Detecting relevant variables and interactions in supervised classification*

Emilio Carrizosa

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

Belén Martín-Barragán

Universidad Carlos III de Madrid (Spain). belen.martin@uc3m.es

Dolores Romero Morales

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

February 25, 2010

Abstract

The widely used Support Vector Machine (SVM) method has shown to yield good results in Supervised Classification problems. When the interpretability is an important issue, then classification methods such as Classification Trees (CART) might be more attractive, since they are designed to detect the important predictor variables and, for each predictor variable, the critical values which are most relevant for classification. However, when interactions between variables strongly affect the class membership, CART may yield misleading information. Extending previous work of the authors, in this paper an SVM-based method is introduced. The numerical experiments reported show that our method is competitive against SVM and CART in terms of misclassification rates, and, at the same time, is able to detect critical values and variables interactions which are relevant for classification.

Keywords: Supervised Classification, Interactions, Support Vector Machines, Binarization

*This work has been partially supported by projects MTM2009-14039, ECO2008-05080 of MEC and FQM-329 of Junta de Andalucía, Spain.

1 Introduction

One of the most important tasks in Data Mining (Hand et al., 2001) is *Supervised Classification*, which seeks procedures for classifying objects in a set Ω into a set $\mathcal{C}$ of classes. Each object $u \in \Omega$ has associated a pair (c^u, x^u), where $c^u \in \mathcal{C}$ is the class membership of u, and x^u, the *predictor vector,* which takes values on a set X, usually assumed to be a subset of $\mathbb{R}^p$. Hereafter, we will simply use the term *variable* to refer to each component of the predictor vector. Not all the information about the objects in Ω is available: the class membership c^u is only known for those objects u in some subset $I \subset \Omega$, called the *training sample.* With this information, a classification rule, i.e., a function which yields a label $c \in \mathcal{C}$ for any predictor vector x, is sought.

The best studied case, which is the case analyzed in this paper, is the 2*-class problem*: $\mathcal{C} = \{-1, 1\}$. The general case of arbitrary finite $\mathcal{C}$ can be reduced to a series of 2-class problems, as has been suggested e.g. in Hastie and Tibshirani (1998), Herbrich (2002) and Vapnik (1998).

A popular and powerful tool for Supervised Classification is the so-called Support Vector Machine (SVM) (Cortes and Vapnik, 1995), which consists in finding the separating hyperplane with maximal margin, i.e., the one furthest from the closest object. This geometrical problem is expressed as a smooth convex problem with linear constraints, solved either in its primal or dual form.

The most popular and powerful versions of SVM embed, via a so-called kernel function, the original variables x_ℓ, $\ell = 1, 2, \ldots, p$, into a higher (possibly infinite) dimensional space (Herbrich, 2002). This way one obtains classifiers with good generalization properties, but it is usually hard to find out which variables are more relevant for the classification and how interactions between them affect the classifier. Many authors have attempted to overcome this weakness by proposing a two-step procedure: first, SVM is run, and then a rule, resembling the SVM-classifier but easier to interpret, is built. See e.g. Baesens et al. (2003); Barakat and Diederich (2006); Martens et al. (2007, 2009); Van Gestel et al. (2007). See also Mannino and Koushik (2000) and the references therein for other approaches to combine the good generalization behavior of SVMs with interpretability, trying to obtain some insights in the final decision model.

The approach proposed here is different, since a one-step SVM-based procedure is designed. In Carrizosa et al. (2009), the authors have introduced the so-called Binarized Support Vector

Machine (BSVM) method: each variable is replaced by a set of binary variables, defined by cutoffs of the original variables. The classification rules obtained this way have a classification behavior comparable to the standard linear SVM and clearly better than Classification and Regression Tress (CART). Moreover, BSVM gives a powerful tool for detecting which are the most relevant variables and critical values.

In this paper we introduce an improvement of the BSVM, the Non-linear Binarized Support Vector Machine (NBSVM), which is able to detect the relevant variables and the relevant interactions between variables. In this way we obtain a new classification procedure which is much more informative, in terms of detection of relevant variables and interactions, than benchmark procedures such as CART, and, as shown in the numerical tests, with competitive classification rates.

In essence, NBSVM is a standard SVM after transforming the variables into a very large set of binary features, generated via queries of type

$$\boxed{\begin{array}{c}\text{Is variable } \ell_1 \text{ greater/lower than or equal to } b_1,\\ \text{and variable } \ell_2 \text{ greater/lower than or equal to } b_2?\end{array}} \tag{1}$$

The remainder of the paper is organized as follows. In Section 2, some synthetic examples are presented, where interactions between the variables play an important role. It is seen in these examples that CARTs yield misleading information: variables which are irrelevant are presented as important ones, and values which do not play an important role at all are presented as critical values for classification. In Section 3, NBSVM is introduced. In Section 4, the construction of the classifier is reduced to solving a linear program, and a solution procedure is outlined. The numerical results are discussed in Section 5 and show that the proposed method is able to find the relevant variables in the synthetic examples presented, and, at the same time, yields good classification rates in publicly available and widely used datasets. Finally, conclusions and some lines for future research are discussed in Section 6.

2 Illustrating the relevance of interactions for classification

Since CART uses a greedy strategy to choose the splitting variable, it may fail to detect the most relevant variables in datasets in which the marginal distribution of each variable alone is not informative about class membership. Similarly, the BSVM proposed in Carrizosa et al. (2009) is not able to capture interactions between the variables since the classifier constructed this way is separable in the variables.

This important weakness of CART and BSVM is illustrated with three artificial datasets. The first two datasets, `cross` and `chess`, are taken from Robnik-Sikonja and Kononenko (2008) and contain exactly one relevant interaction each. The third dataset, `cube`, is introduced here for the first time and has several interactions influencing the class membership.

From a training set, two classification trees, referred to as default tree and tuned tree, are built. The default tree is the CART given by Matlab 6.5 (Matlab 6.5, 2002) with the default options, whereas the tuned tree is the best pruned CART, whose size is chosen by 10-fold crossvalidation (Kohavi, 1995). A BSVM classifier is built as well, using the default options, as proposed in Carrizosa et al. (2009).

2.1 `Cross` synthetic dataset

Dataset `cross` has six variables, out of which only x_1 and x_2 are relevant to the class. It is generated as follows: the vector (x_1, x_2) is uniformly generated on the cross $\{(x_1, x_2) \in [0,1]^2 : 0.4 \leq x_1 \leq 0.6 \text{ or } 0.4 \leq x_2 \leq 0.6\}$, whereas the remaining variables are independent and uniformly generated in the interval $[0,1]$.

The class only depends on the two first variables: objects satisfying $(x_1 - 0.5)(x_2 - 0.5) > 0$ belong to class 1, and to class -1 otherwise. The remaining variables x_ℓ, $\ell = 3, \ldots, 6$, are noise variables, and therefore useless for classification. Moreover, the value 0.5 is the only critical value for the two relevant variables.

We have considered a training set with $1,000$ objects. A graphical representation of the two first variables in the training set can be found in Figure 1. Objects from class 1 are represented as crosses and objects from class -1 as circles.

The default tree is partially shown in Figure 2, although the comments below refer to the whole tree. First, observe that noise variables x_3 and x_4 are used at the first and second levels of the tree, respectively. Second, even though no method will be able, in general, to spot exactly the critical values, CART gets again misleading information. Table 1 shows all cutoffs for the two meaningful variables, x_1 and x_2. Clearly, some of them are far away from the only critical value 0.5, where the most remarkably case is 0.0783, which, instead, is very close to the

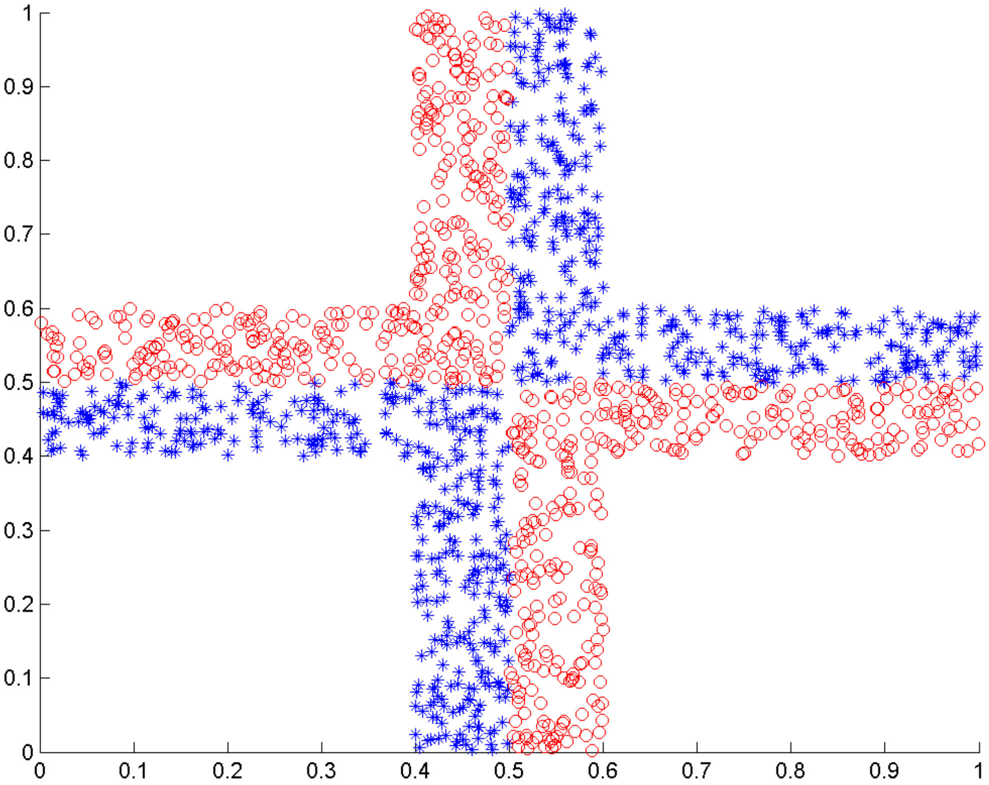

Figure 1: Representation of training set for `cross` dataset.

theoretical minimum 0. The tree default classified correctly 98.70% of an independent testing sample of size $1,000$.

The top part of the tuned tree can be found in Figure 3. As in Figure 2, noise variables, namely x_3 and x_4, are again used. All cutoffs provided by the tuned tree in the two meaningful variables are summarized in Table 2. Even though the extreme cutoff in the default tree has disappeared from the table, there are still two cutoffs being far from the critical value 0.5, namely 0.8066 and 0.3682. With this tuned tree, 98.90% of the independent testing set was correctly classified.

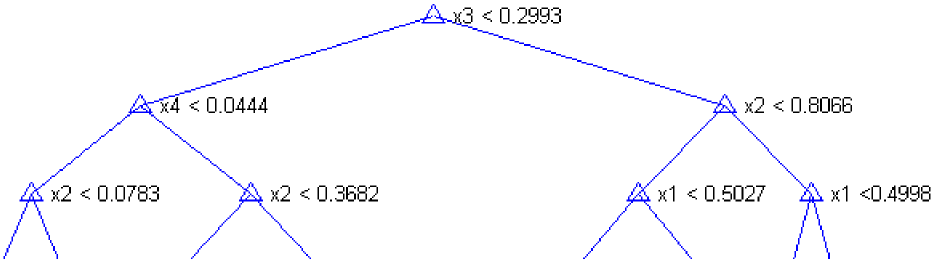

Figure 2: Default CART for `cross` dataset.

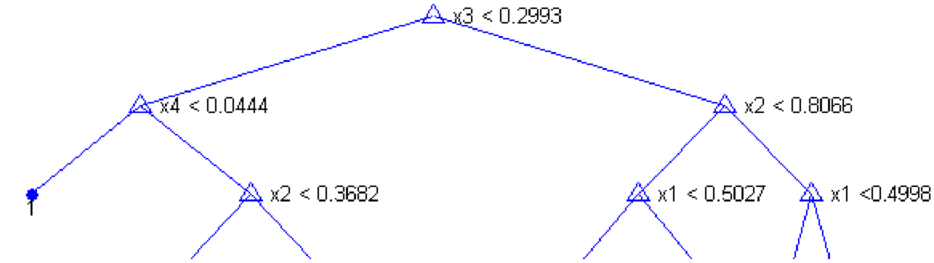

Figure 3: Tuned CART for `cross` dataset.

variable	cutoffs						
x_1	0.4914	0.4992	0.4998	0.5002	0.5027	0.5038	
x_2	0.0783	0.3682	0.4980	0.5001	0.5002	0.5010	0.8066

Table 1: Cutoffs of default CART for variables x_1 and x_2 and dataset `cross`

We have run the BSVM and obtained a simple classifier that assigns objects to class -1 if $x_1 < 0.5297$ and $x_2 \geq 0.4826$, and to class 1 otherwise. In this example it is evident that BSVM detects the two relevant variables, and the cutoffs are relatively close to the only critical

variable	cutoffs					
x_1	0.4914	0.4992	0.4998	0.5027	0.5038	
x_2	0.3682	0.4980	0.5001	0.5002	0.5010	0.8066

Table 2: Cutoffs of tuned CART for variables x_1 and x_2 and dataset `cross`

value 0.5. However, BSVM is outperformed by both CARTs in terms of classification accuracy in the testing set, yielding a classification rate of 68.80%. It is evident that BSVM is not able to capture that the interaction between the two relevant variables is important for the classification.

2.2 `Chess` synthetic dataset

Dataset `chess` has four variables. As in `cross`, the only two relevant variables are x_1 and x_2, which together with the class label yield a 4×4 chess board. In short, the vector of variables (x_1, x_2, x_3, x_4) is generated uniformly in the interval $[0,1]^4$. Then class 1 is assigned to the objects satisfying one of the following conditions:

- $x_1 \leq 0.25$ and $x_2 \leq 0.25$
- $0.5 \leq x_1 \leq 0.75$ and $x_2 \leq 0.25$
- $0.25 \leq x_1 \leq 0.5$ and $0.25 \leq x_2 \leq 0.5$
- $0.75 \leq x_1$ and $0.25 \leq x_2 \leq 0.5$
- $x_1 \leq 0.25$ and $0.5 \leq x_2 \leq 0.75$
- $0.5 \leq x_1 \leq 0.75$ and $0.5 \leq x_2 \leq 0.75$
- $0.25 \leq x_1 \leq 0.5$ and $0.75 \leq x_2$
- $0.75 \leq x_1$ and $0.75 \leq x_2$.

Class value -1 is assigned to the remaining objects.

By construction, the critical values for the relevant variables x_1 and x_2 are seen to be 0.25, 0.5 and 0.75.

We have considered a training set with $1,000$ objects, depicted in Figure 4.

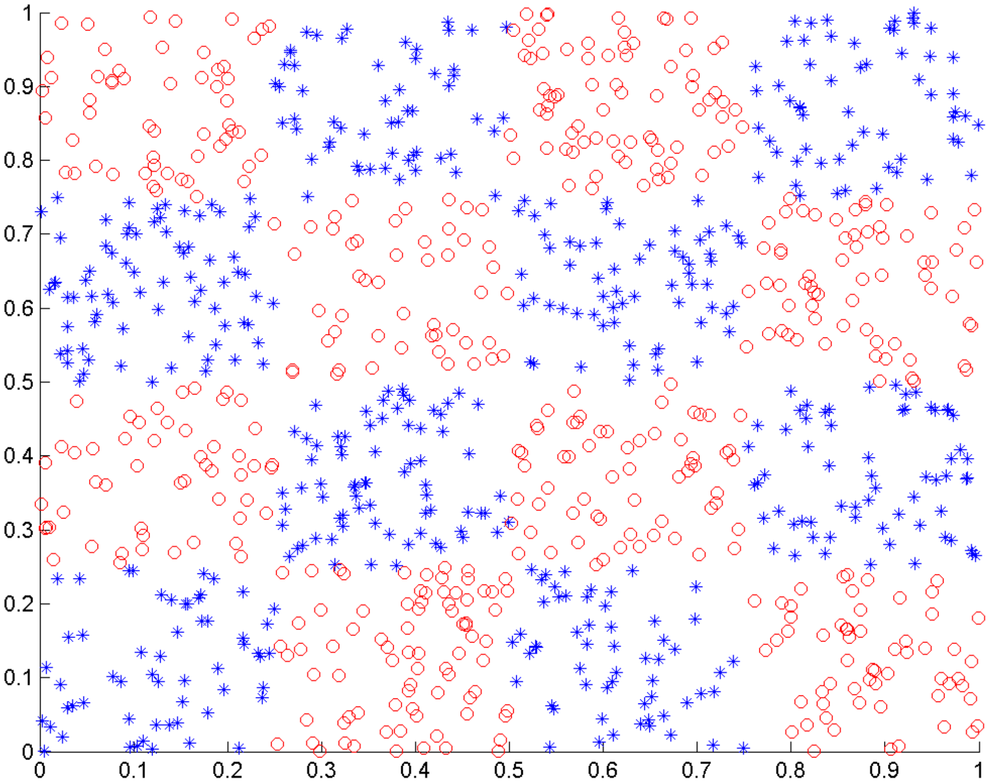

Figure 4: Representation of training set for `chess` dataset.

The three top levels of both the default tree and the tuned tree coincide and are depicted in Figure 5. The cutoffs used in the meaningful variables are given in Tables 3 and 4, respectively. In the default case, 49 out of the 62 nodes in the tree use one of the meaningful variables, where in the tuned tree, 42 out of 52 do (80% and 79%, respectively). In both trees, some of the cutoffs provided are far away from any of the critical values. As in `cross`, we have an extreme cutoff 0.9858, which is virtually equal to 1, the theoretical maximum. There are other examples of meaningless cutoffs. For instance, the cutoff in the root node 0.4018 is between critical points 0.25 and 0.5, and close to their middle point, therefore not being discriminative at all.

The correct classification rate for each tree in an independent testing sample of $1,000$ objects is 90.2% and 90.7% respectively.

BSVM does not give more encouraging results. Indeed, the classifier obtained has 92 features, 45 of which correspond to noise variables. The critical values are not properly identified. For instance, the feature with the highest weight has a cutoff equal to 0.4559, which again does not spot its closest critical value, 0.5. Moreover, the classification rate in the testing set is again discouraging, 50.60%.

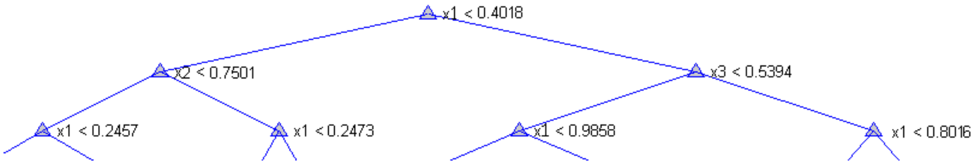

Figure 5: Default CART for `chess` dataset.

2.3 `Cube` synthetic dataset

In the third synthetic dataset, `cube`, several interactions between variables are relevant to the class membership.

It has six variables, whose values are generated uniformly in $[0,1]^6$. The class is determined by the three first variables as follows: Class value 1 is given to objects satisfying one of the two following conditions,

- $x_1 < 0.5$ and $x_2 > 0.5$ and $x_3 > 0.5$
- $x_1 > 0.5$ and $x_2 < 0.5$ and $x_3 < 0.5$.

variable	cutoffs							
x_1	0.2457	0.2473	0.2495	0.2511	0.2527	0.4018	0.4941	0.4974
	0.4986	0.5054	0.5058	0.5160	0.6726	0.7188	0.7427	0.7447
	0.7513	0.7526	0.7557	0.7569	0.7693	0.7726	0.8016	0.9858
x_2	0.0415	0.2430	0.2460	0.2484	0.2506	0.2549	0.2559	0.2578
	0.2628	0.4878	0.4961	0.4988	0.5003	0.5048	0.5073	0.5121
	0.5501	0.5796	0.7374	0.7381	0.7501	0.7505	0.7558	0.7567
	0.7898							

Table 3: Cutoffs of default CART for variables x_1 and x_2 and dataset `chess`

variable	cutoffs							
x_1	0.2457	0.2473	0.2527	0.4018	0.4941	0.4974	0.4986	0.5054
	0.5160	0.6726	0.7188	0.7427	0.7447	0.7513	0.7557	0.7569
	0.7726	0.8016	0.9858					
x_2	0.2430	0.2460	0.2484	0.2506	0.2549	0.2559	0.2578	0.2628
	0.4878	0.4961	0.4988	0.5003	0.5048	0.5073	0.5121	0.5501
	0.5796	0.7374	0.7501	0.7505	0.7558	0.7567	0.7898	

Table 4: Cutoffs of tuned CART for variables x_1 and x_2 and dataset `chess`

Class value -1 is assigned to the remaining objects.

Observe that the three relevant variables have exactly one critical value, with the same value, namely 0.5.

We have considered a training set with $1,000$ objects, depicted in Figure 6.

The top part of the default and the tuned tree are shown in Figures 7 and 8, respectively. First, observe that noise variable x_6 is used at the root of both trees. Second, meaningful variable x_1 is used in the second level, but the suggested critical value, 0.9076, is clearly misleading. Third, the default tree uses noise variables in 9 out of 49 nodes. The tuned tree uses noise variables in 5 out of 21 nodes. Finally, both CARTs yield many misleading cutoffs. This is particularly true for variable x_1 for which we have several cutoffs close to the theoretical maximum 1, where the most clear example is 0.9864.

The classification rate of CART in an independent testing set with 1,000 objects are 91.40%

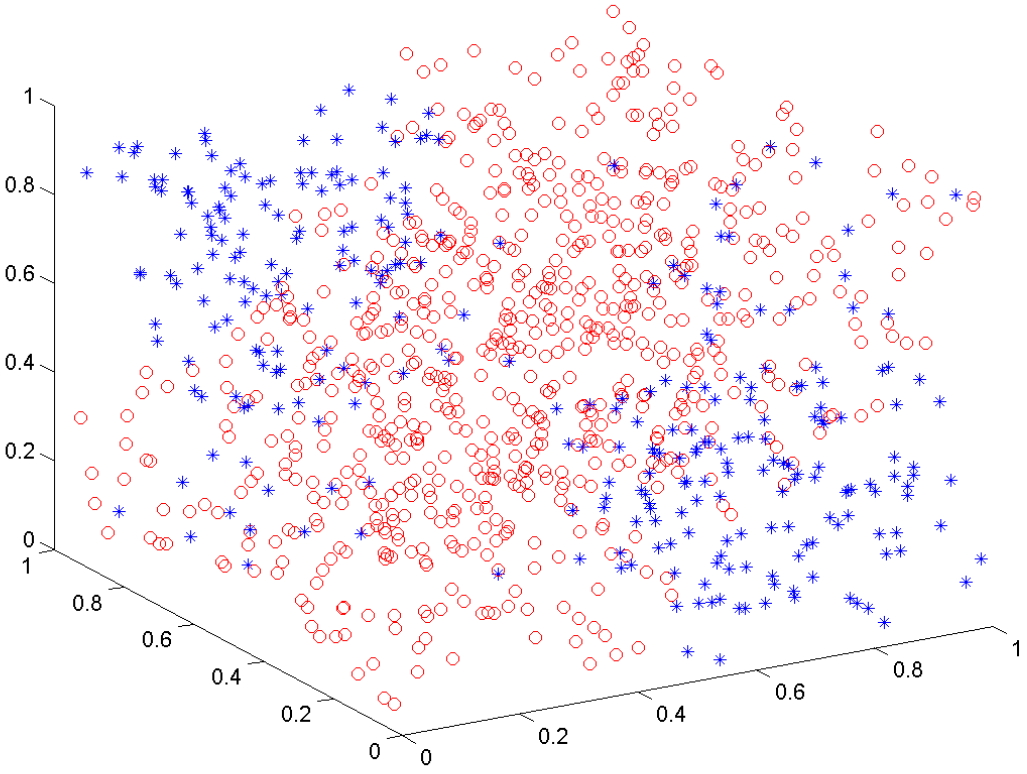

Figure 6: Representation of training set for `cube` dataset.

and 88.50%, respectively.

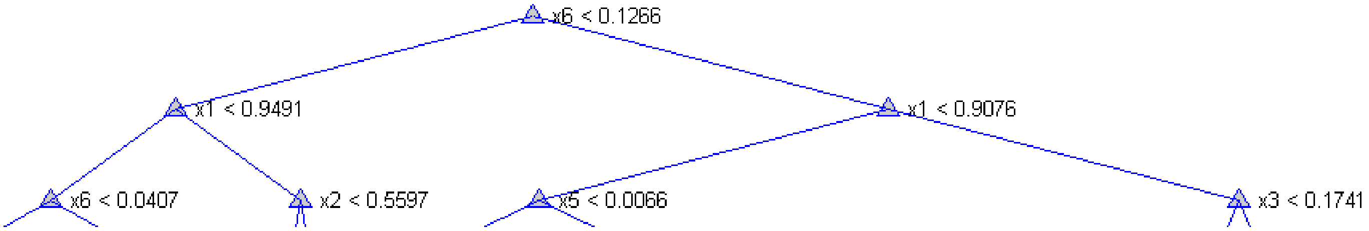

Figure 7: Default CART for `cube` dataset.

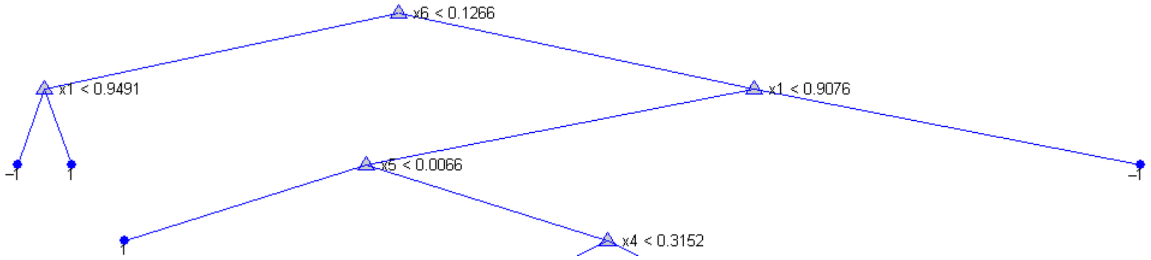

Figure 8: Tuned CART for `cube` dataset.

variable	cutoffs						
x_1	0.2857	0.3387	0.3949	0.4499	0.4807	0.4967	0.5000
	0.5007	0.7932	0.8660	0.9076	0.9491	0.9844	0.9864
x_2	0.0614	0.4144	0.4734	0.4934	0.4934	0.4945	0.4994
	0.5016	0.5045	0.5049	0.5229	0.5450	0.5597	0.8552
x_3	0.1150	0.1741	0.4793	0.4902	0.4975	0.5016	0.5139
	0.5262	0.5510	0.6000	0.6008	0.9130		

Table 5: Cutoffs of default CART for variables x_1, x_2 and x_3 and dataset `cube`

We have run the BSVM and obtained the trivial classifier that classifies all the objects in the same class -1. Hence, BSVM is not able to capture any interaction between the variables. Moreover, correct classification rate in the independent testing set is 74.70%, which coincides with the percentage of objects in the most common class.

In other words, refinements of the BSVM are needed to properly detect interactions between variables.

variable	cutoffs						
x_1	0.3949	0.4967	0.5000	0.5007	0.7932	0.9076	0.9491
x_2	0.4934	0.4945	0.5045	0.5049			
x_3	0.4975	0.5016	0.5139	0.5262	0.6008		

Table 6: Cutoffs of tuned CART for variables x_1, x_2 and x_3 and dataset `cube`

3 Binarizing the variables and their interactions

The set of features proposed in Carrizosa et al. (2009) is extended here. For $\ell = 1, 2, \ldots, p$, let B_ℓ equal to the finite set of middle points between two consecutive values from $\{x_\ell^u : u \in I\}$. Moreover, for $b \in B_\ell$, define

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

Analogously, define $\phi_{\ell b}^{\leq}$ replacing $x_\ell \geq b$ by $x_\ell \leq b$.

Let

$$\mathcal{F}^{\geq} = \{\phi_{lb}^{\geq} : \ell = 1, \ldots, p,\, b \in B_\ell\},$$

and define $\mathcal{F}^{\leq}$ analogously.

In Carrizosa et al. (2009) classifiers consisting of linear combinations of features in $\mathcal{F}^{\geq}$ (or, equivalently, in $\mathcal{F}^{\leq}$) are considered. Such features are meant to detect important variables and their critical values, but not the interaction between them. In this paper, we extend the set of features to

$$\mathcal{F} = \left\{\phi \cdot \tilde{\phi} : \phi, \tilde{\phi} \in \mathcal{F}^{\geq} \cup \mathcal{F}^{\leq}\right\}.$$

Since clearly $\phi \cdot \phi = \phi$, for all $\phi \in \mathcal{F}^{\geq} \cup \mathcal{F}^{\leq}$, we have that $\mathcal{F} \supseteq \mathcal{F}^{\geq}$. We consider classifiers constructed via a score function f of the form

$$f(x) = \sum_{\phi \in \mathcal{F}} \omega_\phi \phi(x) + \beta = \omega^\top \Phi(x) + \beta, \tag{3}$$

yielding the following classifier: objects are assigned to class 1 if $f(x) > 0$, and they are assigned to class -1 otherwise.

Note that for each feature ϕ, the absolute value of coefficient ω_ϕ indicates the weight ϕ has in the score function, and thus it can be seen as a measure of the importance of such feature for the classification. Hence, these weights will be helpful to detect the importance of variables and interactions between.

4 Building the classifier

In order to choose ω and β in (3) we follow a soft-margin SVM-based approach (Cortes and Vapnik, 1995), which consists in finding the hyperplane which maximizes the margin in the feature space, but allowing some objects to be misclassified.

The remainder of this section is as follows: We first derive the Mathematical Programming formulation for the problem of maximizing the soft margin, which will be solved by a Column Generation approach. In Section 4.2, we describe in detail how we tackle the pricing problem, devoted to generate new features. In Section 4.3, we summarize the entire column generation procedure and give some implementation details.

4.1 Soft-margin formulation and column generation

With the motivation given in Carrizosa et al. (2009) for the BSVM, we formulate the NBSVM as the following Linear Programming (LP) problem,

$$\begin{array}{lrl} \min & \sum_{\phi\in\mathcal{F}}(\omega_\phi^+ + \omega_\phi^-) + C\sum_{u\in I}\xi^u & \\ s.t.: & \sum_{\phi\in\mathcal{F}}(\omega_\phi^+ - \omega_\phi^-)c^u\phi(x^u) + \beta c^u + \xi^u \geq 1 & \forall u \in I \\ & \omega_\phi^+ \geq 0 & \phi \in \mathcal{F} \\ & \omega_\phi^- \geq 0 & \phi \in \mathcal{F} \\ & \xi^u \geq 0 & \forall u \in I \\ & \beta \in I\!R, & \end{array} \tag{4}$$

with dual

$$\begin{array}{lrl} \max & \sum_{u\in I}\lambda_u & \\ s.t.: & -1 \leq \sum_{u\in I}\lambda_u c^u\phi(x^u) \leq 1 & \forall \phi \in \mathcal{F} \\ & \sum_{u\in I}\lambda_u c^u = 0 & \\ & 0 \leq \lambda_u \leq C & \forall u \in I. \end{array} \tag{5}$$

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

As in Carrizosa et al. (2009), the well-known Mathematical Programming tool called Column Generation (Gilmore and Gomory, 1961) is proposed to solve Problem (4). This is an iterative procedure in which Problem (4) is solved for a reduced set of columns (in our case features), $F \subset \mathcal{F}$, yielding problems analogous to Problem (4) and its dual (5), but where $\mathcal{F}$ is replaced by its subset F. We denote such problems as Problem (4-F) and (5-F), respectively.

A check to the reduced problem is performed to see whether the current solution is optimal for Problem (4) or not. In the latter case, we generate new features, with the aim of improving the objective value of the current solution. The generated features, obtained from the information given by the dual problem (5-F), are added to F, and then Problem (4-F) is solved again. This process is repeated until no other promising feature is found.

Let $(\omega^*, \beta^*, \xi^*)$ be an optimal solution of the reduced problem (4-F), $(\lambda_u^*)_{u \in I}$ be the values of the corresponding optimal dual solution, and define $\Gamma(\phi)$ as

$$\Gamma(\phi) = \sum_{u \in I} \lambda_u^* c^u \phi(x^u). \tag{6}$$

If $(\omega^*, \beta^*, \xi^*)$ is not optimal for Problem (4), then the corresponding dual solution will be infeasible for Problem (5), i.e., the dual constraint

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

will be violated for at least one feature in $\mathcal{F} \setminus F$. Therefore, promising features can be found by solving the two following pricing problems:

$$\max_{\phi \in \mathcal{F}} \Gamma(\phi), \tag{8}$$

$$\min_{\phi \in \mathcal{F}} \Gamma(\phi). \tag{9}$$

Features found this way and violating dual constraint (7), i.e. satisfying $|\Gamma(\phi)| > 1$, will be added to set F.

4.2 Generation of features

For the particular case in which just features of degree one are considered, $\phi \in \mathcal{F}^{\geq}$, an exact algorithm to solve the pricing problem is proposed in Carrizosa et al. (2009). In this algorithm, the candidates to cutoffs are sorted with respect to their value for variable ℓ, and the optimal

class	x_1	x_2
c^1	5	8
c^2	9	3
c^3	6	6
c^4	3	10

Table 7: Simple example of database.

$\lambda_1 c^1 + \lambda_2 c^2 + \lambda_3 c^3 + \lambda_4 c^4$	$\lambda_1 c^1 + \lambda_2 c^2 + \lambda_3 c^3$	$\lambda_2 c^2 + \lambda_3 c^3$	$\lambda_2 c^2$
$\lambda_1 c^1 + \lambda_3 c^3 + \lambda_4 c^4$	$\lambda_1 c^1 + \lambda_3 c^3$	$\lambda_3 c^3$	-
$\lambda_1 c^1 + \lambda_4 c^4$	$\lambda_1 c^1$	-	-
$\lambda_4 c^4$	-	-	-

Table 8: Value of Γ for features in $\mathcal{F}$.

value of the pricing problem is found by inspecting the ordered list, for each $\ell = 1, \ldots, p$. We show now how to generate features of degree 2, i.e., $\phi \in \mathcal{F}$.

Recall that features in $\mathcal{F}$ are defined by six parameters: two variables (ℓ_1, ℓ_2), their corresponding relations ($\geq$ or $\leq$) and their corresponding cutoffs (b_1, b_2). We propose a heuristic procedure to generate new features, where the variables and relations are chosen randomly, while their cutoffs are sought using a local search procedure. Hereafter, and without loss of generality, we will assume that the variables are fixed and their relations are $\geq$. The following example gives some insights on the design of our local search procedure.

Example 1 *In Table 7, a simple dataset with two variables and four objects is presented. For variable x_1, the candidates to cutoffs are $4, 5.5$ and 7.5, while the ones for variable x_2 are $4.5, 7$ and 9.*

In Table 8, the values of Γ in (6) for all features in $\mathcal{F}$, i.e., for different cutoffs for variable x_1 (columns) and different cutoffs for variable x_2 (rows) are shown. (For ease of notation, we have dropped the symbol * from λ.) By adding the trivial cutoff 3 for both variables, their minimum, we also include all features of degree 1, found in the first column and the first row of the table.

When we move along the positions in the table, right-to-left and down-to-up movements *lead to either* Γ *gaining one term of the form* $\lambda_u c^u$ *or* Γ *remaining unchanged. This suggests a heuristic procedure to find a promising feature to be added to* F*.*

Our local search procedure to generate a new feature is iterative and has the aim of improving the current cutoffs, one at a time. Suppose the feature at hand is $\phi = \phi^{\geq}_{\ell_1,b_1} \cdot \phi^{\geq}_{\ell_2,b_2}$. In Example 1, $\Gamma(\phi)$ corresponds to a position in Table 8. Suppose we choose to improve the cutoff associated with variable ℓ_1. Once the variable has been chosen, the objects in I can be sorted increasingly by their values in variable ℓ_1. Let $u(i)$ the object in the i-th position. (As said before, and in order to consider all features of degree 1, we add the trivial cutoff $x^{u(1)}_{\ell_1}$ to the list of candidates to cutoffs. This is done by simply considering a dummy object $u(0)$ with $x^{u(0)}_{\ell_1} = x^{u(1)}_{\ell_1}$.) We want to replace the current cutoff $b_1 = \frac{x^{u(i)}_{\ell_1} + x^{u(i-1)}_{\ell_1}}{2}$ in order to create a different feature $\hat{\phi} = \phi^{\geq}_{\ell_1,\hat{b}_1} \cdot \phi^{\geq}_{\ell_2,b_2}$ with a better Γ. When changing b_1 to $\hat{b}_1$, with $\hat{b}_1 = \frac{x^{u(j)}_{\ell} + x^{u(j-1)}_{\ell}}{2}$, the values of ϕ change only in the objects $u(k)$ with $i < k \leq j$ if $i < j$ (and, respectively $j < k \leq i$ if $j < i$). Taking this into account, we know that, when moving backward, i.e. $j < i$, then the change in Γ is easily computed using that

$$\Gamma(\hat{\phi}) = \Gamma(\phi) - \sum_{k:\, \phi(x^{u(k)})=1,\, j<k\leq i} \lambda^*_{u(k)} c^{u(k)}.$$

When moving forward, i.e. $j > i$, then, in order to know if the term $\lambda^*_{u(k)} c^{u(k)}$ must be added, we need to check, for all objects $u(k)$ with $i < k \leq j$, whether $x^{u(k)}_{\ell_1}$ is greater than or equal to the cutoff $\hat{b}_1$. If the new feature improves the existing Γ, we update the cutoff. The remaining types of features can be analyzed in a similar way.

All the above comments are taken into account to implement a local search heuristic that, in every step, moves forward or backward in the ordered set of candidates to cutoffs for a randomly chosen variable. The algorithm stops when T movements are performed without any improvement in the value of Γ.

In the following we give the algorithm corresponding to maximizing Γ.

Algorithm c-NBSVM

Step 0. Initialization:

- Sort objects in I in increasing order with respect to x_{ℓ_k} and add the dummy object, $\forall k = 1, 2$.

- $i_k \leftarrow 1$, $\forall k = 1, 2$.
- $b_k \leftarrow x_{\ell_k}^{u_{\ell_k}(1)}$, $\forall k = 1, 2$.
- `steps` $\leftarrow 0$ and `max`$\leftarrow 0$.

Step 1. Randomly choose a variable $\ell \in \{\ell_1, \ell_2\}$.

Step 2. Randomly choose a type of movement: `forward` or `backward`.

Step 3.
- If `forward`, then randomly choose $h \in \{i_k + 1, \dots, \sharp(I)\}$.
- If `backward`, then randomly choose $h \in \{1, 2, \dots, i_k - 1\}$.

Step 4. Let $b = \frac{x_\ell^{u_\ell(h)} + x_\ell^{u_\ell(h-1)}}{2}$ and compute $\Gamma(\phi)$, where b is the new cutoff for x_ℓ in feature ϕ.

Step 5. If $\Gamma(\phi) >$ `max` then $b_\ell \leftarrow b$, `max`$\leftarrow \Gamma(\phi)$ and `steps` $\leftarrow 0$. Otherwise, `steps` $\leftarrow$ `steps`$+1$.

Step 6. If `steps` $< T$, then go to Step 1. Otherwise, stop.

Notice that in the initialization step, both cutoffs are chosen equal to the trivial one, i.e., the minimum in each variable. For this feature,

$$\Gamma(\phi) = \sum_{u \in I} \lambda_u^* c^u = 0,$$

because of the second constraint in the dual formulation (5), which coincides with the amount to which `max` is initialized. If at the end of the algorithm one of the two cutoffs is equal to the trivial one, we can rewrite the feature as one of degree 1. Thus, the algorithm always generates features with cutoffs in B_ℓ.

An analogous algorithm for minimizing Γ can be developed replacing `max` by `min` and changing Step 5 in the obvious way.

4.3 Our column generation implementation

Our implementation of the column generation algorithm is as follows:

Column Generation Algorithm

```
Step 0. Run Algorithm CG-BSVM (Carrizosa et al., 2009) to build an initial set $F$ with features of degree one.
Step 1. Repeat $Q$ times:
    Step 1.1. Randomly choose a pair of variables $\ell_1, \ell_2$ and their corresponding relations. Select $b_1, b_2$ by maximizing $\Gamma$ using **Algorithm c-NBSVM**.
    Step 1.2. If $\Gamma(\phi) > 1$, then $F \leftarrow F \cup \{\phi\}$.
Step 2. Repeat $Q$ times:
    Step 2.1. Randomly choose a pair of variables $\ell_1, \ell_2$ and their corresponding relations. Select $b_1, b_2$ by minimizing $\Gamma$ using **Algorithm c-NBSVM**.
    Step 2.2. If $\Gamma(\phi) < -1$, then $F \leftarrow F \cup \{\phi\}$.
Step 3. If $F$ has not been modified in either Step 1 or Step 2, then STOP: we have found a solution to Problem (4). Otherwise, solve Problem (4-$F$) and go to Step 1.
```

We have chosen our initial set of features to be equal to the set of features generated by the BSVM. Other choices for the initial set of features are possible. For instance, a plausible one would be to generate one feature $\phi^{\geq}_{\ell b}$ per variable ℓ, with b equal to the median of variable x_ℓ in the objects of I.

5 Numerical results

Two different types of experiments have been considered to illustrate the advantages of NBSVM against benchmark procedures.

We first illustrate that indeed NBSVM is able to capture the importance of interactions between relevant variables and detect their critical values. For this purpose, we use the synthetic datasets introduced in Section 2. The results are presented in Section 5.1.

Another fundamental issue of any classifier is its classification ability. In Section 5.2, six datasets from UCI Machine Learning Repository (Newman et al., 1998) are used to compare the classification rates of NBSVM with CART, standard SVM and BSVM.

In all the experiments we have used CPLEX 8.1.0 (CPLEX Reference Manual, 2002) as the LP solver, and the values of T and Q have been fixed to 100 and 25, respectively. Preliminary experiments suggest these values of T and Q yield good results.

5.1 Detecting interactions and critical values on synthetic datasets

In Section 2 three synthetic datasets were described. The behavior of CART and BSVM in these datasets was analyzed. It was shown that, in those examples, none of the methods considered was able to detect the relevant variables and the critical values. In this section, the behavior of NBSVM in these three datasets is studied, showing that, when interactions play an important role in the class membership, NBSVM is able to detect the important variables and their critical values.

As training and testing sets, those in Section 2 have been taken. The tradeoff parameter C is chosen by 10-fold crossvalidation among the values 2^i, for $i = -12, \dots, 12$.

5.1.1 `Cross` synthetic dataset

After running NBSVM in the dataset `cross`, the score function is:

$$f(x) = -1 + 2\phi^{\geq}_{1,0.4998} \cdot \phi^{\geq}_{2,0.5001} + 2\phi^{\leq}_{1,0.5005} \cdot \phi^{\leq}_{2,0.5013}.$$

Several observations are worth to remark. First, none of the four noise variables are used by the classifier. Second, the classifier is based on a very simple score function that uses only two features from $\mathcal{F}$. Moreover, such features reflect the interaction between variables x_1 and x_2. Finally, the critical values for classification are successfully found. From the description of the dataset, it is clear that a good classification can be expected. Indeed, 99.80% of the objects in the testing set are correctly classified.

5.1.2 `Chess` synthetic dataset

In the dataset `chess`, NBSVM behaves very differently. The obtained classifier has 568 features, 116 of which use at least one noise variable. At first sight, crude NBSVM is unable to attain the goal of identifying relevant variables and critical values. However, a closer analysis reveals that valuable information is provided by the classifier. Indeed, if, as done in wrapping procedures for feature selection (Guyon and Elisseff, 2003), all features with weight close to zero are discarded, the picture is quite different, since the 331 features with highest weight (in absolute value) use both relevant variables, being below 10^{-3} the weight of the first feature that uses a noise variable. All the weights of features with noise variables sum up to only 0.0058 in absolute value, out of 36, the total for all the features. Hence, we can conclude that even after extracting many features, the interaction between the relevant variables is well detected.

Table 9 shows the cutoffs of both relevant variables in the five features with the highest weight. They all are rather close to the exact critical values, namely 0.25, 0.5 or 0.75. The NBSVM classifier yields a correct classification rate of 97.50%, outperforming the 90.7% obtained with CART in Section 2.2.

variable	cutoffs				
x_1	0.2682	0.2503	0.5027	0.7512	0.2498
x_2	0.4969	0.2457	0.7458	0.7501	0.7464

Table 9: Cutoffs of NBSVM for variables x_1 and x_2 and dataset `chess`.

Interpreting just the features with highest weight might be unfair without information about classification ability of such features by themselves. For the dataset `chess`, additional experiments have been done to address this issue. We fix the tradeoff parameter C to the one used in the experiment above, i.e. $C = 4$. A backward elimination wrapping procedure is used (Guyon and Elisseff, 2003) fixing to m the maximum number of features to be allowed. First, NBSVM is run in the training set, then the features with lowest weight are taken off, until a set of m features or less remains. The weights of these features are recomputed solving Problem (4). Not all the features are used after weights are recomputed. For example, with $m = 50$, only 30 features are used, and the correct classification rate is 97.90%. When reducing even further, $m = 20$, 17 features are used, and the classification rate remains almost the same: 97.80%.

5.1.3 `Cube` synthetic dataset

As described in Section 2.3, the interactions in dataset `cube` are more complex and affect several variables. After running NBSVM in the dataset `cube`, the score function is:

$$f(x) = -1 + 2\phi^{\leq}_{1,0.4974} \cdot \phi^{\geq}_{2,0.5034} + 2\phi^{\leq}_{2,0.4969} \cdot \phi^{\leq}_{3,0.4994} - 2\phi^{\leq}_{3,0.5013} \cdot \phi^{\leq}_{1,0.4983}.$$

As in the case of `cross`, several facts are remarkable: First, none of the four noise variables are used by the classifier. Second, the classifier is based on a very simple score function that uses only three features from $\mathcal{F}$. Third, the critical values for classification in the interactions between relevant variables x_1, x_2 and x_3 are successfully found. Finally, the classification rate in the independent testing set is 99.80%, which outperforms the 92.80% obtained by CART. Observe that, although the interactions in this data set involve three variables, our algorithm, by considering pairwise interactions, is able to detect critical values and relevant interactions.

5.2 Classification ability on real-life datasets

Several experiments have been performed with real-life datasets from the UCI Machine Learning Repository (Newman et al., 1998). The proposed procedure, NBSVM, is compared with benchmark procedures such as CART (Breiman et al., 1984), (linear) BSVM (Carrizosa et al., 2009), and SVM (Cortes and Vapnik, 1995) with three popular choices of the kernel, namely, the linear kernel (linear), polynomial of degree 2 (deg 2) and radial basis function (rbf).

We follow Rätsch et al. (2001) for the experiment setting: 80% of the objects in the dataset were chosen randomly to train the classifier. The remaining 20% objects were used for testing. The whole procedure is repeated 30 times. The tradeoff parameter C in SVM, BSVM and NBSVM is chosen by 10-fold crossvalidation in the training set, among the values 2^i, for $i = -12, \dots, 12$. Table 10 shows the averages and standard deviations of the percentages of correctly classified objects in the testing set. The box-and-whisker plots are shown in Figure 9.

dataset	SVMs			CART		interpretable SVMs	
	linear	deg 2	rbf	default	tuned	BSVM	NBSVM
`credit`	86.21(2.58)	87.23(2.61)	85.09(2.05)	83.31(2.80)	86.26(2.84)	87.10(2.56)	87.18(2.70)
`pima`	77.16(3.18)	77.47(2.88)	77.60(2.93)	70.54(3.36)	73.72(3.93)	74.74(2.98)	75.58(3.32)
`housing`	85.59(3.60)	87.03(2.71)	87.58(3.21)	83.04(3.28)	82.91(3.89)	85.69(3.82)	87.29(2.75)
`bupa`	67.44(4.32)	72.37(4.86)	72.13(4.78)	63.57(5.17)	64.64(4.95)	71.98(5.11)	72.90(6.79)
`wdbc`	97.60(1.44)	97.69(1.29)	97.87(1.36)	91.90(2.50)	91.17(2.83)	96.67(1.75)	96.81(1.82)
`cleveland`	81.39(4.57)	80.94(4.95)	80.61(4.14)	73.22(4.93)	75.78(5.62)	80.61(6.60)	81.17(5.33)

Table 10: Classification results in six datasets from the UCI Machine Learning Repository.

From the results we can see that classification performance obtained by NBSVM is not worse than the one obtained with SVM using the rbf kernel.

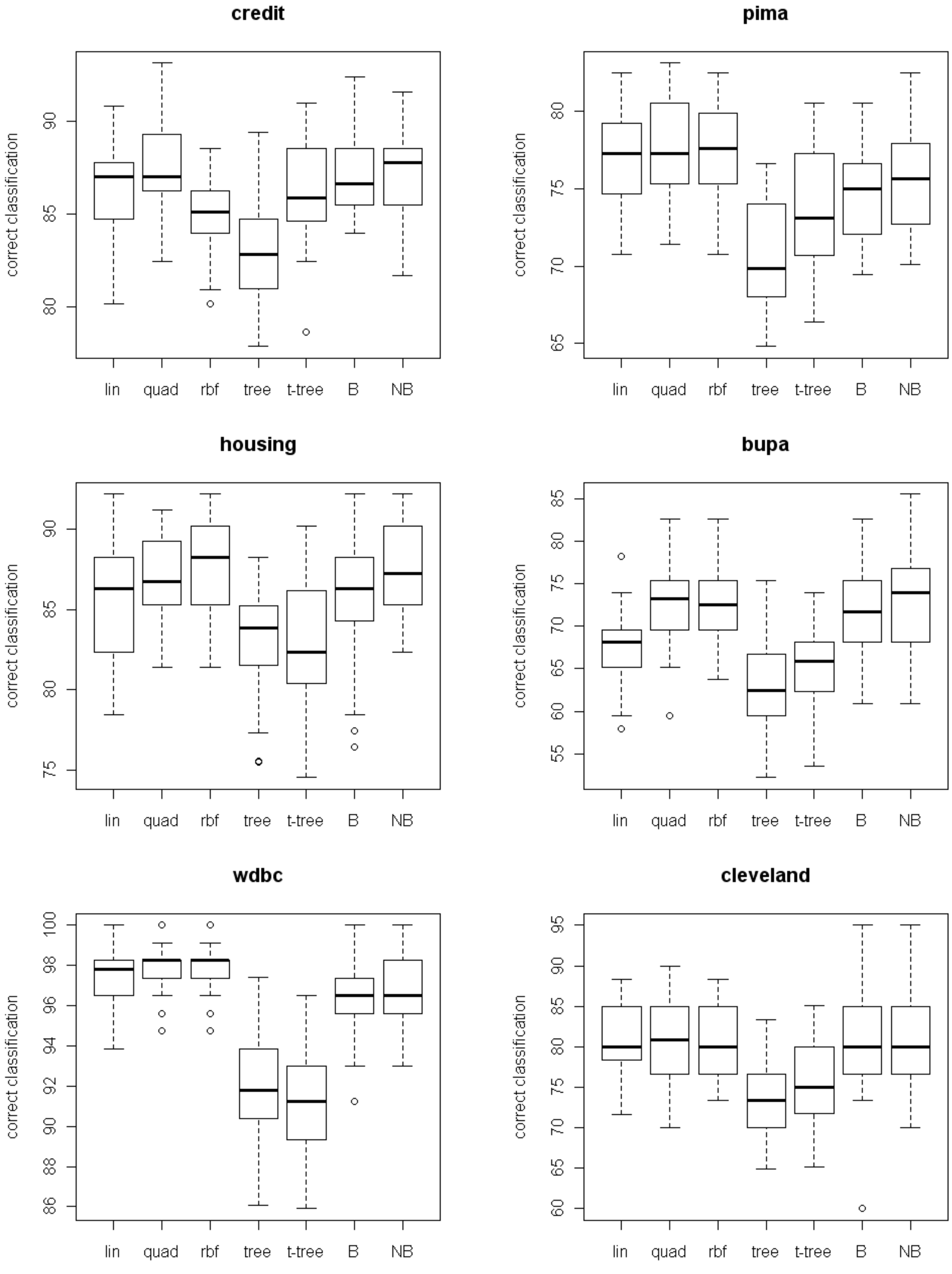

Figure 9: Classification results using linear kernel (lin), quadratic kernel (quad) and rbf kernel (rbf), default tree (tree), tuned tree (t-tree), BSVM (B) and NBSVM (NB)

In housing and bupa, the rbf kernel behaves clearly better than the linear kernel. The reason might be that some interactions between variables are important for classification. This fact is highlighted by a better classification performance of NBSVM, which uses interactions between variables, compared with BSVM, which only uses linear combination of the binarized variables.

Comparing NBSVM with CART, which also yields easy-to-interpret classifiers, we can see that the classification rates of NBSVM are clearly better than those of CART.

6 Conclusions and further research

In this paper an extension of BSVM for supervised classification, the NBSVM, has been proposed. The classification rule is constructed via a binarization of the original variables, since each variable is transformed into a large series of binary variables corresponding to queries of type (1).

Our numerical experiments show that, in situations where interactions are important for classification, NBSVM is able to find the relevant variables and critical values, whereas other methods give misleading information. Moreover, NBSVM provides this useful information without deterioration of the classification ability. Indeed, NBSVM classification ability is comparable to classical SVM with the powerful rbf kernel.

In other words, NBSVM is a very competitive classification tool when one seeks good classification rates as well as information on the critical values of variables and interactions of variables most relevant for classification.

Set $\mathcal{F}$ contains features of degree up to two, allowing us to model pairwise interactions. The very same approach can be used when set $\mathcal{F}$ is enlarged to include features of degree up to g, thus modeling interactions between groups of at most g variables. Clearly, these interactions will only make sense for small values of g, which makes our approach still computationally affordable. A first attempt to address the problem for values of g larger than two has been given in Martín-Barragán (2006). The design of more efficient heuristics deserves further study.

As in Carrizosa et al. (2009), the norm used is the ℓ_1, thus constructing the classifier amounts to solving a large-dimensional linear program. A column-generation procedure has been suggested as heuristic solution procedure. How to use more sophisticated heuristics and other norms (such as the usual Euclidean distance) deserves further study.

References

Baesens, B., R. Setiono, C. Mues, J. Vanthienen. 2003. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science* **49** 312–329.

Barakat, N., J. Diederich. 2006. Eclectic rule-extraction from support vector machines. *International Journal of Computational Intelligence* **2**(1) 59–62.

Breiman, L., J.H. Friedmann, R.A. Olshen, C.J. Stone. 1984. *Classification and Regression Trees*. Wadsworth, Belmont, CA.

Carrizosa, E., B. Martín-Barragán, D. Romero Morales. 2009. Binarized support vector machines. *INFORMS Journal on Computing* Published online in Articles in Advance, April 2009.

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

CPLEX Reference Manual. 2002. *ILOG CPLEX 8.0*. ILOG S.A., Gentilly.

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

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

Hand, D.J., H. Mannila, P. Smyth. 2001. *Principles of Data Mining*. MIT Press.

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

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

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

Mannino, M.V., M.V. Koushik. 2000. The cost-minimizing inverse classification problem: a genetic algorithm approach. *Decision Support Systems* **29**(3) 283–300.

Martens, D., B. Baesens, T. Van Gestel. 2009. Decompositional rule extraction from support vector machines by active learning. *IEEE Transactions on Knowledge and Data Engineering* **21**(2) 178–191.

Martens, D., B. Baesens, T. Van Gestel, J. Vanthienen. 2007. Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research* **183**(3) 1466–1476.

Martín-Barragán, B. 2006. Mathematical programming for support vector machines. Ph.D. thesis, Universidad de Sevilla.

Matlab 6.5. 2002. Software available at `http://www.mathworks.com/`.

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

Rätsch, G., T. Onoda, K.R. Müller. 2001. Soft margins for adaboost. *Machine Learning* **42**(3) 287–320.

Robnik-Sikonja, M., I. Kononenko. 2008. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering* **20**(5) 589–600.

Van Gestel, T., D. Martens, B. Baesens, D. Feremans, J. Huysmans, J. Vanthienen. 2007. Forecasting and analyzing insurance companies' ratings. *International Journal of Forecasting* **23**(3) 513–529.

Vapnik, V. 1998. *Statistical Learning Theory*. Wiley.