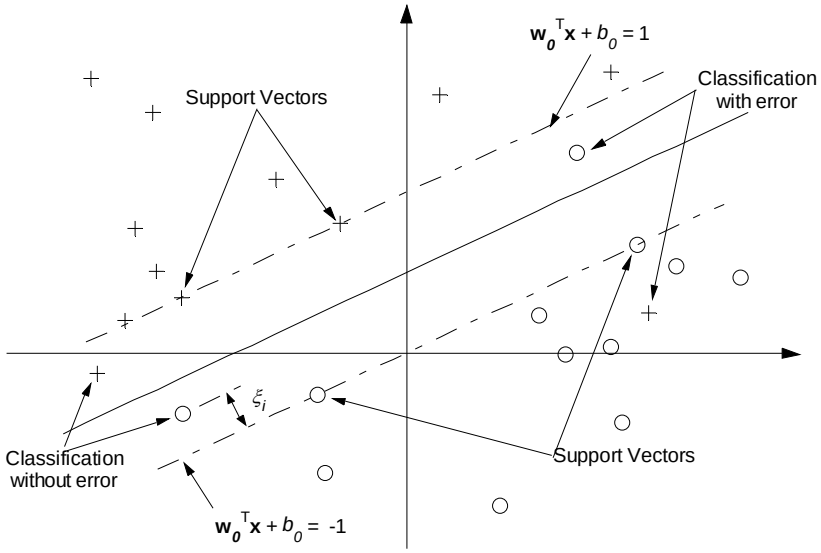

Figure 2: Example for non-separable classes.

To take into account both the maximization of the margin of separation and the minimization of the training error, the following linearly-constrained convex minimization problem was proposed [1]

$$\min_{\mathbf{w},\xi}\left[J(\mathbf{w},\boldsymbol{\xi}) = \left(\frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{p}\sum_{i=1}^{N}\xi_i^p\right)\right] \quad \text{subject to} \quad \mathbf{D}(\mathbf{X}^T\mathbf{w} + b\mathbf{e}) \geq \mathbf{e} - \boldsymbol{\xi} \tag{8}$$

where each column of $\mathbf{X} \in R^{MxN}$ is an observation ($\mathbf{x}_i$), $\mathbf{D}$ is a diagonal matrix with class-label elements D_{ii} equal to 1 if $\mathbf{x}_i$ belongs to class C_1, or -1 otherwise, $\xi \in R^N$ is a vector of training errors and vector $\mathbf{e}$ has all its elements equal to one.

The first term of the objective function in Eq. 8 is due to the maximization of the margin of separation. Its second term corresponds to the training error weighted by the regularization parameter C, which measures how much emphasis is given to the minimization of the training error. The parameter p defines the classifier type. When p=1 the obtained classifier is called L_1 large-margin SVM (L1SVM), and when p=2 it is called L_2 large-margin SVM (L2SVM). The choice of value for C is a very important matter because if too much emphasis is given to the minimization of the training error, then the trained classifier can achieve poor generalization performance, or if too much emphasis is given to the maximization of the margin of separation, then the classification error for the training patterns may be too big and the classifier would be useless.

The solution for the L1SVM classifier ($p = 1$) can be obtained using the method of Lagrange multipliers. The Lagrangian function is given by

$$J(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\mu}) = \left(\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\boldsymbol{\xi}^T\mathbf{e}\right) - \boldsymbol{\alpha}^T\left(\mathbf{D}(\mathbf{X}^T\mathbf{w} + b\mathbf{e}) - \mathbf{e} + \boldsymbol{\xi}\right) - \boldsymbol{\mu}^T\boldsymbol{\xi} \tag{9}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ are the Lagrange multiplier vectors associated to the constraint in Eq. 8, and to the constraint $\xi_i \geq 0$, respectively.

In order to obtain the optimal solution that minimize the Lagrangian function, it is necessary to calculate $\nabla J = \mathbf{0}$, obtaining

$$\mathbf{w}_o = \mathbf{XD}\boldsymbol{\alpha} \tag{10}$$

$$\boldsymbol{\alpha}^T\mathbf{De} = 0 \tag{11}$$

and

$$\boldsymbol{\alpha} + \boldsymbol{\mu} = C\mathbf{e} \tag{12}$$

Eqs. 10, 11, and 12 result from making the the partial derivatives of the Lagrangian function with respect to **w**, *b*, and $\boldsymbol{\xi}$ all equal to zero. Substituting Eq. 10 in Eq. 9 and using Eqs. 11 and 12, we obtain

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{DX}^T\mathbf{XD}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{e} \tag{13}$$

Therefore the solution to Eq. 8 can also be obtained by the solution of the dual-problem described below [1]

$$\min_{\boldsymbol{\alpha}}\left[J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{DX}^T\mathbf{XD}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{e} \right] \quad \text{subject to:} \begin{cases} \boldsymbol{\alpha}^T\mathbf{De} = 0 \\ 0 \leq \boldsymbol{\alpha}_i \leq C, \quad i = 1,2,\ldots,N \end{cases} \tag{14}$$

where the constraints $\alpha_i \geq 0$, $i = 1,2,\ldots,N$, are imposed by the method of Lagrange multipliers, and all α_i must be less than or equal to *C* due to Eq. 12 and because μ_i is also a Lagrange multiplier and must also be nonnegative.

After solving Eq. 14, an optimal Lagrange multiplier vector $\boldsymbol{\alpha}_o$ is obtained. Each element of $\boldsymbol{\alpha}_o$ must obey the Karush-Kuhn-Tucker (KKT) conditions for optimality given by:

$$\begin{aligned} (\mathbf{x}^T\mathbf{w}_o + b_o)d_i > 1 &\leftrightarrow \alpha_i = 0 \\ (\mathbf{x}^T\mathbf{w}_o + b_o)d_i = 1 &\leftrightarrow 0 < \alpha_i < C \\ (\mathbf{x}^T\mathbf{w}_o + b_o)d_i < 1 &\leftrightarrow \alpha_i = C \end{aligned} \tag{15}$$

where d_i is the classification label of the training pattern $\mathbf{x}_i$, i.e., it is the *i*-th element of the vector **De**. The optimal weight vector $\mathbf{w}_o$ is given by using $\boldsymbol{\alpha}_o$ in Eq. 10. The optimal bias is given by Eq. 3, rewritten below in a different manner

$$b_o = d_S - \mathbf{w}_o^T\mathbf{x}_S \tag{16}$$

where d_S is the classification label (± 1) of the support vector $\mathbf{x}_S$, associated to a Lagrange multiplier that is not equal to zero or *C*.

To classify an unknown pattern **y** after the L1SVM classifier was trained, it is necessary to calculate

$$\text{dist}(\mathbf{y}) = \mathbf{y}^T\mathbf{w}_o + b_o \tag{17}$$

which is proportional to the distance of $\mathbf{y}$ from the optimal hyperplane. If dist(**y**) is greater than or equal to zero, then **y** belongs to class C_1, otherwise it belongs to class C_2.

A nonlinear L1SVM classifier can be easily obtained proceeding in the same way used to obtain a linear L1SVM

classifier, but replacing $\mathbf{x}$ for $g(\mathbf{x})$ and $\mathbf{X}^T\mathbf{X}$ for the kernel matrix $\mathbf{K} \in R^{NxN}$ whose element $K_{ij} = g(\mathbf{x}_i)^T g(\mathbf{x}_j)$ and N is the number of training observations. A radial-basis function (RBF) kernel, defined below, is usually used in nonlinear classifiers [14]

$$K_{ij} = \exp\left(\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right) \tag{18}$$

where σ is a parameter defined by the user.

For a nonlinear L1SVM classifier, Eq. 17 becomes

$$\text{dist}(\mathbf{y}) = \mathbf{k}_\mathbf{y}^T \mathbf{D}\boldsymbol{\alpha}_o + b_o \tag{19}$$

where $\mathbf{k}_\mathbf{y} = [g(\mathbf{y})^T g(\mathbf{x}_1)\ g(\mathbf{y})^T g(\mathbf{x}_2) \ldots g(\mathbf{y})^T g(\mathbf{x}_r)]^T$ and $\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_r$ are support vectors, also obtained after training. Again, if $\text{dist}(\mathbf{y}) \geq 0$ then $\mathbf{y} \in C_1$, otherwise $\mathbf{y} \in C_2$.

With respect to the L2SVM classifier, its dual-problem described in Eq. 20 [15] is obtained by the same procedure that was carried out to derive the L1SVM classifier, i.e., calculating $\nabla(J)$, with $p = 2$ in Eq. 8, and making it equal to zero.

$$\min_{\boldsymbol{\alpha}}\left[J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{DKD}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{e} + \frac{\boldsymbol{\alpha}^T\boldsymbol{\alpha}}{2C} \right] \quad \text{subject to}: \begin{cases} \boldsymbol{\alpha}^T\mathbf{De} = 0 \\ \boldsymbol{\alpha}_i \geq 0, \quad i = 1,2,\ldots,N \end{cases} \tag{20}$$

It is important to note that Eqs. 10 and 11 remain valid for the L2SVM classifier, but making the partial derivative of J with respect to $\boldsymbol{\xi}$ equal to zero gives

$$\xi_o = \frac{\boldsymbol{\alpha}_o}{C} \tag{21}$$

Applying Eq. 21 in Eq. 8 results in the third term in Eq. 20. The Eq. 21 also gives the explanation for the difference in the constraint relative to the upper bound of $\boldsymbol{\alpha}$ for the L2SVM classifier when compared to the L1SVM classifier. This L2SVM formulation is valid for the linear case, where the kernel matrix $\mathbf{K}$ is equal to $\mathbf{X}^T\mathbf{X}$, as well as for the nonlinear case. It is also important to note that Eq. 17 (or Eq. 19 in the nonlinear case) remains valid for the L2SVM classifier, but Eq. 16 must be changed to

$$b_o = d_S(1-\xi_S) - \mathbf{x}_S^T\mathbf{w}_o \tag{22}$$

where ξ_S is the training error associated with the support vector $\mathbf{x}_S$.

2.2 Proximal Support Vector Machines

The PSVM algorithms was first introduced by Mangasarian and Fung [10] as a fast-training algorithm for a modified SVM problem. The PSVM algorithm associated with the DAGSVM algorithm [16] was used by Li and others in [17] for the generalization of the PSVM classifier for more than 2 classes. The formulation for the linear case is described below

$$\min_{\mathbf{w},b,\boldsymbol{\xi}}\left[J(\mathbf{w},b,\boldsymbol{\xi}) = \frac{1}{2}C\boldsymbol{\xi}^T\boldsymbol{\xi} + \frac{1}{2}\left(\mathbf{w}^T\mathbf{w} + b^2\right) \right] \quad \text{subject to} \quad \boldsymbol{\xi} = \mathbf{e} - \mathbf{D}\left(\mathbf{X}^T\mathbf{w} + \mathbf{e}b\right) \tag{23}$$

The PSVM algorithm is a natural evolution of a similar approach that had been used before in the formulation of the Active SVM (ASVM) algorithm proposed by Mangasarian et al. [18], where the term b^2 was added to the objective function, in the same way that was done in Eq. 23, as an artificial tool included to obtain an analytic solution for $\mathbf{w}$, b and $\boldsymbol{\xi}$ based on a small system of linear equations. It is important to note that the inclusion of b^2 in Eq. 23 does not improve classification performance, but rather, it will decrease the probability of correct classification on training and test sets for small values of the regularization parameter C [13]. This happens because much effort is made to minimize the magnitudes of $\mathbf{w}$ and b

simultaneously, which implies that parameter b be unnecessarily small and the optimal hyperplane be biased, i.e., closer to the origin than necessary.

Eq. 23 also shows that the traditional SVM inequality constraint is replaced by an equality constraint. This fact changes the nature of the support hyperplanes ($\mathbf{w}_o^T\mathbf{x} + b_o = \pm 1$). These hyperplanes are no longer bounding hyperplanes. Instead, they correspond to proximal hyperplanes around which the points of each class are clustered. Figure 3 illustrates this approach. In this case, the training error is always greater than zero for vectors not located at the proximal hyperplane. As it occurs with traditional L1SVM or L2SVM, only those vectors located on the wrong side of the optimal hyperplane are wrongly classified ($\xi_i > 1$). In fact, the geometrically motivated proximal formulation has been considered in the more general context of regularization networks [10].

Using Lagrange for solving the PSVM minimization problem and considering $\boldsymbol{\alpha}$ as a vector of Lagrange multipliers, the solution for the PSVM problem in Eq. 23 can be obtained as follows:

$$\boldsymbol{\alpha}_o = \left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{e} \tag{24}$$

$$\mathbf{w}_o = \mathbf{X}\mathbf{D}\boldsymbol{\alpha}_o \tag{25}$$

$$b_o = \mathbf{e}^T\mathbf{D}\boldsymbol{\alpha}_o \tag{26}$$

$$\xi_o = \frac{\boldsymbol{\alpha}_o}{C} \tag{27}$$

where $\mathbf{H} = \mathbf{D}[\mathbf{X}^T \quad -\mathbf{e}\ ]$. The Eq. 24 was obtained after substituting $\mathbf{w}_o$, b_o and $\boldsymbol{\xi}$, given by Eqs. 25 to 27, respectively, in the equality constraint given in Eq. 23.

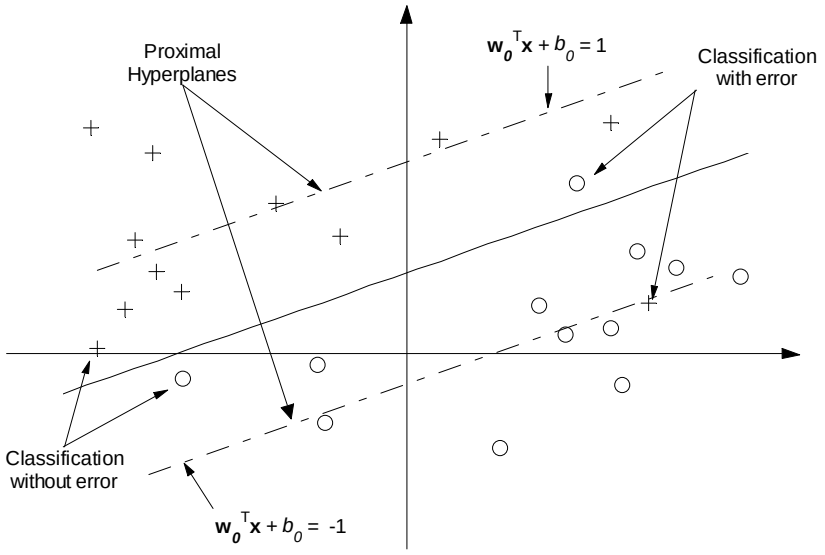

Figure 3: Example for non-linearly separable classes with proximal hyperplanes.

We can use the matrix inversion Lemma [19] to obtain:

$$\boldsymbol{\alpha}_o = C\left[\mathbf{I} - \mathbf{H}\left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\right]\mathbf{e} \tag{28}$$

The solution given by Eq. 28 is less complex than the one given by Eq. 24 because it involves the inversion of the $(M+1)\times(M+1)$ matrix $\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}$, where M is the number of observation parameters, which is much smaller than N, the

number of observations used in training.

For the nonlinear case, the PSVM classifier is given by the following equations (see [10] for details):

$$\boldsymbol{\alpha}_o = \mathbf{DKD}\left(\frac{\mathbf{I}}{C} + \mathbf{GG}^T\right)^{-1}\mathbf{e} \tag{29}$$

$$b_o = \mathbf{e}^T\mathbf{D}\boldsymbol{\alpha}_o \tag{30}$$

$$\boldsymbol{\xi} = \frac{\boldsymbol{\alpha}_o}{C} \tag{31}$$

where $\mathbf{G}=\mathbf{D}[\mathbf{K} \quad -\mathbf{e}]$, and $\mathbf{K}$ is the Kernel Matrix. Given $\boldsymbol{\alpha}_o$ and b_o, the Eq. 19 can be used to classify a test pattern $\mathbf{y}$. It is very important to note that the generalization to the nonlinear PSVM classifier was done by replacing $\mathbf{X}$ for $\mathbf{K}$. This non-conventional procedure may degrade its performance in the nonlinear case.

The PSVM algorithm leads to an analytic solution for a modified SVM problem. Its main advantage compared to traditional SVM problems is its low computational complexity for the linear case. For the nonlinear case, reduced-kernel techniques [10] can be used to reduce the N x N dimensionality of the kernel matrix $\mathbf{K}$. In the next section, a new algorithm called UPSVM, is presented. It eliminates the dependence of the objective function to the bias parameter b and therefore eliminates a disadvantage of the PSVM algorithm: The biased optimal hyperplane obtained for small values of the regularization parameter C. Besides, the UPSVM-algorithm generalization to the nonlinear classifier is similar to that used in the L1SVM-classifier generalization.

2.3 Unbiased Proximal SVM

The minimization of b^2 does not lead to improved classifier generalization or to less training errors. Instead, it can decrease the probability of correct classification of test data and increase the number of training errors. Therefore, a new algorithm was proposed in [12] that does not take into account parameter b in the objective function to be minimized. Using proximal hyperplanes, the new problem for the linear case can be mathematically formulated as

$$\min_{\mathbf{w},\boldsymbol{\xi}}\left[J(\mathbf{w},\boldsymbol{\xi}) = \frac{1}{2}C\boldsymbol{\xi}^T\boldsymbol{\xi} + \frac{1}{2}\mathbf{w}^T\mathbf{w}\right] \quad \text{subject to} \quad \boldsymbol{\xi} = \mathbf{e} - \mathbf{D}\left(\mathbf{X}^T\mathbf{w} + \mathbf{e}b\right) \tag{32}$$

Substituting the equality constraint into the objective function in Eq. 32, we obtain

$$\min_{\mathbf{w},b}\left[J(\mathbf{w},b) = \frac{1}{2}C\left(\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} + 2b\mathbf{w}^T\mathbf{X}\mathbf{e} + b^2N + N - 2\mathbf{w}^T\mathbf{X}\mathbf{D}\mathbf{e} - 2b\mathbf{e}^T\mathbf{D}\mathbf{e}\right) + \frac{1}{2}\mathbf{w}^T\mathbf{w}\right] \tag{33}$$

Eq. 33 shows the dependence of the objective function to the hyperplane parameters $\mathbf{w}$ and b. In order to find a solution for $\mathbf{w}$ and b, the gradient of $J(\mathbf{w},b)$ is calculated, which gives

$$\frac{\partial J}{\partial \mathbf{w}} = C\left(\mathbf{X}\mathbf{X}^T\mathbf{w} + b\mathbf{X}\mathbf{e} - \mathbf{X}\mathbf{D}\mathbf{e}\right) + \mathbf{w} \tag{34}$$

and

$$\frac{\partial J}{\partial b} = C\left(\mathbf{w}^T\mathbf{X}\mathbf{e} + bN - \mathbf{e}^T\mathbf{D}\mathbf{e}\right) \tag{35}$$

Making $\nabla J = \mathbf{0}$ gives

$$\mathbf{w}_o = \left[\frac{\mathbf{I}}{C} + \mathbf{X}\left(\mathbf{I} - \frac{1}{N}\mathbf{e}\mathbf{e}^T\right)\mathbf{X}^T\right]^{-1}\mathbf{X}\mathbf{D}\mathbf{e} - \mathbf{X}\left(\frac{\mathbf{e}\mathbf{e}^T}{N}\right)\mathbf{D}\mathbf{e} \tag{36}$$

and

$$b_o = \frac{\mathbf{e}^T\mathbf{D}\mathbf{e} - \mathbf{e}^T\mathbf{X}^T\mathbf{w}_o}{N} \tag{37}$$

Supposing the same number of training observations for classes C_1 and C_2, then $\mathbf{e}^T\mathbf{D}\mathbf{e} = 0$, and Eqs 36 and 37 can be simplified to

$$\mathbf{w}_o = \left[\frac{\mathbf{I}}{C} + \mathbf{X}\left(\mathbf{I} - \frac{1}{N}\mathbf{e}\mathbf{e}^T\right)\mathbf{X}^T\right]^{-1}\mathbf{X}\mathbf{D}\mathbf{e} \tag{38}$$

and

$$b_o = -\frac{\mathbf{e}^T\mathbf{X}^T\mathbf{w}_o}{N} \tag{39}$$

An analytic solution for a Proximal SVM has been found, but without taking into account the minimization of b. It means that the optimal hyperplane will be unbiased even for small values of C. The solution of Eqs. 38 and 39 also involves the inversion of an M x M matrix instead of the inversion of an N x N matrix, which guarantees low computational complexity ($O(N)$ [12]). This complexity is much lower than that of the Sequential Minimal Optimization (SMO) algorithm [9] used to train L1SVM. Its important to observe that the computational complexity of the Sequential Minimal Optimization (SMO) algorithm used to train L1SVM is not deterministic, but has been estimated [9] as kN^{γ}, where N is the number of observations used in training, k is a positive real number, and $\gamma = 2$ in the best case.

To obtain a nonlinear UPSVM algorithm we will proceed in the same way that traditional SVM problems were generalized [1], i.e., substituting $\mathbf{X}^T\mathbf{X}$ by the kernel matrix $\mathbf{K}$, whose element $K_{i,j} = g(\mathbf{x}_i)^T g(\mathbf{x}_j)$. Applying the inversion lemma [19] to Eq. 38 yields

$$\mathbf{w}_o = C\left[\mathbf{I} - \mathbf{X}\left(\frac{\mathbf{I}}{C} + \mathbf{A}\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{A}\mathbf{X}^T\right]\mathbf{X}\mathbf{D}\mathbf{e} \tag{40}$$

where

$$\mathbf{A} = \left(\mathbf{I} - \frac{1}{N}\mathbf{e}\mathbf{e}^T\right) \tag{41}$$

If

$$\boldsymbol{\alpha}_o = C\mathbf{D}\left[\mathbf{I} - \left(\frac{\mathbf{I}}{C} + \mathbf{A}\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{A}\mathbf{X}^T\mathbf{X}\right]\mathbf{X}\mathbf{D}\mathbf{e} \tag{42}$$

we can rewrite Eqs. 39 and 40, respectively, as

$$b_o = -\frac{\mathbf{e}^T\mathbf{X}^T\mathbf{w}_o}{N} = -\frac{\mathbf{e}^T\mathbf{X}^T\mathbf{X}\mathbf{D}\boldsymbol{\alpha}_o}{N} \tag{43}$$

and

$$\mathbf{w}_o = \mathbf{X}\mathbf{D}\boldsymbol{\alpha}_o \tag{44}$$

Note that Eq. 44 has the same form as Eq. 10. Therefore $\boldsymbol{\alpha}_o$ can be understood as a Lagrange-multiplier vector, although it was not calculated explicitly. Substituting $\mathbf{X}^T\mathbf{X}$ by $\mathbf{K}$ we obtain the nonlinear UPSVM algorithm as

$$\alpha_o = C\mathbf{D}\left[\mathbf{I} - \left(\frac{\mathbf{I}}{C} + \mathbf{AK}\right)^{-1}\mathbf{AK}\right]\mathbf{De} \tag{45}$$

and

$$b_o = -\frac{\mathbf{e}^T\mathbf{KD}\boldsymbol{\alpha}_o}{N} \tag{46}$$

After obtaining $\boldsymbol{\alpha}_o$ and b_o from the training procedure described by Eqs. 45 and 46, the classification task is accomplished by calculating the distance from the test vector $\mathbf{y}$ to the optimal hyperplane. This can be done without knowing the vector $\mathbf{w}_o$ by using Eq. 11, where b_o is given by Eq. 46. If dist($\mathbf{y}$) is positive than $\mathbf{y} \in C_1$, otherwise $\mathbf{y} \in C_2$.

3 Algorithms and Implementations

3.1 SMO Algorithm and Implementation

The L1SVM dual problem described by Eq. 14 can be solved using the SMO algorithm. The SMO algorithm can be divided in three sub-algorithms. The main routine establishes the loop and rules for the choice of the first pattern ($\mathbf{x}_i$) to be used in each loop iteration. The second routine implements heuristic rules to choose the second pattern, and the third routine implements the small QP optimization problem and updates an error cache and the bias parameter b when the Lagrange multipliers associated with the training patterns being used are changed.

The main routine of SMO pseudo-code consists of an outer loop that checks if there is any training pattern that violates the Karush-Kuhn-Tucker (KKT) conditions, given a chosen tolerance ε. The KKT conditions are shown in Eq. 47, where d_i is the i-th element of the vector $\mathbf{De}$ and α_i is the Lagrange multiplier associated with the training pattern $\mathbf{x}_i$.

$$\begin{aligned}
&(\mathbf{x}^T\mathbf{w}_o + b_o)d_i > 1 \;\leftrightarrow\; \alpha_i = 0 \\
&(\mathbf{x}^T\mathbf{w}_o + b_o)d_i = 1 \;\leftrightarrow\; 0 < \alpha_i < C \\
&(\mathbf{x}^T\mathbf{w}_o + b_o)d_i < 1 \;\leftrightarrow\; \alpha_i = C
\end{aligned} \tag{47}$$

Every time a pattern that violates the KKT conditions is found, another pattern is chosen using heuristic rules (second routine), and both are used to minimize the objective function (third routine). The search for violating patterns is done once over the entire training set. Thereafter the search is done over a small training subset made of patterns associated with Lagrange multipliers different from zero and C. This subset contains patterns which are candidates for SVs and have more difficulty to satisfy the KKT conditions. The outer loop continues to search exclusively on this subset until it is self conditioned, i.e., all patterns in this subset do not violate the KKT conditions. When this happens, the outer loop searches once again over the entire training set. If no violating patterns are found after the last iteration over the entire training set, the outer loop ends and an optimal Lagrange-multiplier vector $\boldsymbol{\alpha}_o$ is returned along with an optimal bias b_o.

In the second routine, the second pattern $\mathbf{x}_j$ is chosen from the subset of patterns associated with Lagrange multipliers different from zero and different from C, so that $|E_i - E_j|$ is maximized, where $E_i = dist(\mathbf{x}_i) - d_i$, i.e., E_i is the i-th training pattern error. This is done because the maximum step in the objective function minimum direction is proportional to $|E_i - E_j|$ and inversely proportional to η (see Eq. 53), a parameter that depends on kernel evaluation. Because kernel evaluation is time consuming, the rule is simplified. If no joint optimization can be achieved using these two patterns, then the SMO algorithm continues to search randomly a second pattern inside this subset and, after that, over the entire training set.

It is also important to note that it is suggested in [9] to maintain a cached error vector to speed the SMO algorithm, but in the Matlab implementation, it is faster to maintain a complete updated error vector than to update specific error values in a cache vector in each iteration.

In the third routine, constrained optimization using only two Lagrange multipliers is done, if possible. Due to the constraints in Eq. 14, the optimal Lagrange multipliers lie in a diagonal line within a square. It can be proved [9] that new

Lagrange multipliers in each iteration, if they exist, are given by:

$$\alpha_i^{new} = \begin{cases} H, & \text{if} \quad \alpha_i^{aux} \geq H \\ \alpha_i^{aux}, & \text{if} \quad L \leq \alpha_i^{aux} \leq H \\ L, & \text{if} \quad \alpha_i^{aux} \leq L \end{cases} \tag{48}$$

$$\alpha_j^{new} = \alpha_j^{old} + s(\alpha_i^{old} - \alpha_i^{new}) \tag{49}$$

where $s = d_i d_j$, and L and H are the minimum and maximum values for α_i, respectively. The equations for L, H and α_i^{aux} are given below:

$$L = \begin{cases} \max(0, \alpha_i^{old} - \alpha_j^{old}), & \text{if} \quad s = -1 \\ \max(0, \alpha_i^{old} + \alpha_j^{old} - C), & \text{if} \quad s = 1 \end{cases} \tag{50}$$

$$H = \begin{cases} \min(C, C + \alpha_i^{old} - \alpha_j^{old}), & \text{if} \quad s = -1 \\ \min(C, \alpha_i^{old} + \alpha_j^{old}), & \text{if} \quad s = 1 \end{cases} \tag{51}$$

and

$$\alpha_i^{aux} = \alpha_i^{old} - d_i \frac{E_i - E_j}{\eta} \tag{52}$$

where

$$\eta = 2K_{i,j} - K_{i,i} - K_{j,j} \tag{53}$$

The third and innermost routine ends with the calculation of b^{new} and the error cache vector, $\mathbf{E}^{new}$. The updated value for b^{new} is

$$b^{new} = \begin{cases} b_i, & \text{if} \quad 0 < a_i^{new} < C \\ b_j, & \text{if} \quad 0 < a_j^{new} < C \\ \dfrac{b_i + b_j}{2}, & \text{otherwise} \end{cases} \tag{54}$$

where

$$b_i = E_i + d_i(\boldsymbol{\alpha}_i^{new} - \boldsymbol{\alpha}_i^{old})K_{i,j} + d_j(\boldsymbol{\alpha}_j^{new} - \boldsymbol{\alpha}_j^{old})K_{j,j} + b^{old} \tag{55}$$

and

$$b_j = E_j + d_i(\boldsymbol{\alpha}_i^{new} - \boldsymbol{\alpha}_i^{old})K_{i,i} + d_j(\boldsymbol{\alpha}_j^{new} - \boldsymbol{\alpha}_j^{old})K_{i,j} + b^{old} \tag{56}$$

It is important to state that if $0 < a_i^{new} < C$ and $0 < a_j^{new} < C$ then b_i is equal to b_j.

The updated value for $\mathbf{E}^{new}$ is

$$\mathbf{E}^{new} = \mathbf{E}^{old} + d_i(\alpha_i^{new} - \alpha_i^{old})\mathbf{k}_i + d_j(\alpha_j^{new} - \alpha_j^{old})\mathbf{k}_j + (b - b^{New}) \tag{57}$$

where $\mathbf{k}_i$ corresponds to the *i*-th column of matrix **K**. The complete pseudo-code implementation for the SMO algorithm is given on Tables 1, 2, and 3. It is also made available online at www.lps.ufrj.br/profs/campos. To use it, it is necessary to provide the kernel matrix ($\mathbf{X}^T\mathbf{X}$ in the linear case), the vector of classification labels, the value of the regularization parameter *C* and the maximal number of iterations. The program will return the vector of Lagrange multipliers, the bias parameter and the actual number of iterations spent.

Table 1: Program Listing – SMO

```
function [alpha,b,N iter] = smo(Kernel,d1,C1,LIMIT)
clear global X K d C sv E alpha b
global X K d C sv E alpha b
K = Kernel; d = d1; C = C1;
E = -d;             % Vector of Training error
alpha = zeros(N,1); % Lagrange Multipliers
b = 0;% Bias paremeter
ExamineAll = 1;     % Control Flag
N iter = 1;         % Number of iterations
NKKTV = 0;          % Number of KKT Violations
while (ExamineAll | NKKTV > 0)&(N iter<LIMIT)
       sv = find((alpha ~= 0)&(alfa ~= C));
       NKKTV = 0;
       if ExamineAll
              for i=1:N
                     NKKTV = NKKTV + ExamineExample(i);
              end
              ExamineAll = 0;
       else
              for i=1:length(sv)
                     NKKTV = NKKTV + ExamineExample(sv(i));
              end
              if NKKTV == 0
                     ExamineAll = 1;
              end
       end
       N iter = N iter + 1;
end
```

Table 2: Program Listing – ExamineExample

```
function result=ExamineExample(i2)
global X K d C sv E alpha b
result=0; tol=0.001;
r2=E(i2)*d(i2);
if ( (r2<-tol) & (alpha(i2)<C) ) | ( (r2>tol) & (alpha(i2)>0) )
       % KKT condition violation - use heuristic rules to choose 2nd pattern
       % 1th rule (choose the most promising sv cadidate)
       if length(sv)>0
              if E(i2) >0
                     [value,i]=min(E(sv));
              else
                     [value,i]=max(E(sv));
              end
              if TakeStep(sv(i),i2)
                     result=1;
                     return;
              else
                     % 2nd rule (Search for other sv candidate)
                     [value,k]=sort(rand(1,length(sv)));
                     for i=1:length(sv)
                            if TakeStep(sv(k(i)),i2)
                                   result=1;
                                   return;
                            end
                     end
              end
       end
       % 3rd rule (Search all non sv candidates)
       i1=find( (alpha==0) |(alpha==C) );
```

```
	if length(i1) >0
		[value,k]=sort(rand(1,length(i1)));
		for i=1:length(i1)
			if TakeStep(i1(k(i)),i2)
				result=1;
				return;
			end
		end
	end
end
```

Table 3: Program Listing – TakeStep

```
function result=TakeStep(i1,i2)
global X K d C sv E alpha b
result=0; tol2=10^ (-4);
if i1==i2
	return;
end
s=d(i1)*d(i2);
if s ~= 1
	L=max([0,alpha(i2)-alpha(i1)]); H=min([C,C+alpha(i2)-alpha(i1)]);
else
	L=max([0,alpha(i2)+alpha(i1)-C]); H=min([C,alpha(i2)+alpha(i1)]);
end
if L==H
	return;
end
eta=2*K(i1,i2)-K(i1,i1)-K(i2,i2);
if eta<0
	a2=alpha(i2)-d(i2)*(E(i1)-E(i2))/eta;
	if a2 <L
		a2=L;
	elseif a2>H
		a2=H;
	end
else
	gama=alpha(i1)+s*alpha(i2);
	v1 = E(i1)+d(i1)-alpha(i1)*d(i1)*K(i1,i1)-d(i2)*alpha(i2)*K(i1,i2)+b;
	v2 = E(i2)+d(i2)-alpha(i1)*d(i1)*K(i1,i2)-d(i2)*alpha(i2)*K(i2,i2)+b;
	Lobj = L*(1-s) - 0.5*K(i1,i1)*(gama-s*L)^ 2 - 0.5*K(i2,i2)*L^ 2 - s*K(i1,i2)*( ...
gama-s*L)*L - d(i1)*(gama-s*L)*v1-d(i2)*L*v2;
	Hobj = H*(1-s) - 0.5*K(i1,i1)*(gama-s*H)^ 2 - 0.5*K(i2,i2)*H^ 2 - s*K(i1,i2)*( ...
gama-s*H)*H - d(i1)*(gama-s*H)*v1-d(i2)*H*v2;
	if Lobj >Hobj+eps
		a2=L;
	elseif Lobj <Hobj-eps
		a2=H;
	else
		a2=alpha(i2);
	end
end
if a2 <tol2
	a2=0;
elseif a2 >C - tol2
	a2=C;
end
if abs(a2-alpha(i2)) <eps*(a2+alpha(i2)+eps)
	return
end
a1=alpha(i1)+s*(alpha(i2)-a2);
% calculation of the bias parameter
if (a2>0)&(a2 <C)
	b New=E(i2)+d(i1)*(a1-alpha(i1))*K(i1,i2)+d(i2)*(a2-alpha(i2))*K(i2,i2)+b;
else
	if (a1>0)&(a1<C)
		b New=E(i1)+d(i1)*(a1-alpha(i1))*K(i1,i1)+d(i2)*(a2-alpha(i2))*K(i1,i2)+b;
```

```
        else
                b1=E(i1)+d(i1)*(a1-alpha(i1))*K(i1,i1)+d(i2)*(a2-alpha(i2))*K(i1,i2)+b;
                b2=E(i2)+d(i1)*(a1-alpha(i1))*K(i1,i2)+d(i2)*(a2-alpha(i2))*K(i2,i2)+b;
                b New=(b1+b2)/2;
        end
end
% calculation of the training error
E=E+d(i1)*(a1-alpha(i1))*K(:,i1)+d(i2)*(a2-alpha(i2))*K(:,i2)+(b-b New);
% Update of Lagrange multipliers and bias
b=b New;
alpha(i1)=a1;
alpha(i2)=a2;

result=1;
```

3.2 PSVM Algorithm and Implementation

The PSVM Algorithm is described by Eqs. 25, 26 and 28, for the linear case, and by Eqs. 29 and 30 for the nonlinear case. Table 4 shows a pseudo-code implementation for the PSVM algorithm, where **X** is the data matrix for the linear case, or the kernel matrix for the nonlinear case and **D** is the classification label matrix. In the linear case, $\mathbf{w}_o$ and b_o are returned in variables w0 and b0, respectively. In the nonlinear case, $\mathbf{D}\boldsymbol{\alpha}_o$ (not $\boldsymbol{\alpha}_o$) is returned in variable w0, and b_o is returned in variable b0.

Table 4: Program Listing – PSVM

```
function [w0,b0]=psvm(X,D,C,linear)
M = size(X,1); % Number of parameters (in linear case)
N = size(X,2); % Number of observations
e = ones(N,1);
H = D*[X' -e];
alpha0 = H*((eye(M+1)/C+H'*H)\(H'*e));
alpha0 = C*(e - alpha0);
b0 = sum(D*alpha0);
if linear
        w0 = X*D*alpha0;
else
        w0 = alpha0;
end
```

3.3 UPSVM Algorithm and Implementation

The UPSVM Algorithm is described by Eqs. 38 and 39, for the linear case, and by Eqs. 45 and 46, for the nonlinear case. Table 5 shows a pseudo-code implementation for the proposed algorithm, where **X** is the data matrix for the linear case (*linear* = 1), or the kernel matrix for the nonlinear case (*linear* = 0), and **D** is the classification label matrix. In the linear case, $\mathbf{w}_o$ and b_o are returned in variables w0 and b0, respectively. In the nonlinear case, $\mathbf{D}\boldsymbol{\alpha}_o$ (not $\boldsymbol{\alpha}_o$) is returned in variable w0, b_o is returned in variable and b0.

Table 5: Program Listing – UPSVM

```
function [w0,b0]=upsvm(X,D,C,linear)
M = size(X,1); % Number of parameters
N = size(X,2); % Number of observations
e = ones(N,1);
d = D*e; % Classification vector
A = (eye(N)-e*e'/N);
if linear
        H = eye(M)/C+X*A*X';
        w0 = H\(X*d);
        b0 = -mean(w0'*X);
else
        H = eye(N)/C+ A*X;
        w0 = C(1-H\(A*X*d));
        b0 = -mean(w0'*X);
```

```
end
```

4 Conclusions

Support Vector Machines is an important and increasingly used tool for Pattern classification and Regression applications. This Tutorial detailed three algorithms (SMO, PSVM, UPSVM) used for training SVM to be applied to pattern classification. All algorithms discussed can be used to train linear or nonlinear classifiers that can be used to classify linear or nonlinear separable classes, respectively. Their Matlab implementations are also presented and detailed.

The SMO algorithm solves the original L1SVM quadratic-programming (QP) minimization problem iteratively by working with only two Lagrange Multipliers at a time while maintaining the rest unchanged. Differently from other known fast training algorithms, the SMO does not need an external QP optimizer. The small QP problem solved in each iteration is done analytically which gives to SMO its fastness.

In the PSVM formulation derived as an evolution from the Active SVM (ASVM), the original L2SVM is modified to obtain an analytical solution based on a small system of linear equations in the case of linearly separable classes. Unfortunately the optimal hyperplane can be biased when low values of the regularization parameter C is considered. The UPSVM circumvent this problem by eliminating the bias parameter from the objective function to be minimized, always leading to an unbiased optimal hyperplane. Another difference between UPSVM and PSVM is that the former uses a conventional and more correct approach to generalize the linear classifier to the nonlinear case.

5 References

[1] V.N.Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1st edition, 1998.

[2] Edgar Osuna, Robert Freund, and Federico Girosi, ``Training Support Vector Machines: an Application to Face Detection,'' *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1997, pp. 130--136.

[3] Daniel J. Sebald and James A. Bucklew ``Support Vector Machine Techniques for Nonlinear Equalization,'' *IEEE Transactions on Signal Processing*, 2000, vol 48, no. 11, pp. 3217--3226.

[4] Klaus-Robert Müller and Alex J. Smola and Gunnar Rätsch and Bernhard Schölkopf and Jens Kohlmorgen and Vladimir Vapnik, ``Predicting Time Series with Support Vector Machines,'' *Proceedings of the 7th International Conference on Artificial Neural Networks*, 1997, pp. 999--1004.

[5] Haigin YangLaiwan Chan, and Irwin King ``Support Vector Machine Regression for Volatile Stock Market Prediction,'' *Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning*, 2002, pp. 391--396.

[6] Kyoung-jae, ``Financial Time Series Forecasting Using Support Vector Machines,'' *Neurocomputing, Volume 55, Issues 1-2*, 2003, pp. 307--319.

[7] Richard O. Duda, Peter E. Hart, and David G. Stork *Pattern Classification*, Wiley, 2nd. edition, 2000.

[8] T. Joachims, *Advances in Kernel Methods --- Support Vector Learning*, vol. 11, chapter Making Large-Scale SVM Learning Practical, pp. 41--56, 1998.

[9] J.C. Platt, *Advances in Kernel Methods --- Support Vector Learning*, vol. 12, chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 41--64, 2000.

[10] O.L. Mangasarian and G. Fung, ``Proximal Support Vector Machine Classifiers,'' in *Proceedings International Conference on Knowledge, Discovery, and Data Mining*, 2001, pp. 64--70.

[11] S.V.N. Vishwanathan, A.J. Smola, and M.N. Murty ``SimpleSVM,'' *Proceedings of the International Conference on Machine Learning*, 2003.

[12] F. A. C. de Bastos and M. L. R. de Campos, ``A Fast Training Algorithm for Unbiased Proximal SVM,'' *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005, pp. 245--248.

[13] Felipe. A. C. de Bastos, ``Classificação de Modulações Baseado em Máquinas de Vetores de Suporte," *Ph.D. Thesis* (in Portuguese) Electrical Engineering Program, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, April, 2007.

[14] S. Haykin, *Neural Networks - A Comprehensive Foundation*, Prentice Hall, 2nd. edition, 1999.

[15] S. Abe, ``Analysis of Support Vector Machines," *Proceedings of the IEEE Workshop On Neural Networks for Signal Processing*, 2002, pp. 89--98

[16] J.C. Platt, N. Cristianini, and J.S. Taylor, ``Large Margin Dags for Multiclass Classification," in *Advances in Neural Information Processing Systems (NIPS)*, 2000, vol. 12, pp. 547--553.

[17] K.-L. Li, Z.-F. Tian, and H.-H. Huang, ``A Novel Multiclass SVM Classifier Based on Ddag," in *Proceedings Of First International Conference On Machine Learning And Cybernetics*, 2002, vol. 3, pp. 1203--1207.

[18] O.L. Mangasarian and D.R. Musicant, ``Active Support Vector Machine Classification," in *Advances in Neural Information Processing Systems (NIPS)*, 2000, vol. 13, pp. 577--583.

[19] G. H. Gollub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins, 3rd edition, 1996.