A Column Generation Approach for Support Vector Machines *

Emilio Carrizosa

Universidad de Sevilla (Spain). ecarrizosa@us.es

Belén Martín-Barragán

Universidad de Sevilla (Spain). belmart@us.es

Dolores Romero Morales

University of Oxford (United Kingdom). dolores.romero-morales@sbs.ox.ac.uk

23rd March 2006

Abstract

The widely used Support Vector Machine (SVM) method has shown to yield good results in Supervised Classification problems. Other methods such as Classification Trees have become more popular among practitioners than SVM thanks to their interpretability, which is an important issue in Data Mining.

In this work, we propose an SVM-based method that automatically detects the most important predictor variables, and those values which are critical for the classification. Its classification ability is comparable to the standard linear SVM and clearly better than Classification Trees. Moreover, the proposed method is robust, i.e., it is stable in the presence of outliers and invariant to change of scale or measurement units of the predictor variables. The method involves the optimization of a Linear Programming problem with a large number of decision variables, for which we use the well-known

*This work has been partially supported by projects MTM2005-09362-C03-01 of MEC, Spain, and FQM-329 of Junta de Andalucía, Spain.

Column Generation technique.

When the classification rule obtained is too complex to allow interpretability, a wrapper feature selection method is applied, yielding a classification rule whose behavior slightly differs from linear SVM and still remains better than Classification Trees.

Keywords: Data Mining, Supervised Classification, Column Generation, Support Vector Machines

1 Introduction and literature review

Classifying objects or individuals into different classes or groups is one of the aims of Data Mining. This topic has been addressed in different areas such as Statistics, Operations Research and Artificial Intelligence. A general introduction of Data Mining can be found in [9].

We focus on the well-known so-called Supervised Classification problem, usually referred as Discriminant Analysis by statisticians, where we have a set of objects Ω and the aim is to build a classification rule which predicts the class membership c^u of an object u into one of a predefined set of classes, by means of its *predictor vector* x^u. The *predictor vector* x^u takes values in a set X which is usually assumed to be a subset of $\mathbb{R}^p$, such as $\{0,1\}^p$, and the components x_ℓ, $\ell = 1, 2, \ldots, p$, of the predictor vector x are called *predictor variables*. The other piece of information defining the object u, called the *class-membership* c^u, takes values in the set of classes $\mathcal{C}$. Object u is said to belong to class c^u.

Not all the information about the objects in Ω is available, but only in a subset I, called the *training sample*, where both predictor vector and class-membership of the objects are known. With this information, the classification rule must be built.

In the last fifteen years, Mathematical Programming techniques have been successfully applied to Data Mining problems, see for instance [2, 3, 16]. The Support Vector Machines (SVM) [4] approach is based on margin maximization, which consists in finding the hyperplane which is furthest from the closest object. SVM has shown to be a very powerful tool for Supervised Classification. The most popular versions of SVM embed, via a kernel function, the original predictor variables into a higher (possibly

infinite) dimensional space, [11]. In this way one obtains classifiers with good generalization properties but, in general, hard to interpret.

In some application fields, practitioners, such as doctors or businessmen, may be very unwilling to use a classifier they cannot interpret. For them, Data Mining methods sometimes proceed like a black-box, so they would not feel confident enough to use the classifier unless they can interpret it somehow.

For instance, it is easy to interpret and manage queries of type

- Is predictor variable ℓ_1 big?
- Is predictor variable ℓ_2 small?
- Does predictor variable ℓ_3 attain a very extreme value?

where the concept of 'big', 'small' and 'extreme value' must be quantified, e.g. in the form

$$\boxed{\text{is predictor variable } \ell \text{ greater than or equal to } b?} \tag{1}$$

This type of queries are used e.g. in Classification Trees. In this way, practitioners can interpret the classifier, describing how it works. Moreover, they can directly see which role the different predictor variables are playing in the classifier, and detect the values of a predictor variable critical for the classification.

In this paper we propose a new model that, by using piecewise constant functions, automatically selects the adequate scale for each variable. The obtained rule is based on queries of type (1), which makes interpretability easier. Moreover, as shown empirically, the obtained classifiers are robust against the presence of outliers.

We restrict ourselves to the case in which two classes exist, $\mathcal{C} = \{-1, 1\}$. The multiclass case can be reduced to a series of two-class problems, as has been suggested e.g. in [10, 11, 20].

We work in an SVM-based framework, where the feature space is defined by binarizing each variable, and considering all the possible cutoffs. For this reason, we call our method the Binarized Support Vector Machine, BSVM. Numerical results show that the proposed approach gives a classifier that behaves similar to the standard linear SVM and clearly better than Classification Trees. Moreover, the tradeoff between

interpretability and classification ability can be controlled via an upper bound on the number of features allowed.

The classifier proposed in this paper, BSVM, is described in Section 2. Since the number of features to be considered may be huge, the BSVM method yields an optimization problem with a large number of decision variables, namely $\sharp(I)^p$, where $\sharp(\cdot)$ denotes the cardinality of a set. In Section 3, a Column-Generation-based algorithm is proposed in order to solve such an optimization problem. Numerical results are shown in Section 4, whereas conclusions and some lines for future research are discussed in Section 5.

2 Binarized Support Vector Machines

In practical applications, simple rules of type (1) are very desirable because of their interpretability. For example, a doctor would say that having high blood pressure is a symptom of disease. Choosing the threshold b from which a specific blood pressure would be considered high is not usually an easy task.

We theoretically consider all the possible rules of type (1), mathematically formalized by the function

$$\phi_{\ell b}(x) = \begin{cases} 1 & \text{if } x_\ell \geq b \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

for $b \in I\!R$ and $\ell = 1, 2, \ldots, p$. In what follows, each predictor variable is called *variable*, whereas each function of type $\phi_{\ell b}$ is called *feature*.

This binarization procedure could be done in a tedious preprocessing step, where all the possible features are created. However, we do not do it as a preprocessing step but, as we will see later, propose a method to generate features when needed.

The set of possible cutoff values b (and, thus the number of features) is, in principle, infinite. However, given a training sample I, many of those possible cutoffs will yield exactly the same classification in the objects in I, which are the objects whose information is available. In this sense, for a certain predictor variable ℓ and a given training sample I, we can constrain the choice of $b \in I\!R$ to the finite set

$B_\ell = \{x_\ell^u : u \in I\}$. In this way, the family of features under consideration is given by

$$\mathcal{F} = \{\phi_{\ell b} : b \in B_\ell,\, \ell = 1, 2, \ldots, p\}.$$

These features are used to classify in the following way: each feature $\phi_{\ell b}$ has associated a weight $\omega_{\ell b}$, measuring its contribution for the classification into the class -1 or 1. The weighted sum of those features and a threshold β constitute the *score function*,

$$f(x) = \omega^\top \Phi(x) + \beta = \sum_{\ell=1}^{p} \sum_{\{b \in B_\ell |\, x_\ell \geq b\}} \omega_{\ell b} + \beta, \tag{3}$$

where $\Phi(x) = (\phi_{\ell b}(x))_{b \in B_\ell,\, \ell = 1,2,\ldots,p}$ and $\omega^\top \Phi(x)$ denotes the scalar product of vectors ω and $\Phi(x)$.

Objects will be allocated to class -1 if $f(x) < 0$, and to class 1 if $f(x) > 0$. In case of ties, i.e. $f(x) = 0$, objects can be allocated randomly or by some predefined order. In this paper, following a worst case approach, they will be considered as misclassified.

For a certain predictor variable ℓ, the coefficient $\omega_{\ell b}$ associated to feature $\phi_{\ell b}$ represents the amount with which the query 'is $x_\ell \geq b$?' contributes to the score function (3). Hence, the weight $\omega_{\ell b}$ gives insightful knowledge about how predictor variable ℓ influences the classification, since we are replacing variable ℓ by the function $s \mapsto \sum_{\{b \in B_\ell |\, s \geq b\}} \omega_{\ell b}$, thus determining the change of scale to be applied to variable ℓ. Moreover, those variables ℓ for which $\omega_{\ell b}$ are zero for all $b \in B_\ell$ are not needed for the classification, and can be discarded. For instance, taking the Wisconsin Breast Cancer Database from the UCI Machine Learning Repository, [14], with data from cancer diagnosis, and using the BSVM, it turns out that only 12 out of 30 variables have at least one non-zero $\omega_{\ell b}$. In other words, only 12 out of the 30 variables are relevant for the classification. In Figure 1 we show, for each of these twelve variables, its contribution to the score function. As an illustration, Table 1 shows, for variable `Worst Texture`, its cutoffs b, the corresponding weights $\omega_{\ell b}$ and the cumulative weights $\sum_{b' \leq b} \omega_{\ell b'}$.

We have also plotted in Figure 1 the median (represented by a star) and the mean (represented as a cross). It can be seen how, although the mean, or the median, is sometimes a good choice for the cutoff, this does not happen in general, and BSVM prefers other choices.

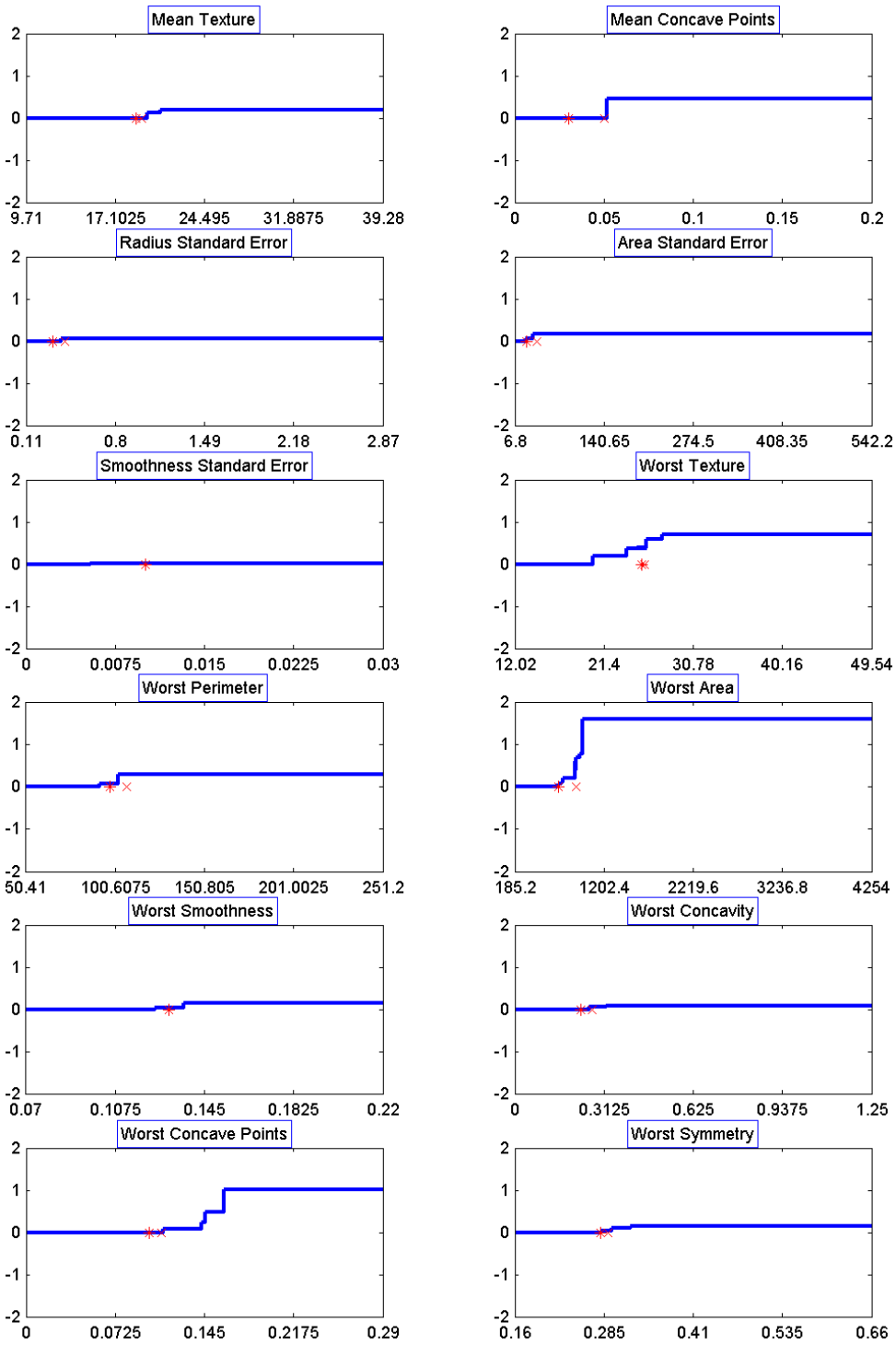

Figure 1: Automatic choice of the scales

b	$\omega_{\ell b}$	$\sum_{b' \leq b} \omega_{\ell b'}$
20.24	0.17795	0.17795
23.75	0.19729	0.37524
25.73	0.01160	0.38685
25.84	0.20116	0.58801
27.57	0.11219	0.70019

Table 1: Scale for the variable `Worst Texture`.

Since the output of the features proposed in this paper is always binary, the importance represented by the coefficient is always measured in the same scale. The practitioner could choose to interpret those features with the highest coefficient in absolute value, obtaining in this way a great insight in the behavior of the classifier. However, if the practitioner prefers to keep the number of features low in order to get a complete picture about how the classifier works, then a wrapper feature selection procedure, as in [7], could be used to gain interpretability. In Section 4.3 we give some numerical results using a simple but powerful wrapper procedure known as *recursive feature elimination*, as first proposed in [8].

In order to choose ω and β we follow an SVM-based approach which consists in finding the hyperplane which maximizes the margin in the feature space, i.e. the space of the images $\Phi(x^u)$ of the objects u in the training sample I. The use of margin maximization is theoretically motivated by the bounds on the generalization ability [17, 19, 20], where the probability of misclassifying a forthcoming individual is bounded by a function which is decreasing in the margin. The so-called *hard-margin* SVM approach proposes the choice of the hyperplane with maximal margin, i.e. the hyperplane which correctly classifies all objects in I and is furthest from the closest object. In contrast, the so-called *soft-margin* approach, allows some objects to be misclassified. We use this latter version as it has been empirically shown to avoid overfitting, a phenomenon which happens when a low misclassification rate in I does not generalize

to forthcoming objects. The soft-margin maximization problem is formulated in this paper by

$$\begin{array}{lll} \min & \|\omega\| + C\sum_{u\in I}\xi^u & \\ s.t.: & c^u\left(\omega^\top\Phi(x^u)+\beta\right)+\xi^u \geq 1 & \forall u\in I \\ & \xi^u \geq 0 & \forall u\in I \\ & \omega\in I\!R^N, \beta\in I\!R, & \end{array} \tag{4}$$

where the decision variables are the weight vector ω, the threshold value β and perturbations ξ^u associated with the misclassification of object $u \in I$. $\|\cdot\|$ denotes the L_1 norm, $N = \sum_{\ell=1}^{p} \sharp(B_\ell)$ and C is a constant that trades off the margin in the correctly classified objects and the perturbations ξ^u. An appropriate value of C is usually chosen by crossvalidation techniques, see e.g. [12].

Whereas the distance between points has usually been considered in the literature as the Euclidean norm, yielding the margin to be measured by the Euclidean norm as well, other norms such as the L_1 norm and the L_∞ norm have been considered, [13], and have been successfully applied, see for instance [1, 18, 21].

Contrary to the Euclidean case, in which a maximal margin hyperplane can be found by solving a quadratic program with linear constraints, in the L_1 norm case, used in this paper, an optimal solution can be found by solving a Linear Programming (LP) problem. In [15], empirical results show that 'in terms of separation performance, L_1, L_∞ and Euclidean norm-based SVM tend to be quite similar'. Moreover, polyhedral norms, as L_1 norm, contribute to sparsity in the classifier, yielding ω with many components equal to zero, see for instance [5].

Since we use the L_1 norm, Problem (4) can be formulated as the following LP problem,

$$\begin{array}{lll} \min & \sum_{\ell=1}^{p}\sum_{b\in B_\ell}(\omega^+_{\ell b}+\omega^-_{\ell b}) + C\sum_{u\in I}\xi^u & \\ s.t.: & \sum_{\ell=1}^{p}\sum_{b\in B_\ell}(\omega^+_{\ell b}-\omega^-_{\ell b})c^u\phi_{\ell b}(x^u) + \beta c^u + \xi^u \geq 1 & \forall u\in I \\ & \omega^+_{\ell b}\geq 0 & \forall b\in B_\ell, \forall \ell = 1,2,\ldots,p \\ & \omega^-_{\ell b}\geq 0 & \forall b\in B_\ell, \forall \ell = 1,2,\ldots,p \\ & \xi^u \geq 0 & \forall u\in I \\ & \beta\in I\!R. & \end{array} \tag{5}$$

After finding the maximal margin hyperplane in the feature space defined by $\mathcal{F}$, the score function has the form described in (3).

For each feature, the absolute value of its coefficient indicates the importance of that feature for the classification. Using basic Linear Programming theory, it is easy to see that the number of features with non-zero coefficient is not larger than the number of objects in the database. However, this number can still be large for interpretability of the classifier.

3 Column Generation

In this paper, we propose Problem (5) to be solved by the well-known Mathematical Programming tool called Column Generation, initially introduced for the cutting-stock problem [6]. Instead of solving directly Problem (5), which has a high number of decision variables, the Column Generation technique solves a series of reduced problems where decision variables, corresponding to features in the set $\mathcal{F}$, are iteratively added as needed.

For $F \subset \mathcal{F}$, let *Master Problem* (5-F) be Problem (5) with the family of features F. In each iteration, we solve the Problem (5-F). The next step is to check whether the current solution is optimal for Problem (5) or not, and, in the latter case, generate a new feature ϕ improving the objective value of the current solution. The generated feature is added to the family of features F. This process is repeated until optimality is reached.

In order to generate new features, the Column Generation technique uses the dual formulation of Problem (5),

$$\begin{array}{ll} \max & \sum_{u\in I} \lambda_u \\ s.t.: & -1 \leq \sum_{u\in I} \lambda_u c^u \phi(x^u) \leq 1 \quad \forall \phi \in \mathcal{F} \\ & \sum_{u\in I} \lambda_u c^u = 0 \\ & 0 \leq \lambda_u \leq C \quad u \in I. \end{array} \tag{6}$$

The dual formulation of the Master Problem (5-F) only differs from this one in the first set of constraints, which should be attained $\forall \phi \in F$ instead of $\forall \phi \in \mathcal{F}$.

Let (ω^*, β^*) be an optimal solution of Master Problem (5-F), and let $(\lambda_u^*)_{u\in I}$ be the values of the corresponding optimal dual solution. If the optimal solution of the Master Problem (5-F) is also optimal for Problem (5), then, for every feature $\phi \in \mathcal{F}$ the constraints of the Dual Problem (6) will hold, i.e.

$$-1 \leq \sum_{u\in I} \lambda_u^* c^u \phi(x^u) \leq 1. \tag{7}$$

Denote $\Gamma(\phi) = \sum_{u\in I} \lambda_u^* c^u \phi(x^u)$. If (ω^*, β^*) is not optimal for Problem (5), then the most violated constraint gives us information about which feature is promising and could be added to F, in the sense that adding such a feature to the set F would yield, at that iteration, the highest improvement of the objective function. Thus, we wish to generate a new feature $\phi \in \mathcal{F}$ maximizing $|\Gamma(\phi)|$. Finding such a ϕ can be reduced to solving two optimization problems:

- $\max_{\phi\in\mathcal{F}} \Gamma(\phi)$,
- $\min_{\phi\in\mathcal{F}} \Gamma(\phi)$.

In the remainder of this Section we give a more detailed description of the implementation of the Column Generation algorithm.

3.1 Initial set of features

In the column generation procedure new features are sequentially added to the problem, based on the dual values of the current solution. The Column Generation technique starts with an initial Master Problem, i.e. a set of features F_0 must be chosen to initialize the algorithm. We have chosen to start with one feature per variable, with the cutoff set equal to its median in the objects of I.

3.2 Generation of features

Finding the best $\phi \in \mathcal{F}$ is reduced to finding a predictor variable ℓ and a cutoff $b \in B_\ell$, such that $|\Gamma(\phi_{\ell b})|$, with $\phi_{\ell b}$ defined by (2), is maximal. In this section we describe an algorithm for, fixed the predictor variable ℓ, finding the cutoff b maximizing $\Gamma(\phi_{\ell b})$. Finding the minimum can be done in a similar way.

First, we sort all the objects in decreasing order by the values of the predictor variable ℓ. Denote by $u(i)$ the object in i-th position. For simplicity, suppose there are not repeated values, i.e. $x_\ell^{u(1)} > x_\ell^{u(2)} > \ldots > x_\ell^{u(\sharp(I))}$. The case with repeated values will be analyzed later.

Once ℓ is fixed and all the objects are sorted by the values of the predictor variable ℓ, the value $\Gamma(\phi_{\ell b})$ can be efficiently calculated with a recursive procedure. Indeed, for certain $i \in \{1, 2, \ldots, \sharp(I)\}$, we have $\Gamma(\phi_{\ell b_{i+1}}) = \Gamma(\phi_{\ell b_i}) + \lambda^*_{u(i+1)} c^{u(i+1)}$, where b_i denote the cutoff chosen as $x^{u(i)}$. Moreover, since λ^*_u is nonnegative for all $u \in I$, whenever $c^u(i+1) = 1$, then $\Gamma(\phi_{\ell b_{i+1}}) > \Gamma(\phi_{\ell b_i})$. Thus, checking whether $\phi_{\ell b_i}$ is a maximum is not needed for every i, but only for those i such that $c^{u(i)} = 1$ and $c^{u(i+1)} = -1$.

In the case in which there are repeated values in $\{x^u_\ell : u \in I\}$, the rule above does not apply. Let i and t be such that $x_\ell^{u(i-1)} > x_\ell^{u(i)} = x_\ell^{u(i+1)} = \ldots = x_\ell^{u(i+t)} > x_\ell^{u(i+t+1)}$. Note that, in the set of objects where predictor variable ℓ has the same value, there could be objects belonging to different classes. In this case, then $b = b_i$ must be checked, whatever the value of $c^{u(i+t+1)}$. However, if $c^{u(i)} = c^{u(i+1)} = \ldots = c^{u(i+t)}$ and $c^{u(i+t+1)} = 1$ we know that setting $b = b_j$, for any $j = i, i+1, \ldots, i+t$, will be improved by setting $b = b_{i+t+1}$. This means that $b = b_i$ does not give a maximum of $\Gamma(\phi_{\ell b})$. Only if $c^{u(i)} = 1$ and $c^{u(i+t+1)} = -1$, it is worth to consider $b = b_i$ as a candidate to be the maximum.

The minimization of Γ is done analogously. For example, in case of no repeated values, candidates to be a minimum correspond to objects $u(i)$ belonging to class -1 where the next object $u(i+1)$ belongs to class 1.

Taking into account all these considerations we obtain, for a fixed predictor variable ℓ, given the dual values λ^*_u, the algorithm described below, which finds the cutoff b^+_ℓ (and respectively, b^-_ℓ) for which $\Gamma(\phi_{\ell b^+_\ell})$ (respectively, $\Gamma(\phi_{\ell b^-_\ell})$) is maximal (respectively, minimal).

Algorithm 1: Choosing a cutoff for variable ℓ.

Step 0. Sort the objects by $x_\ell : x_\ell^{u(i)}$.

Step 1. Set $i \leftarrow 1$, $sum \leftarrow 0$, $max \leftarrow 0$ and $min \leftarrow 0$.

Step 2. Set $sum \leftarrow sum + \lambda^*_{u(i)} c^{u(i)}$.

Step 3. **Step 3.0.** If $x_\ell^{u(i)} = x_\ell^{u(i+1)}$, then, go to Step 4.

Step 3.1. Otherwise, if for some $t > 0$, $x_\ell^{u(i-t-1)} < x_\ell^{u(i-t)} = \ldots = x_\ell^{u(i)} < x_\ell^{u(i)-1}$ and there exists j with $j = 1, \ldots, t$ and $c^{u(i)} \neq c^{u(i-j)}$, then:

- If *sum*>*max*, then set $max \leftarrow sum$.
- If *sum*<*min*, then set $min \leftarrow sum$.

Step 3.2. Otherwise,

- if $c^{u(i)} = 1$, $c^{u(i+1)} = -1$ and *sum*>*max*, then set $max \leftarrow sum$.
- if $c^{u(i)} = -1$, $c^{u(i+1)} = 1$ and *sum*<*min*, then set $min \leftarrow sum$.

Step 4. Set $i \leftarrow i + 1$. If $i \leq \sharp(I)$, then go to Step 2, otherwise STOP.

3.3 Implementation details

The column generation algorithm has been implemented as follows. The initial set of features F_0 is built, as described in Section 3.1, using features whose cutoffs are the medians of the predictor variables. Then, Problem (5-F_0) is solved for such initial set of features. The dual values of the optimal solution found, are used to generate new features.

In every step of the column generation algorithm, instead of generating just one feature (the one maximizing $|\Gamma(\phi)|$), we generate two features for every predictor variable ℓ, given by the cutoffs for which $\Gamma(\phi_{\ell b})$ is maximal and minimal. This is done using Algorithm 1, as described in Section 3.2. We do it for all the predictor variables, thus obtaining $2p$ features. Those generated features having $|\Gamma(\phi)| > 1$ are added to F and the LP problem (5-F) is solved. These steps are repeated until all the generated features have $|\Gamma(\phi)| \leq 1$, in which case, we have found an optimal solution of Problem (5). A summary of this Column Generation Algorithm is described as follows.

Algorithm for Column Generation.

Step 0. Set $F_0 \leftarrow \{\phi_{1b_1^*}, \phi_{2b_2^*}, \ldots, \phi_{pb_p^*}\}$, where b_ℓ^* is the median of the predictor variable ℓ, for $\ell = 1, 2, \ldots, p$. Set $F \leftarrow F_0$.

Step 1. Solve Problem (5-F). Let (ω^*, β^*) be its optimal solution, with dual values λ_u^*, $\forall u \in I$.

Step 2. For each $\ell = 1, 2, \ldots, p$ do:

Step 2.0. Run **Algorithm 1** to choose b_ℓ^+.

Step 2.1. If $\Gamma(\phi_{\ell b_\ell^+}) > 1$, then set $F \leftarrow F \cup \{\phi_{\ell b_\ell^+}\}$.

Step 2.2. Run **Algorithm 1** to choose b_ℓ^-.

Step 2.3. If $\Gamma(\phi_{\ell b_\ell^-}) < -1$, then set $F \leftarrow F \cup \{\phi_{\ell b_\ell^-}\}$.

Step 3. If F has been modified, then go to Step 1, otherwise STOP: we have found an optimal solution of Problem (5).

4 Numerical results

4.1 Databases

First we are going to analyze the classification ability of the set of features proposed in the paper. With this aim, a series of numerical experiments have been performed using databases publicly available from the UCI Machine Learning Repository [14]. A summary of the characteristics of the databases used in the experiments is shown in Table 2. Five different databases were used, namely, the Cylinder Bands Database, called here `bands`; the Credit Screening Databases, called here `credit`; the Ionosphere Database, called here `ionosphere`; the Sonar Database, called here `sonar`; and the New Diagnostic Database, contained in the Wisconsin Breast Cancer Databases, called here `wdbc`.

For each database, the name of the file (as called in the database), the number of predictor variables (all quantitative) p and the total number of objects $\sharp(\Omega)$ are given in Table 2. In case of existence of missing values, as occurs in `bands` and `credit`, objects with missing values have been removed from the database. In databases `bands` and `credit`, some of the predictor variables were nominal. Each of these predictor variables has been replaced by a set of binary variables in the following way: for every possible

value $\hat{x}$ of the original nominal variable ℓ, a new binary variable is built taking value one when x_ℓ is equal to $\hat{x}$ and zero otherwise.

name	filename	p	$\sharp(\Omega)$
`bands`	bands.data	56	365
`credit`	crx.data	43	666
`ionosphere`	ionosphere.data	34	351
`sonar`	sonar.all-data	60	208
`wdbc`	wdbc.data	30	569

Table 2: Information about the databases.

In order to compare the quality of the BSVM classifier with the classification quality of other classifiers, we have tested the performance of two very different benchmark methods: Classification Trees, both with pruning (prTree) and without pruning (Tree), and SVM with linear kernel. All results presented are obtained by 10-fold crossvalidation, e.g. [12]. The average percentages of correctly classified objects in both the training sample (`tr`) and testing sample (`test`) are displayed in Table 3 for different values of the parameter C. Finding a procedure to derive an appropriate choice of the value C, which trades off the margin and the deviations of the misclassified objects, is beyond the purpose of this paper. CPLEX 8.1.0 was used as the LP solver.

It is a well-known fact in SVM that, if the parameter C is chosen too close to zero, one may obtain as optimal solution of Problem (5) a vector with $\omega = 0$, from which a trivial classifier assigning all objects to one class is obtained. This degenerate situation (indicated in Tables 4-5 as `d.c.`) is avoided by taking a bigger C.

4.2 Comparing with other techniques

Results of the BSVM are shown in Table 4, where the average percentages of correctly classified objects in the training and testing samples are displayed along with the number of generated features ($\sharp$ `features`) with non-zero coefficient in the classifier, and the number of predictor variables actually used by the classifier ($\sharp$ `var`). As the results show, the BSVM behaves considerably better than Classification Trees and comparable to the standard linear SVM technique. Classification Trees are widely used in applied

fields as diverse as medicine (diagnosis), computer science (data structures), botany (classification), and psychology (decision theory), mainly because they are easy to interpret. We claim that the BSVM maintains this property without losing the good classification ability of the standard linear SVM.

4.3 Reducing the number of features

The results in Table 4 show that the number of features is usually over one or even two hundreds, which makes hard to detect the most relevant features. In order to obtain a more interpretable classifier, we proceed with a pruning procedure in which features are recursively deleted. In this procedure, which has been successfully applied in standard SVM, see [8], all the generated features with zero coefficient in the classifier and the feature with non-zero coefficient having the smallest absolute value are eliminated. Then, the coefficients are recomputed by the optimization of the LP problem (5). This elimination procedure is repeated until the number of features is below a number given in advance.

The average percentages of correctly classified objects in the training and testing sample are shown, in Table 5, when the elimination procedure is applied until 30 or less features remain in the classification rule. As can be seen, the classification ability slightly deteriorates, but still keeps on being better than the one of the Classification Trees.

4.4 Behavior in presence of outliers

The classifier proposed in this paper is based on threshold functions, thus it seems that extreme observations, with very high or very low values, will not have a strong influence in the classifier. To empirically test this fact, a series of experiments have been performed where some outliers were artificially introduced in the database. Every cell in the database `wdbc` was chosen to be an outlier with probability 0.05. Those cells chosen, were modified by adding ρ times the range of its predictor variable, for $\rho = 10, 100, 1000$. In Tables 6 and 7 results of database `wdbc` are shown for the benchmark methods and for the BSVM. The classification ability of the linear SVM classifier dramatically worsens when introducing outliers, whereas the BSVM is hardly affected.

5 Conclusions and further research

In this paper a new SVM-based tool for supervised classification has been proposed where the classifier gives insightful knowledge about the way the predictor variables influence the classification. Indeed, the nonlinearity behavior of the data is modelled by the BSVM classifier using simple queries, of type (1), easily interpretable by practitioners, for instance by representations similar to Figure 1. Its classification behavior, which is between SVM and Classification Trees, makes BSVM an interesting tool when a good classification ability is required, but interpretability of the results is an important issue. Our numerical experience shows that BSVM is much more robust than SVM against outliers.

The binarization procedure has been applied to each variable separately. If interactions between variables are expected to be relevant, more general binarization procedures might be considered. This issue will be addressed in a forthcoming paper.

References

[1] K. Bennet. Combining support vector and mathematical programming methods for induction. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 307–326. MIT Press, 1999.

[2] P.S. Bradley, O.L. Mangasarian, and D. Musicant. Optimization methods in massive datasets. In J. Abello, P.M. Pardalos, and M.G.C. Resende, editors, *Handbook of Massive Datasets*, pages 439–471. Kluwer Academic Publishers, Boston, 2002.

[3] E. Carrizosa and B. Martín-Barragán. Two-group classification via a biobjective margin maximization model. *European Journal of Operational Research*. In press.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] G. Fung and O.L. Mangasarian. A feature selection newton method for support vector machine classification. *Computational Optimization and Applications*, 28(2):185–202, 2004.

[6] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[7] I. Guyon and A. Elisseff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, Mar 2003.

[8] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[9] H. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.

[10] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26(2):451–471, 1998.

[11] R. Herbrich. *Learning Kernel Classifiers. Theory and Algorithms*. MIT Press, 2002.

[12] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann, 1995.

[13] O.L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

[14] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998. University of California, Irvine, Dept. of Information and Computer Sciences.

[15] J.P. Pedroso and N. Murata. Support vector machines with different norms: motivation, formulations and results. *Pattern Recognition Letters*, 22:1263–1272, 2001.

[16] A.M. Rubinov, A.M. Bagirovand, N.V. Soukhoroukova, and J. Yearwood. Unsupervised and supervised data classification via nonsmooth and global optimization. *TOP*, 11(1):1–93, 2003.

[17] J. Shawe-Taylor, P.L. Bartlett, R.C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

[18] A. Smola, T.T. Friess, and B. Schölkopf. Semiparametric support vector and linear programming machines. In M.J. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 10*, pages 585–591. MIT Press, 1999.

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

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

[21] J. Weston, A. Gammerman, M.O. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 293 – 305. MIT Press, 1999.

bands			
method	C	% `tr`	% `test`
SVM	0.01	64.44	64.44
SVM	0.10	74.57	65.93
SVM	1	80.16	71.85
SVM	10	81.65	72.96
SVM	100	82.18	72.22
SVM	1000	82.35	72.59
prTree		74.12	64.81
Tree		92.43	66.67
`credit`			
method	C	% `tr`	% `test`
SVM	0.01	86.36	86.31
SVM	0.10	86.31	86.31
SVM	1	86.74	85.85
SVM	10	86.80	86.00
SVM	100	86.91	85.85
SVM	1000	87.09	85.54
prTree		86.31	86.31
Tree		95.15	81.69
`ionosphere`			
method	C	% `tr`	% `test`
SVM	0.01	66.25	65.71
SVM	0.10	89.21	87.71
SVM	1	91.84	87.71
SVM	10	94.03	88.29
SVM	100	95.21	87.71
SVM	1000	95.65	86.29
prTree		91.17	89.43
Tree		98.03	87.71
`sonar`			
method	C	% `tr`	% `test`
SVM	0.01	54.56	54.00
SVM	0.10	84.33	75.00
SVM	1	88.39	74.50
SVM	10	92.28	73.50
SVM	100	97.06	75.00
SVM	1000	100.00	73.50
prTree		82.56	71.50
Tree		97.56	77.00
`wdbc`			
method	C	% `tr`	% `test`
SVM	0.01	87.16	86.79
SVM	0.10	95.95	95.71
SVM	1	98.27	98.04
SVM	10	98.45	97.68
SVM	100	99.11	96.96
SVM	1000	99.50	96.43
prTree		96.71	94.46
Tree		99.31	93.75

Table 3: Classification behavior in benchmark methods.

bands				
C	% tr	% test	♯ features	♯ var
0.01	d.c.			
0.1	d.c.			
0.1	98.85	73.33	121.1	28.0
10	100.00	72.59	128.8	28.3
100	100.00	72.22	128.5	28.3
1000	100.00	72.59	128.4	28.4
credit				
C	% tr	% test	♯ features	♯ var
0.01	86.31	86.31	1.0	1.0
0.1	86.31	86.31	1.0	1.0
0.1	95.71	82.92	138.2	21.7
10	100.00	80.00	199.5	24.8
100	100.00	79.69	200.3	24.7
1000	100.00	80.31	200.5	24.8
ionosphere				
C	% tr	% test	♯ features	♯ var
0.01	d.c.			
0.1	91.17	90.57	2.0	2.0
0.1	100.00	90.57	92.7	31.1
10	100.00	90.57	93.1	31.2
100	100.00	90.57	93.1	31.2
1000	100.00	90.57	93.4	31.1
sonar				
C	% tr	% test	♯ features	♯ var
0.01	d.c.			
0.1	91.83	75.00	39.2	25.9
0.1	100.00	80.50	97.5	47.8
10	100.00	80.00	97.4	47.8
100	100.00	80.50	97.5	47.8
1000	100.00	80.50	97.5	47.8
wdbc				
C	% tr	% test	♯ features	♯ var
0.01	92.60	90.54	1.0	1.0
0.1	97.74	96.07	21.1	9.0
0.1	100.00	95.71	68.4	24.8
10	100.00	96.25	68.2	25.0
100	100.00	95.89	67.6	25.0
1000	100.00	96.07	68.0	25.0

Table 4: Classification behavior on BSVM.

BSVM (reducing number of features)										
	bands		credit		ionosphere		sonar		wdbc	
C	% tr	% test	% tr	% test	% tr	% test	% tr	% test	% tr	% test
0.01	d.c.		86.31	86.31	d.c.		d.c.		92.60	90.54
0.1	d.c.		86.31	86.31	91.17	90.57	91.94	76.50	97.74	95.89
1	92.14	72.22	91.93	84.00	100.00	90.00	100.00	77.50	100.00	95.71
10	94.81	66.30	89.50	80.92	100.00	89.43	100.00	77.00	100.00	96.07
100	95.47	66.30	90.09	82.00	100.00	89.71	100.00	77.00	100.00	96.07
1000	95.14	66.67	91.42	80.77	100.00	89.43	100.00	77.00	100.00	96.25

Table 5: Reducing the number of features to at most 30.

Benchmark methods with outliers. Database wdbc									
		original		$\rho = 1$		$\rho = 10$		$\rho = 100$	
method	C	% tr	% test	% tr	% test	% tr	% test	% tr	% test
SVM	0.01	87.16	86.79	65.26	65.18	63.21	63.21	63.21	63.21
SVM	0.10	95.95	95.71	90.24	88.75	64.13	63.21	63.47	62.86
SVM	1	98.27	98.04	91.53	90.18	65.65	59.64	64.94	62.86
SVM	10	98.45	97.68	92.24	88.57	64.54	57.68	64.96	61.61
SVM	100	99.11	96.96	92.42	88.57	64.72	57.50	64.94	60.89
SVM	1000	99.50	96.43	92.40	88.75	64.70	57.50	64.94	60.71
prTree		96.71	94.46	96.31	93.04	95.71	92.14	95.99	92.32
Tree		99.31	93.75	98.95	93.75	98.95	93.75	98.95	93.75

Table 6: Classification behavior of benchmark methods with outliers.

BSVM with outliers. Database wdbc								
	original		$\rho = 1$		$\rho = 10$		$\rho = 100$	
C	% tr	% test	% tr	% test	% tr	% test	% tr	% test
0.01	92.60	90.54	90.00	85.89	90.00	85.89	90.00	85.89
0.1	97.74	96.07	98.35	95.36	98.35	95.36	98.35	95.36
1	100.00	95.71	100.00	95.18	100.00	95.18	100.00	95.18
10	100.00	96.25	100.00	95.00	100.00	95.00	100.00	95.00
100	100.00	95.89	100.00	95.00	100.00	95.00	100.00	95.00
1000	100.00	96.07	100.00	95.18	100.00	95.18	100.00	95.18

Table 7: Classification behavior of BSVM with outliers.