Machine Learning
https://doi.org/10.1007/s10994-023-06366-1

Multiclass optimal classification trees with SVM-splits

Víctor Blanco[1] · Alberto Japón[2] · Justo Puerto[2]

Received: 22 November 2021 / Revised: 9 March 2023 / Accepted: 29 June 2023

Abstract
In this paper we present a novel mathematical optimization-based methodology to construct tree-shaped classification rules for multiclass instances. Our approach consists of building Classification Trees in which, except for the leaf nodes, the labels are temporarily left out and grouped into two classes by means of a SVM separating hyperplane. We provide a Mixed Integer Non Linear Programming formulation for the problem and report the results of an extended battery of computational experiments to assess the performance of our proposal with respect to other benchmarking classification methods.

Keywords Supervised classification · Optimal classification trees · Support vector machines · Multiclass

Mathematics Subject Classification 90C27 · 62H30 · 90C90

1 Introduction

Interpretability is a crucial requisite demanded to machine learning methods provoked by the tremendous amount of methodologies that have arised in the last decade (Du et al., 2019). It is expected that the model that results when applying a machine learning methodology using a training sample, apart from being able to adequately predict the behaviour of out-of-sample observations, can be interpreted. Different tools have been applied to derive *interpretable* machine learning methods. One of the most popular strategies to simplify the obtained models is *feature selection*, in which a reduced set of attributes is to be

Editor: Sicco Verwer.

✉ Alberto Japón
ajapon1@us.es

Víctor Blanco
vblanco@ugr.es

Justo Puerto
puerto@us.es

1 Institute of Mathematics (IMAG), Universidad de Granada, Granada, Spain

2 Institute of Mathematics (IMUS), Universidad de Sevilla, Sevilla, Spain

Published online: 20 September 2023

selected without loosing quality in the predictions. Reducing the number of parameters to analyze, the models can be easier to understand, yielding higher descriptive accuracy. One could also consider models that can be modulated, in the sense that a great proportion of its prediction-making process can be interpreted independently. This is the case of generalized linear models (Hastie & Tibshirani, 2017). Other methods incorporate interpretability as a synonym of being able to be reproduced by humans in its entire construction (Breiman et al., 1984; Letham et al., 2015). This is the case of Decision Trees with small depth which can be visualized and interpreted easily by users even not familiar with the tools behind their construction. We adopt a tree-based methodology through this paper.

Among the wide variety of strands derived under the lens of Machine Learning, classification is one that has attracted a lot of attention because of its applicability in many different fields (Bahlmann et al., 2002; Harris, 2013; Kašćelan et al., 2016; Majid et al., 2014; Radhimeenakshi, 2016). Classification methodologies aim to adequately predict the class of new observations provided that a given sample has been used to construct the *classification rule*. The role of Mathematical Programming in the construction of classification models has been widely recognized, and some of the most popular methods to derive classification rules are based on solving optimization problems (see, Blanco et al., 2020b; Cortes & Vapnik, 1995; Carrizosa et al., 2021; Blanco et al., 2020a; Günlük et al.,2018, among many others). Moreover, Mathematical Programming has also been proven to be a flexible and accurate tool when requiring interpretability to the obtained models (see, e.g. Baldomero-Naranjo et al., 2020; 2021; Blanco et al.,2022b; Gaudioso et al., 2017).

However, most of the optimization tools derived to construct classifiers assume instances with only two classes. In this paper, we provide a novel classification method in which the instances are allowed to be classified into two or more classes. The method is constructed using one of the most interpretable classification method, Classification Trees, but combined with Support Vector Machines, which provide highly predictive models.

We have developed a Mathematical Programming model that allows to construct an Optimal Classification Tree for a given training sample, in which each split is generated by means of a SVM-based hyperplane. When building the tree, the labels of the observations are ignored in the branch nodes, and they are only accounted for in the leaf nodes where misclassification errors are considered. The classification tree is constructed to minimize the complexity of the tree (assuring interpretability) and also the misclassification risk (assuring predictive power).

1.1 Related works

Several machine learning methodologies have been proposed in the literature in order to construct highly predictive classification rules. The most popular ones are based on Deep Learning mechanisms (Agarwal et al., 2018), k-Nearest Neighborhoods (Cover & Hart, 1967; Tang & Xu, 2016), Naïve Bayes (Lewis, 1998), Classification Trees (CT) (Breiman et al., 1984; Friedman et al., 2001) and Support Vector Machines (SVM) (Cortes & Vapnik, 1995). Among them, CT and SVM, which are, by nature, optimization-based methodologies, apart from producing highly predictive classifiers have been proven to be very flexible tools since both allow the incorporation of different elements (through the adequate optimization models by means of constraints and objective functions) to be adapted to different situations, as Feature Selection (Günlük et al., 2018; Baldomero-Naranjo et al., 2020, 2021; Jiménez-Cordero et al., 2021), accuracy requirements (Benítez-Peña et al., 2019; Gan et al., 2021) or dealing with unbalanced or noisy instances (Eitrich & Lang, 2006; Blanco et al., 2022a; Blanquero et al., 2021), amongst others.

Support Vector Machines were originally introduced by Cortes and Vapnik (1995) as a binary classification tool that builds the decision rule by means of a separating hyperplane with large separation between the two classes. This hyperplane is obtained by solving a convex quadratic optimization problem, in which the goal is to separate data by their two differentiated classes, maximizing the margin between them and minimizing the misclassification errors. Duality properties of this optimization problem allow one to extend the methodology to find nonlinear separators by means of kernels.

Classification Trees were firstly introduced by Breiman et al. (1984), and the decision rule is based on a hierarchical relation among a set of nodes which is used to define paths that lead observations from the root node (highest node in the hierarchical relation), to some of the leaves in which a class is assigned to the data. These paths are obtained according to different optimization criteria over the predictor variables of the training sample. The decision rule comes up naturally, the classes predicted for new observations are the ones assigned to the terminal nodes in which observations fall in. Clearly, the classification rules derived from CTs are easily interpretable by means of the splits that are constructed at the tree nodes. In Breiman et al. (1984), a greedy heuristic procedure, the so-called CART approach, is presented to construct CTs. Each level of the tree is sequentially constructed: starting at the root node and using the whole training sample, the method minimizes an impurity measure function obtaining as a result a split that divides the sample into two disjoint sets which determine the two descendant nodes. This process is repeated until a given termination criteria is reached (minimum number of observations belonging to a leaf, maximum depth of the tree, or minimum percentage of observations of the same class on a leaf, amongst others). In this approach, the tree grows following a top-down greedy approach, an idea that is also shared in other popular decision tree methods like C4.5 (Quinlan, 1993) or ID3 (Quinlan, 1996). The advantage of these methods is that the decision rule can be obtained rather quickly even for large training samples, since the whole process relies on solving manageable problems at each node. Nevertheless, these types of heuristic approaches may not obtain the *optimal* classification tree, since they look for the best split locally at each node, not taking into account the splits that will come afterwards. Thus, these local branches may not capture the proper structure of the data, leading to misclassification errors in out-of-sample observations. Furthermore, the solutions provided by these methods can result into very deep (complex) trees, resulting in overfitting and, at times, loosing interpretability of the classification rule. This difficulty is usually overcome by pruning the tree as it is being constructed by comparing the gain on the impurity measure reduction with respect to the complexity cost of the tree. The recent advances on modeling and solving difficult optimization problems together with the flexibility and adaptability of these models have motivated the use of optimization tools to construct supervised classification methods with a great success (Bertsimas & Dunn, 2019; Carrizosa et al., 2021)). In particular, recently, Bertsimas and Dunn (2017) introduced the notion of *Optimal Classification Trees* (OCT) by approaching Classification and Regression Trees under optimization lens, providing a Mixed Integer Linear Programming formulation for its optimal construction. Moreover, the authors proved that this model can be solved for reasonable size datasets, and equally important, that for many different real datasets, significant improvements in accuracy with respect to CART can be obtained. In contrast to the standard CART approach, OCT builds the tree by solving a single optimization problem taking into account (in the objective function) the complexity of the tree, avoiding post pruning processes. Moreover, every split is directly applied in order to minimize the misclassification errors on the terminal nodes, and hence, OCTs are more likely to capture the hidden patterns of the data.

Recently, several algorithms have been designed to efficiently training OCTs. Demirović et al. (2022); Lin et al. (2020); Hu et al. (2019) propose different dynamic programming based algorithms to construct OCTs, also enhancing them with different features. Demirović and Stuckey (2001) develop a biobjective opitimization based approach to obtain OCT with nonlinear classification metrics (F1-score, Matthews correlation coefficient, or Fowles–Mallows index) in contrast to the classical linear accuracy goodness measure or with sparsity properties. Different methodologies have been derived also to obtain optimal decision trees based on Constraint Programming (Verhaeghe et al., 2020) or SAT (Yu et al., 2020; Hu et al., 2020; Narodytska et al., 2018). Verwer and Zhang (2019) and Firat et al. (2020) provide an alternative integer programming formulation and solution approaches to construct OCTs with a number of decision variables significantly smaller than the one proposed by Bertsimas and Dunn (2017). Finally, in order to obtain interpretable classification rules, CTs are usually derived in their univariate version, i.e., with single-variables splits. Nevertheless, oblique CT, with trees where multiple variables can be involved in the splits, have been also widely studied and one can find numerous heuristic methods in the literature,(Murthy et al., 1994; Carreira-Perpiñán & Tavallali, 2018). OCTs with oblique splits (OCT-H) have also been successfully applied. Moreover, due to the good results of oblique trees, algorithms have been designed that combine SVM with CT (Montañana et al., 2021; Chen & Ge, 2019), intending that these oblique splits also consider maximum margin between observations. Aligned with this trend, in this paper we also study an optimal method for combining oblique classification trees with SVM-based splits for multiclass instances.

While SVM were initially designed to deal only with bi-class instances, some extensions have been proposed in the literature for multiclass classification. The most popular multiclass SVM-based approaches are One-Versus-All (OVA) and One-Versus-One (OVO). The former, namely OVA, computes, for each class $r \in \{1, \ldots, k\}$, a binary SVM classifier labeling the observations as 1, if the observation is in the class r, and -1 otherwise. The process is repeated for all classes (k times), and then each observation is classified into the class whose constructed hyperplane is the furthest from it in the positive halfspace. In the OVO approach, classes are separated with $\binom{k}{2}$ hyperplanes using one hyperplane for each pair of classes, and the decision rule comes from a voting strategy in which the most represented class among votes becomes the class predicted. OVA and OVO inherit most of the good properties of binary SVM. In spite of that, they are not able to correctly classify datasets where separated clouds of observations may belong to the same class (and thus are given the same label) when a linear kernel is used. Another popular method is the directed acyclic graph SVM, DAGSVM (Agarwal et al., 2018). In this technique, although the decision rule involves the same hyperplanes built with the OVO approach, it is not given by a unique voting strategy but for a sequential number of votings in which the most unlikely class is removed until only one class remains. In addition, apart from OVA and OVO, there are some other methods based on decomposing the original multiclass problem into several binary classification ones. In particular, in Allwein et al. (2000) and Dietterich and Bakiri (1994), this decomposition is based on the construction of a coding matrix that determines the pairs of classes that will be used to build the separating hyperplanes. Alternatively, other methods such as CS (Crammer & Singer, 2001), WW (Weston & Watkins, 1999) or LLW (Lee et al., 2004), do not address the classification problem sequentially but as a whole considering all the classes within the same optimization model. Obviously, this seems to be the correct approach. In particular, in WW, k hyperplanes are used to separate the k classes, each hyperplane separating one class from the others, using $k-1$ misclassification errors for each observation. The same separating idea, is applied in CS but reducing the number of misclassification errors for each observation to a unique value. In LLW, a different error measure is proposed to cast the Bayes classification rule into the SVM problem implying theoretical statistical properties in the obtained classifier. These properties cannot be ensured in WW or CS.

We can also find a quadratic extension based on LLW proposed by Guermeur and Monfrini (2011). In van den Burg and Groenen (2016), the authors propose a multiclass SVM-based approach, *GenSVM*, in which the classification boundaries for a problem with k classes are obtained in a $(k-1)$-dimensional space using a simplex encoding. Some of these methods have become popular and are implemented in most software packages in machine learning as `e1071` (Meyer et al., 2015), `scikit-learn` (Pedregosa et al., 2011) or (Lauer & Guermeur, 2011). Finally, in the recent work (Blanco et al., 2020a) the authors propose an alternative approach to handle multiclass classification extending the paradigm of binary SVM classifiers by construnting a polyhedral partition of the feature space and an assignment of classes to the *cells* of the partition, by maximizing the separation between classes and minimizing two intuitive misclassification errors.

1.2 Contributions

In this paper, we propose a novel approach to construct Classification Trees for multiclass instances by means of a mathematical programming model. Our method is based on two main ingredients: (1) An optimal binary classification tree (with oblique cuts) is constructed in the sense of Bertsimas and Dunn (2017), in which the splits and pruned nodes are determined in terms of the misclassification errors at the leaf nodes; (2) The splits generating the branches of the tree are built by means of binary SVM-based hyperplanes separating fictitious classes (which are also decided by the model), i.e., maximizing separation between classes and minimizing the distance-based misclassification errors.

Our specific contributions include:

1. Deriving an interpretable classification rule which combines two of the most powerful tools in supervised classification, namely oblique OCT and SVM.
2. The obtained classification tree is optimally designed to approach multiclass instances by means of introducing fictitious binary classes at the intermediate splits of the OCT and compute the hinge-loss SVM errors based on that classes.
3. The classifier is constructed using a mathematical programming model that can be formulated as a Mixed Integer Second Order Cone Programming problem that can be solved using any of the available off-the-shelf optimization software. The classifier is simple to apply and interpretable.
4. Several valid inequalities are presented for the formulation that allow one to strengthen the model and to solve larger size instances in smaller CPU times.
5. Our approach, based on the results of the computational experiments that we perform on realistic datasets from UCI, outperforms some of the state-of-the-art decision tree-based methodologies as CART, OCT and OCT-H.

1.3 Paper structure

Section 2 is devoted to fix the notation and to recall the tools that are used to derive our method. In Sect. 3 we detail the main ingredients of our approach and illustrate its performance on a toy example. The mathematical programming model that allows us to construct the classifier is given in Sect. 4, where we include all the elements involved in the model: parameters, variables, objective function and constraints. In Sect. 5 we report the results of our experiments to assess the performance of our method compared with other tree-shaped classifiers. Finally, Sect. 6 is devoted to draw some conclusions and future research lines on the topic.

2 Preliminaries

This section is devoted to introduce the problem under study and to fix the notation used through this paper. We also recall the main tools involved in our proposed approach namely, Support Vector Machines and Optimal Classification Trees. These methods are adequately combined to develop a new method, called Multiclass Optimal Classification Trees with Support Vector Machines based splits (MOCTSVM).

We are given a training sample, $\mathcal{X} = \left\{(x_1, y_1), \dots, (x_n, y_n)\right\} \subseteq \mathbb{R}^p \times \{1, \dots, K\}$, which comes as the result of measuring p features over a set of n observations $(x_1, \dots, x_n)$ as well as a label in $\{1, \dots, K\}$ for each of them $(y_1, \dots, y_n)$. The goal of a classification method is to build a decision rule so as to accurately assign labels (y) to data (x) based on the behaviour of the given training sample $\mathcal{X}$.

The first ingredient that we use in our approach is the Support Vector Machine method. SVM is one of the most popular optimization-based methods to design a classification rule in which only two classes are involved, usually referred as the positive $(y = +1)$ and the negative class $(y = -1)$. The goal of linear SVM is to construct a hyperplane separating the two classes by maximizing their separation and simultaneously minimizing the misclassification and margin violation errors. Linear SVM can be formulated as the following convex optimization problem:

$$\begin{aligned}
\min\ & \frac{1}{2}\|\omega\|_2^2 + c\sum_{i\in N} e_i\\
\text{s.t.}\ \ & y_i(\omega' x_i + \omega_0) \geq 1 - e_i, && \forall i \in N,\\
& \omega \in \mathbb{R}^p,\ \omega_0 \in \mathbb{R},\\
& e_i \in \mathbb{R}_+, && \forall i \in N.
\end{aligned}$$

where c is the regularization parameter that states the trade-off between training errors and model complexity (margin), ω' is the transpose of the vector ω and $\|\cdot\|_2$ is the Euclidean norm in $\mathbb{R}^p$ (other norms can also be considered but still keeping similar structural properties of the optimization problem (Blanco et al., 2020b)). Note that with this approach, the positive (resp. negative) class will tend to lie on the positive (resp. negative) half space induced by the hyperplane $\mathcal{H} = \{z \in \mathbb{R}^p : \omega' z + \omega_0 = 0\}$. On the other hand, the popularity of SVM is mostly due to the so called kernel trick. This allows one to project the data onto a higher dimensional space in which a linear separation is performed in a most accurate way with no need of knowing such a space, but just knowing the form of its inner products, and maintaining the computational complexity of the optimization problem (see Cortes and Vapnik (1995) for further details).

The second method that we combine in our approach is Classification Trees. CTs are a family of classification methods based on a hierarchical relationship among a set of nodes. These methods allow one to create a partition of the feature space by means of hyperplanes that are sequentially built. CT starts on a node containing the whole sample, that is called the root node, in which the first split is applied. When applying a split on a node, by means of a hyperplane separating the observations, two new branches are created leading to two new nodes, which are referred to as its child nodes. The nodes are usually distinguished into two groups: branch nodes, that are nodes in which a split is applied, and on the other hand the leaf nodes, which are the terminal nodes of the tree. Given a branch node and a hyperplane split in such a node, their branches (left and right) are defined as each of the two halfspaces defined by the hyperplane. The final goal of CT is to construct branches in order to obtain leaf nodes as pure as possible with respect to the classes. In this way, the classification rule for a given observation consists of assigning it to the most popular class of the leaf where it belongs to.

There is a vast amount of literature on CTs since they provide an easy, interpretable classification rule. One of the most popular methods to construct CT is known as CART, introduced in Breiman et al. (1984). However, CART does not guarantee the *optimality* of the classification tree, in the sense that more accurate trees could be obtained if instead of locally constructing the branches one looks at the final configuration of the leaf nodes. For instance, in Fig. 1(left) we show a CT constructed by CART for a biclass problem with maximal depth 2. We draw the classification tree, and also in the top right corner, the partition of the feature space (in this case $\mathbb{R}^2$). As can be observed, the obtained classification is not perfect (not all leaf nodes are composed by pure classes) while in this case is not difficult to construct a CT with no classification errors. This situation is caused by the myopic construction done by the CART approach that, at each node only cares on better classification at their children, but not at the final leaf nodes, while subsequent branching decisions clearly affect the overall shape of the tree.

Motivated by this drawback of CART, in Bertsimas and Dunn (2017), the authors propose an approach to build an Optimal Classification Tree (OCT) by solving a single Mathematical Programming problem in which not only single-variable splits are possible but *oblique* splits involving more than one predictive variable (by means of general hyperplanes in the feature space) can be constructed. In Fig. 1(right) we show a solution provided by OCT with hyperplanes (OCT-H) for the same example. One can observe that

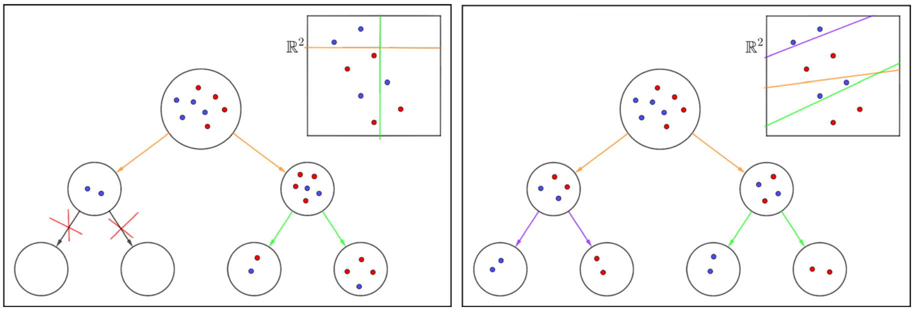


Fig. 1 Example of a CT obtained with CART (left) and OCT-H (right) approaches for the same instance

when splitting the root node (orange branches) a good local split is not obtained (the nodes contain half of the observations in different classes), however, when adding the other two splits, the final leaves only have observations of the same class, resulting in a perfect classification rule for the training sample.

Both approaches, OCTs and SVMs can be combined in order to construct classification trees in which the classes separated by the hyperplanes determined in the CT are *maximally* separated, in the sense of the SVM approach. This idea is not new and has been proven to outperform standard optimal decision trees methods amongst many different biclass classification problems, as for instance, in Blanco et al. (2022b) where the OCTSVM method is proposed. In Fig. 2 we show how one could construct OCTs with larger separations between the classes using OCTSVM but still with the same 100% accuracy in the training sample as in OCT-H, but more protected to misclassification in out-sample observations.

Nevertheless, as far as we know, the combination of OCT and SVM has only been analyzed for biclass instances. The extension of this method to multiclass settings (more than two classes) is not trivial, since one could construct more complex trees or use a multiclass SVM-based methodology (see e.g., Crammer & Singer, 2001; Weston & Watkins, 1999; Lee et al., 2004). However, these adaptations of the classical SVM method have been proved to fail in real-world instances (see e.g., Blanco et al., 2020a). In the rest of the paper we describe a novel methodology to construct accurate multiclass tree-shaped classifiers based on a different idea: constructing CTs with splits induced by bi-class SVM separators in which the classes of the observations at each one of the branch nodes are determined by the model, but adequately chosen to provide small classification errors at the leaf nodes. The details of the approach are given in the next section.

3 Multiclass OCT with SVM splits

In this section we describe the method that we propose to construct classification rules for multiclass instances, in particular Classification Trees in which splits are generated based on the SVM paradigm.

As already mentioned, our method is based on constructing OCT with SVM splits, but where the classes of the observations are momentarily ignored and only accounted for at the leaf nodes. In order to illustrate the idea under our method, in Fig. 3 we show a toy instance with a set of points with four different classes (blue, red, orange and green).

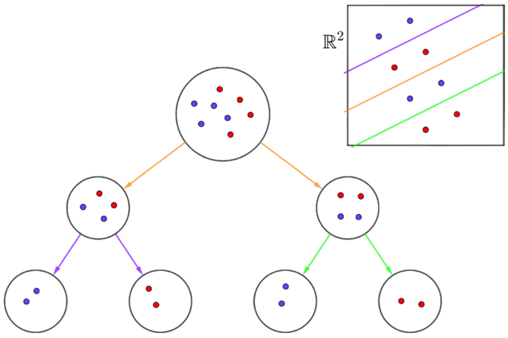


Fig. 2 Example of a CT obtained with OCTSVM

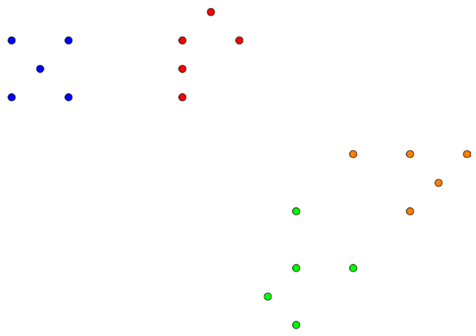


Fig. 3 Instance for a 4-class problem (Color figure online)

First, at the root node (the one in which all the observations are involved), our method constructs a SVM separating hyperplane for two fictitious classes (which have to be also decided). A possible separation could be the one shown in Fig. 4, in which the training dataset has been classified into two classes (black and white).

This separation allows one to generate two child nodes, the black and the white nodes. At each of these nodes, the same idea is applied until the leaf nodes are reached. In Fig. 5 we show the final partition of the feature space according to this procedure.

Clearly, ignoring the original classes of the training sample in the whole process would result in senseless trees, unless one accounts for the goodness in the classification rule in the training sample at the leaf nodes. Thus, at the final leaf nodes, the original labels are recovered and the classification is performed according to the generated hyperplanes. The final result of this tree is shown in Fig. 6 where one can check that the constructed tree achieves a perfect classification of the training sample.

Once the tree is constructed with this strategy, the decision rule comes up naturally as it is usually done in decision trees methods, that is, out of sample observations will follow a path on the tree according to the splits and they will be assigned to the class of the leaf

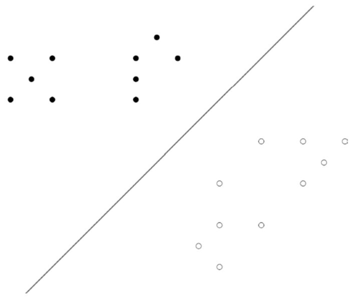


Fig. 4 Root split on the 4-class classification problem

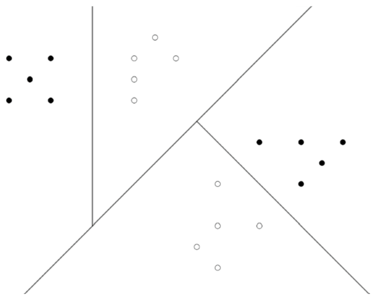


Fig. 5 Child node splits on the 4-class classification problem with the two fictitious classes decided by our model

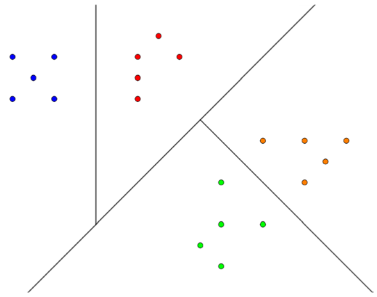


Fig. 6 Child node splits on the 4-class classification problem with their original labels (colors) (Color figure online)

where they lie in (the most represented class of the leaf over the training set). In case a branch is pruned when building the tree, observations will be assigned to the most represented class of the node where the prune took place.

4 Mathematical programming formulation for MOCTSVM

In this section we derive a Mixed Integer Non Linear Programming formulation for the MOCTSVM method described in the previous section.

We assume to be given a training sample $\mathcal{X} = \left\{(x_1, y_1), \dots, (x_n, y_n)\right\} \subseteq \mathbb{R}^p \times \{1, \dots, K\}$. We denote by $N = \{1, \dots, n\}$ the index set for the observations in the training sample. We also consider the binary representation of the labels y as:

$$Y_{ik} = \begin{cases} 1 & \text{if } y_i = k, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for all } \ i \in N, \ k = 1, \dots K.$$

Moreover, without loss of generality we will assume the features to be normalized, i.e., $x_1, \dots, x_n \in [0,1]^p$.

We will construct decision trees with a fixed maximum depth D. Thus, the classification tree is formed by at most $T = 2^{D+1} - 1$ nodes. We denote by $\tau = \{1, \dots, T\}$ the index set for the tree nodes, where node 1 is the root node and nodes $2^D, \dots, 2^{D+1} - 1$ are the leaf nodes.

For any node $t \in \tau \backslash \{1\}$, we denote by $p(t)$ its (unique) parent node. The tree nodes can be classified in two sets: branching and leaf nodes. The branching nodes, that we denote by τ_b, will be those in which the splits are applied. In constrast, in the leaf nodes, denoted by τ_l, no splits are applied but is where predictions take place. The branching nodes can be also classified into two sets: τ_{bl} and τ_{br} depending on whether they follow the left or the right branch on the path from their parent nodes, respectively. τ_{bl} nodes are indexed with even numbers meanwhile τ_{br} nodes are indexed with odd numbers.

We define a level as a set of nodes which have the same depth within the tree. The number of levels in the tree to be built is $D + 1$ since the root node is assumed as the zero-level. Let $U = \{u_0, \dots, u_D\}$ be the set of levels of the tree, where each $u_s \in U$ is the set of nodes at level s, for $s = 0, \dots, D$. With this notation, the root node is u_0 while u_D represent the set of leaf nodes.

In Fig. 7 we show the above mentioned elements in a 3-depth tree.

Apart from the information about the topological structure of the tree, we also consider three regularization parameters that have to be calibrated in the validation process that allow us to find a trade-off between the different goals that we combine in our model: margin violation and classification errors of the separating splitting hyperplanes,

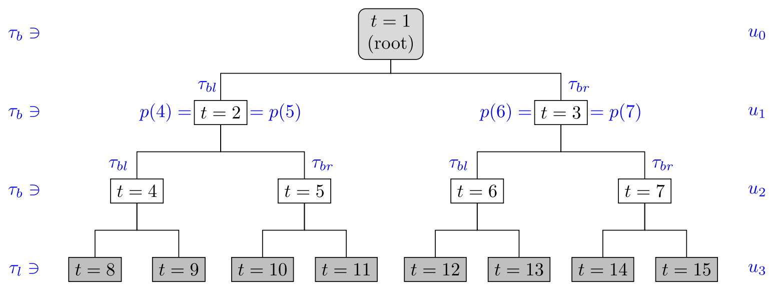


Fig. 7 Elements in a depth $D = 3$ tree

correct classification at the leaf nodes and complexity of the tree. These parameters are the following:

c_1	: unit misclassification cost at the leaf nodes.
c_2	: unit distance based misclassification errors for SVM splits.
c_3	: unit cost for each splitting hyperplane introduced in the tree.

The complete list of index sets and parameters used in our model are summarized in Table 1.

4.1 Variables

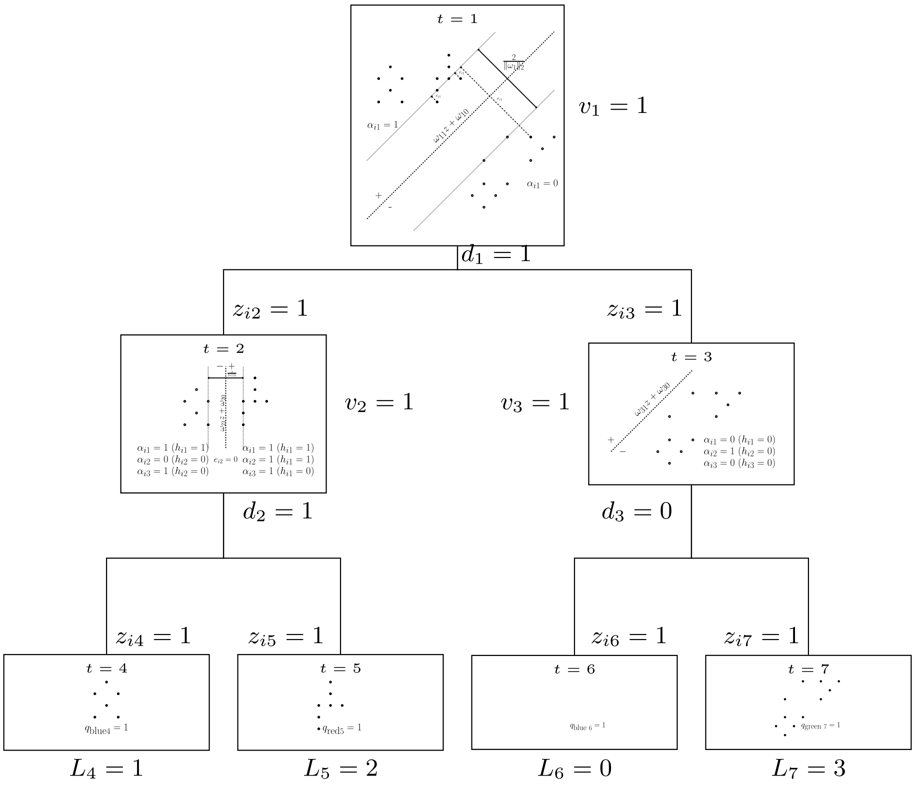
Our model uses a set of decision and auxiliary variables that are described in Table 2. We use both binary and continuous decision variables to model the MOCTSVM. The binary variables allow us to decide the allocation of observations to the decision tree nodes, or to decide whether a node is splited or not in the tree. The continuous variables allow us to determine the coefficients of the splitting hyperplanes or the misclassification errors (both

Table 1 Index sets and parameters used in our model

$N = \{1, \dots, n\}$	Index set for the observations in the training sample
D	Maximal depth of the tree
$T = 2^{D+1}$	Maximal number of nodes in a D-depth tree
$\tau = \{1, \dots, T\}$	Index set for the set of nodes of the tree
$p(t)$	Parent of node t, for $t \in \tau \backslash \{1\}$
$\tau_b \in \tau$	Branching nodes of the tree
τ_l	Leaf nodes of the tree
$\tau_{bl} \in \tau_b$	Nodes that follow the left branch on the path from their parent nodes
$\tau_{br} \in \tau_b$	Nodes whose right branch has been followed on the path from their parent nodes
u_s	Nodes at level s of the tree, for $s = 0, \dots, D$
$U = \{u_0, \dots, u_d\}$	Sets of levels of the tree
c_1	Unit misclassification cost
c_2	Unit distance based missclassification errors for SVM splits
c_3	Unit cost for splitting hyperplanes

Table 2 Summary of the variables used in our model

Continuous decision variables	
$\omega_t \in \mathbb{R}^p$	Coefficients of the separating hyperplane of node t
$\omega_{t_0} \in \mathbb{R}$	Intercept of the separating hyperplane of node t
$e_{it} \in \mathbb{R}_+$	Misclassification error of observation i at node t
$\delta \in \mathbb{R}_+$	Inverse of the minimum margin between splitting hyperplanes
Binary decision variables	
$z_{it} \in \{0, 1\}$	Is one if observation i belongs to node t and zero otherwise
$d_t \in \{0, 1\}$	Is one if a split is applied at node t and zero otherwise
Auxiliary variables	
$L_t \in \mathbb{Z}_+$	Number of misclassified observations at leaf node t
$\alpha_{it} \in \{0, 1\}$	Is one if observation i belongs to the reference fictitious class in node t and zero otherwise
$h_{it} \in \{0, 1\}$	Is one if observation i is in node t and lies on the positive half space of the hyperplane of node t, and zero otherwise
$v_t \in \{0, 1\}$	Is one if not all observations in node t lie on the positive half space of the hyperplane in node t and zero otherwise
$q_{kt} \in \{0, 1\}$	Is one if class k is the most represented one in leaf node t and zero otherwise



$t=1$
$\frac{2}{\|\omega_1\|_2}$
$\omega_1^T x + \omega_{10}$
$\alpha_{i1}=1$
$\alpha_{i1}=0$
$v_1 = 1$
$d_1 = 1$
$z_{i2} = 1$
$z_{i3} = 1$
$t=2$
$v_2 = 1$
$\alpha_{i1}=1\ (h_{i1}=1)$
$\alpha_{i2}=0\ (h_{i2}=0)$
$e_{i2}=0$
$\alpha_{i3}=1\ (h_{i2}=0)$
$\alpha_{i1}=1\ (h_{i1}=1)$
$\alpha_{i2}=1\ (h_{i1}=1)$
$\alpha_{i3}=1\ (h_{i1}=0)$
$t=3$
$v_3 = 1$
$\omega_3^T x + \omega_{30}$
$\alpha_{i1}=0\ (h_{i1}=0)$
$\alpha_{i2}=1\ (h_{i2}=0)$
$\alpha_{i3}=0\ (h_{i3}=0)$
$d_2 = 1$
$d_3 = 0$
$z_{i4} = 1$
$z_{i5} = 1$
$z_{i6} = 1$
$z_{i7} = 1$
$t=4$
$q_{\text{blue}4}=1$
$t=5$
$q_{\text{red}5}=1$
$t=6$
$q_{\text{blue }6}=1$
$t=7$
$q_{\text{green }7}=1$
$L_4 = 1$
$L_5 = 2$
$L_6 = 0$
$L_7 = 3$

Fig. 8 Illustration of the sets of variables used in our model in a toy example (Color figure online)

in the SVM separations or at the leaf nodes). We also use auxiliary binary and integer variables that are useful to model adequately the problem.

In Fig. 8 we illustrate the use of these variables in a feasible solution of a toy instance with three classes (red,blue and green).

The whole set of training observation is considered at the root node (node $t = 1$). There, the original labels are ignored and to determine the *fictitious* class of each observation a SVM-based hyperplane is constructed. Such a hyperplane is defined by the coefficients $\omega_1 \in \mathbb{R}^p$ and $\omega_{10} \in \mathbb{R}$ (hyperplane/line drawn with a dotted line in the picture) and it induces a margin separation $\left(\frac{2}{\|\omega_1\|_2}\right)$ and misclassification errors e_{i1}. In the feasible solution drawn in the figure, only three observations induce positive errors (those that are classified either in the margin area or in the opposite side of the hyperplane). Such a hyperplane also determines the splitting rule for the definition of the children of that node. Since the node is split ($d_1 = 1$), the observations that belongs to the *positive side* of the hyperplane are assigned to the left node (node $t = 2$) while those in the negative side are assigned to the right node (node $t = 3$) through the z-variables. At node $t = 2$, the same scheme is applied, that is, the hyperplane defined by ω_2 is constructed, inducing SVM-based margin and errors and since $d_2 = 1$, also the splitting rule applies to define nodes $t = 4$ and $t = 5$. At node $t = 2$, one must control the observations in that node to quantify the misclassifying errors, e_{i2}, only for those observations in the objective function. Specifically, we only account for these errors for the observations that belong to the node ($z_{i2} = 1$) and either belong to the positive ($\alpha_{i2} = 1$) or the negative ($\alpha_{i2} = 0$) side of the hyperplane. Also, in order to control the complexity of the tree, the h-variables are used to know whether an observation belongs to the node and to the positive side of the SVM-hyperplane. If all observations in a node belong to the positive side of the hyperplane, the variable v assumes the value 0. Otherwise, in case v takes value 1, two situations are possible: (1) there are observations in both sides of the hyperplane (as in node $t = 2$) inducing a new split ($d_2 = 1$), and (2) all observations belong to the negative side (as in node $t = 3$) determining that the tree is pruned at that node ($d_3 = 0$).

Concerning the leaf nodes, node $t = 2$ is split into nodes $t = 4$ and $t = 5$ and node $t = 3$, which was decided to be no longer split, is fictitiously split in two leaf nodes, although one of them is empty and the other one receives all the observations of the parent node (node $t = 3$). The allocation of any leaf node τ_l to a class is done through the q-variables (to the most popular class in the node or arbitrarily in case the node has no observations) and the number of misclassified observations is accounted for by the L-variables.

4.2 Objective function

As already mentioned, our method aims to construct classification trees with small misclassification errors at the leaf nodes, but at the same time with maximal separation between the classes with the SVM-based hyperplanes and minimum distance based errors.

Using the variables described in the previous section, the four terms that are included in the objective functions are the following:

Margins of the splitting hyperplanes: The separating hyperplane of branching node $t \in \tau_b$ has margin $\frac{2}{\|\omega_t\|_2}$. Thus, our method aims to maximize the minimum of these margins. This is equivalent to minimize the maximum among the inverse margins $\left\{\frac{1}{2}\|\omega_t\|_2^2 : t \in \tau_b\right\}$ which is represented by the auxiliary variable δ.

Misclassification Errors at the leaf nodes: Variable L_t accounts for the number of misclassified observations in leaf node t, i.e., the number of observations that do not belong to the most represented class in that leaf node. These variables allow us to count the overall number of misclassified observations in the training sample. Therefore, the amount to be minimized by the model is given by the following sum:

$$c_1 \sum_{t \in \tau_l} L_t$$

Distance-based Errors at branching nodes: Each time a split is added to the tree, a SVM-based hyperplane in which the labels are assigned based on the global convenience of for the overall tree is incorporated. Thus, we measure, at each branching node in τ_b, the distance-based errors incurred by the SVM classifier at that split. This amount is measured by the e_{it} variables and is incorporated to the model through the sum:

$$c_2 \sum_{i \in N} \sum_{t \in \tau_b} e_{it}$$

Complexity of the tree The simplicity of the resulting tree is measured by the number of splits that are done in its construction. Since the d_t variable tells us whether node t is split or not, this term is accounted for in our model as:

$$c_3 \sum_{t \in \tau_b} d_t$$

Summarizing, the overall objective function of our model is:

$$\min \ \delta + c_1 \sum_{t \in \tau_l} L_t + c_2 \sum_{i \in N} \sum_{t \in \tau_b} e_{it} + c_3 \sum_{t \in \tau_b} d_t. \tag{OBJ}$$

Note that the coefficients c_1, c_2 and c_3 trade-off the misclassification of the training sample, the separation between classes and the complexity of the tree, respectively. These parameters should be carefully calibrated in order to construct *simple* decision trees with high predictive power, as can be seen in our computational experiments.

4.3 Constraints

The requirements on the relationships between the variables and the rationale of our model are described through the following constraints that define the mathematical programming model.

First of all, in order to adequately represent the maximum among the inverse margins of the splitting hyperplanes, we require:

$$\delta \geq \frac{1}{2} \|\omega_t\|_2^2, \forall t \in \tau_b. \tag{C1}$$

Next, we impose how the splits are performed in the tree. To this end, we need to know which observations belong to a certain node t (z-variable) and how these observations are distributed with respect to the two fictitious classes to be separated (α-variables). Gathering all these elements together, we use the following constraints to define the splits of the decision tree:

$$\omega_t' x_i + \omega_{t0} \geq 1 - e_{it} - (1 - h_{it}) \quad \forall i \in N, t \in \tau_b, \tag{C2a}$$

$$\omega_t' x_i + \omega_{t0} \leq -1 + e_{it} + (1 - z_{it} + \alpha_{it}) \quad \forall i \in N, t \in \tau_b. \tag{C2b}$$

According to this, constraint (C2a) is activated just in case the observation i belongs to the reference class and it is in node t ($h_{it} = 1$). On the other hand, (C2b) is activated if i is allocated to node t ($z_{it} = 1$) but it does not belong to the reference class ($\alpha_{it} = 0$). Therefore, the reference class is located on the positive half space of hyperplane $\mathcal{H}_t$, while the other class is positioned in the negative half space, and at the same time, margin violations are regulated by the e_{it} variables.

To ensure the correct behaviour of the above constraints, we must correctly define the z_{it} variables. First, it is required that each observation belongs to exactly one node per level in the tree. This can be easily done by adding the usual assingment constraints to the problem at each of the levels, $u \in U$, of the tree:

$$\sum_{t \in u} z_{it} = 1 \quad \forall i \in N, u \in U. \tag{C3}$$

Furthermore, we should enforce that if observation i is in node t ($z_{it} = 1$), then observation i must also be in the parent node of t, $p(t)$ ($z_{ip(t)} = 1$), and also observation i can not be in node t if it is not in its parent node ($z_{ip(t)} = 0 \Rightarrow z_{it} = 0$). These implications can be obtained by means of the following constraints:

$$z_{it} \leq z_{ip(t)} \quad \forall i \in N, t = 2, \dots, T. \tag{C4}$$

Nevertheless, the way observations descend through the tree needs a further analysis, since at this point they could just randomly define a path in the tree. Whenever an observation i is in the positive half space of the splitting hyperplane at node t, $\mathcal{H}_t$, this observation should follow the right branch connecting to the child node of t. Otherwise, in case i is on the negative half space, it should follow the left branch. The knowledge on the side of the splitting hyperplane where an observation belongs to is encoded in the α-variables. Then, in case i lies on the positive half space of $\mathcal{H}_t$, α_{it} will never be equal to zero since it would lead to a value of e_{it} greater than one, while $e_{it} < 1$ is guaranteed in case $\alpha_{it} = 1$.

With the above observations, the constraints that assure the correct construction of the splitting hyperplanes with respect to the side of them where the observations belong to are the following:

$$z_{ip(t)} - z_{it} \leq \alpha_{ip(t)} \quad \forall i \in N, t \in \tau_{bl}, \tag{C5a}$$

$$z_{ip(t)} - z_{it} \leq 1 - \alpha_{ip(t)} \quad \forall i \in N, t \in \tau_{br}. \tag{C5b}$$

Constraints (C5a) assure that if observation i is on the parent node of an even node t $\left(z_{ip(t)} = 1\right)$, and i lies on the negative half space of $\mathcal{H}_{p(t)}$ $\left(\alpha_{ip(t)} = 0\right)$, then z_{it} is enforced to be equal to one. As a result, $\alpha_{ip(t)} = 0$ forces observation i to take the left branch in node t. Note that in case $z_{ip(t)} = 1$, and at the same time observation i is not in the left child node of t($z_{it} = 0$ for $i \in \tau_{bl}$), then $\alpha_{ip(t)} = 1$, which means that observation i lies on the positive half space of $\mathcal{H}_{p(t)}$. Constraints (C5b) are analogous to (C5a) but allowing to adequately represent right branching nodes.

Moreover, two additional important elements need to be incorporated to complete our model: the tree complexity and the correct definition of misclassified observations. Note that in usual Optimal Classification Trees that do not use SVM-based splits, the complexity can be easily regulated by just imposing $\|\omega_t\|_2^2 \leq Md_t$ (for a big enough M constant) in all the branch nodes, since in case a node is no further branched ($d_t = 0$), the coefficients of the splitting hyperplane are set to zero. However, in our case, in which the splitting hyperplanes are SVM-based hyperplanes, these constraints are in conflict with constraints (C2a) and (C2b), since in case $d_t = 0$ (and therefore $\omega_t = 0$) it would not only imply that the coefficients ω_t are equal to zero, but also that the distance based errors would be set to the maximum value of 1, i.e., $e_{it} = 1$ for every observation i in the node, even though these errors would not make any sense since observations would not be separated at the node. To overcome this issue, we consider the auxiliary binary variables $h_{it} = z_{it}\alpha_{it}$ (h_{it} takes value 1 if observation i belongs to node t and lies in the positive half-space of the splitting hyperplane applied at node t) and v_t (that takes value zero in case all the points in the node belong to the positive halfspace and one otherwise). The variables are adequatelly defined if the following constraints are incorporated to the model:

$$h_{it} \geq z_{it} + \alpha_{it} - 1, \quad \forall i \in N, t \in \tau_b, \tag{C6a}$$

$$h_{it} \leq z_{it} - \alpha_{it} + 1, \quad \forall i \in N, t \in \tau_b, \tag{C6b}$$

$$\sum_{i \in N} (z_{it} - h_{it}) \leq n v_t, \quad \forall t \in \tau_b, \tag{C6c}$$

$$\sum_{i \in N} h_{it} \leq n(1 + d_t - v_t), \quad \forall t \in \tau_b, \tag{C6d}$$

where constraints (C6a) and (C6b) are the linearization of the bilinear constraint $h_{it} = z_{it}\alpha_{it}$. On the other hand, Constraints (C6c) assure that in case $v_t = 0$, then all observations in node t belong to the positive halfspace of $\mathcal{H}_t$, and constraints (C6d) assure that if $v_t = 1$ and the tree is pruned at node t ($d_t = 0$), then those observations allocated to node t are placed in the negative halfspace defined by the splitting hyperplane. Thus, it implies that d_t takes value one if and only if the observations in node t are separated by $\mathcal{H}_t$, and therefore producing an effective split at the node.

Finally, in order to adequately represent the L_t variables (the ones that measure the number of misclassified observations at the leaf nodes) we use the constraints already incorporated in the OCT-H model in Bertsimas and Dunn (2017). On the one hand, we assign each leaf node to a single class (the most popular class of the observations that belong to that node). We use the binary variable q_{kt} to check whether leaf node $t \in \tau_l$ is assigned to class $k = 1, \ldots, K$. The usual assignment constraints are considered to assure that each node is assigned to exactly one class:

$$\sum_{k=1}^{K} q_{kt} = 1, \quad \forall t \in \tau_l. \tag{C7}$$

The correct definition of the variable L_t is then guaranteed by the following set of constraints:

$$L_t \geq \sum_{i \in N} z_{it} - \sum_{i \in N} Y_{ik} z_{it} - n(1 - q_{kt}), \quad \forall k = 1, \ldots, K, t \in \tau_l, \tag{C8}$$

These constraints are activated if and only if $q_{kt} = 1$, i.e., if observations in node t are assigned to class k. In such a case, since L_t is being minimized in the objective function, L_t will be determined by the number of training observations in node t except those whose label is k, i.e., the number of missclasified observations in node t according to the k-class assignment.

Observe that the constant n in (C8) can be decreased and fixed to the maximum number of misclassified observations in the training sample. This number coincide with the difference between the number of observations in the training sample (n) and the number of observations in the most represented class in the sample.

Summarizing the above paragraphs, the MOCTSVM can be formulated as the following MINLP problem:

$$\min \quad \delta + c_1 \sum_{t\in\tau_l} L_t + c_2 \sum_{i\in N}\sum_{t\in\tau} e_{it} + c_3 \sum_{t\in\tau} d_t \tag{OBJ}$$

$$\text{s.t.} \quad \delta \geq \frac{1}{2}\|\omega_t\|, \quad \forall t \in \tau_b, \tag{C1}$$

$$\omega_t' x_i + \omega_{t0} \geq 1 - e_{it} - (2 - z_{it} - \alpha_{it}), \quad \forall i \in N, t \in \tau_b, \tag{C2a}$$

$$\omega_t' x_i + \omega_{t0} \leq -1 + e_{it} + (1 - z_{it} + \alpha_{it}), \quad \forall i \in N, t \in \tau_b, \tag{C2b}$$

$$\sum_{t\in u} z_{it} = 1, \quad \forall i \in N, u \in U, \tag{C3}$$

$$z_{it} \leq z_{ip(t)}, \quad \forall i \in N, t = 2, \dots, T, \tag{C4}$$

$$z_{ip(t)} - z_{it} \leq \alpha_{ip(t)}, \quad \forall i \in N, t \in \tau_{bl}, \tag{C5a}$$

$$z_{ip(t)} - z_{it} \leq 1 - \alpha_{ip(t)}, \quad \forall i \in N, t \in \tau_{br}, . \tag{C5b}$$

$$h_{it} \geq z_{it} + \alpha_{it} - 1, \quad \forall i \in N, t \in \tau_b, \tag{C6a}$$

$$h_{it} \leq z_{it} - \alpha_{it} + 1, \quad \forall i \in N, t \in \tau_b, \tag{C6b}$$

$$\sum_{i\in N} (z_{it} - h_{it}) \leq n v_t, \quad \forall t \in \tau_b, \tag{C6c}$$

$$\sum_{i\in N} h_{it} \leq n(1 + d_t - v_t), \quad \forall t \in \tau_b, \tag{C6d}$$

$$\sum_{k=1}^{K} q_{kt} = 1, \quad \forall t \in \tau_l, \tag{C7}$$

$$
\begin{aligned}
& L_t \geq \sum_{i \in N} z_{it} - \sum_{i \in N} Y_{ik} z_{it} - n(1-q_{kt}), && \forall k = 1, \dots, K, t \in \tau_l, \\
& e_{it} \in \mathbb{R}^+, \alpha_{it}, h_{it} \in \{0,1\}, && \forall i \in N, t \in \tau_b, \\
& z_{it} \in \{0,1\}, && \forall i \in N, t = 1, \dots, T, \\
& q_{kt} \in \{0,1\}, && \forall k = 1 \dots, K, t \in \tau_l, \\
& \omega_t \in \mathbb{R}^p, \omega_{t0} \in \mathbb{R}, d_t \in \{0,1\}, && \forall t = 1, \dots, T.
\end{aligned} \tag{C8}
$$

4.4 Strengthening the model

The MINLP formulation presented above is valid for our MOCTSVM model. However, it is a computationally costly problem, and although it can be solved by most of the off-the-shelf optimization solvers (as Gurobi, CPLEX or XPRESS), it is able to solve optimally only small to medium size instances. To improve its performance, the problem can be strengthen by means of valid inequalities which allows one to reduce the gap between the continuous relaxation of the problem and its optimal integer solution, being then able to solve larger instances in smaller CPU times. In what follows we describe some of these inequalities that we have incorporated to the MINLP formulation:

- If observations i and i' belongs to different nodes, they cannot be assigned to the same node for the remainder levels of the tree:

$$z_{is} + z_{i's} \leq z_{it} + z_{i't}, \forall t \in u, s \in u' u \leq u'$$

- If leaf nodes t and s are the result of proper splitting hyperplanes, then, both nodes cannot be assigned to the same class:

$$q_{kt} + q_{ks} \leq 2 - d_{p(t)}, \forall t, s = 2, \dots, T (t \neq s) \text{ with } p(t) = p(s), k = 1, \dots, K.$$

- Variable α_{it} is enforced to take value 0 in case $z_{it} = 0$:

$$\alpha_{it} \leq z_{it}, \forall i \in N, t \in \tau_b.$$

- Variable h_{it} is not allowed to take value one if α_{it} takes value zero:

$$h_{it} \leq \alpha_{it}, \forall i \in N, t \in \tau_b.$$

- There should be at least a leaf node to which each class is assigned to (assuming that each class is represented in the training sample). It also implies that the number of nodes to which a class is assigned is bounded as:

$$1 \leq \sum_{t \in \tau_l} q_{kt} \leq 2^D - 1, \forall k = 1, \dots, K.$$

In order to reduce the dimensionality and also to avoid symmetries of the MINLP problem, one can also apply some heuristic strategies to fix the values of some of the binary variables in a preprocessing phase. Specifically, we choose an observation in the training sample, i_0, with maximum number of observations in the same class *close enough to* it (at distance less or equal than a given $\varepsilon > 0$), i.e., $i_0 \in \arg\max_{i \in N} \mid \{i' \in N : \|a_i - a_{i'}\| \leq \varepsilon \text{ and } y_i = y_{i'}\} \mid$. This observation, and all the observations in the same class which are close enough to it will be likely predicted in the same node. Thus, we fix to one all the z-variables assigning these selected observations to a given leaf node (for example to the first left leaf node of the tree) and fixing to zero the allocation of these points to the rest of the leaf nodes.

5 Experiments

In order to analyze the performance of this new methodology we have run a series of experiments among different real datasets from UCI Machine learning Repository (Asuncion & Newman, 2007). We have chosen twelve datasets with number of classes between two and seven. The dimension of these problems is reported in Table 3 by the tuple (n : number of observations, p : number of features, K : number of classes).

We have compared the MOCTSVM model with three other Classification Tree-based methodologies, namely CART, OCT and OCT-H. The maximum tree depth, D, for all the models was equal to 3, and the minimum number of observations per node in CART, OCT and OCT-H was equal to the 5% of the training size.

We have performed, for each instance a 5-fold cross validation scheme, i.e., datasets have been splited into five random train-test partitions where one of the folds is used to build the model and the remaining are used to measure the accuracy of the predictions. Moreover, in order to avoid taking advantage of beneficial initial partitions, we have repeated the cross-validation scheme five times for all the datasets.

The CART method was coded in `R` using the `rpart` library. On the other hand, MOCTSVM, OCT and OCT-H were coded in `Python` and solved using the optimization solver `Gurobi` 8.1.1. All the experiments were run on a PC Intel Xeon E-2146 G processor at 3.50GHz and 64GB of RAM. A time limit of 300 s was set for training the training folds. Although not all the problems were optimally solved within the time limit, as can be observed in Table 3, the results obtained with our model already outperform the other methods.

Table 3 Average accuracies (± standard deviations) obtained in our computational experiments

	CART	OCT	OCT-H	MOCTSVM	Diff
Australian (690,14,2)	85.54 ± 0.81	85.22 ± 1.27	**85.65 ± 1.02**	85.27 ± 1.11	− 0.38 ± 0.63
BalanceScale (625,4,3)	69.55 ± 1.76	73.30 ± 1.20	**90.43 ± 1.07**	89.53 ± 1.28	− 0.90 ± 1.69
Banknote (1372,5,2)	89.27 ± 0.95	88.50 ± 1.17	98.89 ± 0.33	**98.91 ± 0.46**	0.02 ± 0.43
BreastCancer (683,9,2)	92.69 ± 1.01	94.16 ± 0.54	95.10 ± 1.26	**96.27 ± 0.64**	1.17 ± 1.39
Dermatology (358,34,6)	75.69 ± 3.60	77.82 ± 4.34	91.41 ± 2.83	**95.39 ± 1.47**	3.98 ± 2.74
Heart (294,13,5)	64.37 ± 1.48	65.14 ± 1.57	64.30 ± 1.79	**66.41 ± 1.54**	1.26 ± 1.38
Iris (150,4,3)	94.26 ± 1.90	95.37 ± 0.97	95.64 ± 1.46	**95.72 ± 1.79**	0.08 ± 1.70
Parkinson (240,40,2)	72.29 ± 4.05	73.53 ± 2.26	74.92 ± 3.01	**80.83 ± 1.89**	5.91 ± 3.06
Seeds (210,7,3)	86.36 ± 4.02	88.52 ± 2.69	91.12 ± 2.99	**92.98 ± 1.82**	1.85 ± 2.23
Teaching (150,5,3)	41.91 ± 5.64	48.35 ± 3.80	48.09 ± 2.92	**48.62 ± 3.37**	0.26 ± 4.60
Thyroid (215,5,3)	89.77 ± 2.37	92.43 ± 2.12	92.46 ± 2.49	**94.57 ± 2.08**	2.11 ± 3.10
Waveform (5000,21,3)	70.07 ± 1.30	69.68 ± 1.49	71.42 ± 1.84	**79.64 ± 1.16**	8.22 ± 1.68
Wine (178,13,3)	84.52 ± 2.66	92.22 ± 3.41	89.35 ± 3.71	**94.13 ± 1.78**	1.90 ± 3.37
Zoo (101,16,7)	74.96 ± 5.79	87.75 ± 1.99	89.11 ± 2.58	**92.31 ± 2.15**	3.20 ± 2.95

In order to calibrate the parameters of the different models that regulate the complexity of the tree, we have used different approaches. On the one hand, for CART and OCT, since the maximum number of nodes for such a depth is $2^D - 1 = 7$, one can search for the tree with best complexity by searching in the grid $\left\{1, \dots, 2^D - 1\right\}$ of possible active nodes. For OCT-H, we search the complexity regularization factor in the grid $\left\{10^i : i = -5, \dots, 5\right\}$. Finally, in MOCTSVM we used the same grid $\left\{10^i : i = -5, \dots, 5\right\}$ for c_1 and c_2, and $\left\{10^i : i = -2, \dots, 2\right\}$ for c_3.

The dataset *Waveform*, has 5000 observations and 21 features with 3 different classes. The limitations of the off-the-shelf solvers to certify optimality of MISOCO problems allow one to solve only small to medium size instances up to optimality or with a reasonable MIP Gap within the time limit. Thus, for this dataset we adopt an aggregation strategy which has been successfully applied in other machine learning and facility location problems (see e.g., Blanco et al., 2022a; b). Specifically, for each training sample in the cross validation, we aggregate the observations using the k-means clustering algorithm until the resulting dataset contains around 100 points. Then, we solve the problem on this simplified dataset. The obtained results show that our approach is easily scalable applying this type of aggregation strategies.

In Table 3 we report the results obtained in our experiments for all the models. The first column of the table indicates the identification of the dataset (together with its dimensionality). Second, for each of the methods that we have tested, we report the obtained average test accuracy and the standard deviation. We have highlighted in bold the best average test accuracies obtained for each dataset.

As can be observed, our method clearly outperforms in most of the instances the rest of the methods in terms of accuracy. Clearly, our model is designed to construct Optimal Classification Trees with larger separations between the classes, which results in better accuracies in the test sample. The datasets *Australian* and *BalanceScale* obtain their better results with OCT-H, but, as can be observed, the differences with respect the rest of the methods are tiny (it is the result of correctly classifying in the test sample just a few more observations than the rest of the methods). In that case, our method gets an accuracy almost as good as OCT-H. In the rest of the datasets, our method consistently gets better classifiers and for instance for *Dermatology* the difference with respect to the best classifiers among the others ranges in $[4\%, 19\%]$, for *Parkinson* the accuracy with our model is at least 6% better than the rest, for Wine we get 5% more accuracy than OCTH and 10% more than CART and for Zoo the accuracy of our model is more than 17% greater than the one obtained with CART.

Concerning the variability of our method, the standard deviations reported in Table 3 show that our results are, in average, more stable than the others, with small deviations with respect to the average accuracies. This behaviour differs from the one observed in CART or OCT, where larger deviations are obtained, implying that the accuracies highly depends of the test folder where the method is applied.

In most of the instances that were solved in the training phase by the cross validation scheme, the solutions at the end of the time limit were not guaranteed to be optimal. The MIPGaps that we obtained were large (close to 100%) since this type of problems, with big M (C8) and norm-based constraints (C1) have weak continuous relaxations that usually exhibit a large MIPGap unless the optimality of the solution is certified. Thus, in this case the MIPGap is not an accurate value to measure the quality of the obtained solution and could be even meaningless. For this reason, we have decided not to report it.

6 Conclusions and further research

We have presented in this paper a novel methodology to construct classifiers for multiclass instances by means of a Mathematical Programming model. The proposed method outputs a classification tree were the splits are based on SVM-based hyperplanes. At each branch node of the tree, a binary SVM hyperplane is constructed in which the observations are classified in two fictitious classes (the original classes are ignored in all the splitting nodes), but the global goodness of the tree is measured at the leaf nodes, where misclassification errors are minimized. Also, the model minimizes the complexity of the tree together with the two elements that appear in SVM-approaches: margin separation and distance-based misclassifying errors. We have run an extensive battery of computational experiments that shows that our method outperforms most of the Decision Tree-based methodologies both in accuracy and stability.

Future research lines on this topic include the analysis of nonlinear splits when branching in MOCTSVM, both using kernel tools derived from SVM classifiers or specific families of nonlinear separators. This approach will result into more flexible classifiers able to capture the nonlinear trends of many real-life datasets. Additionally, we also plan to incorporate feature selection in our method in order to construct highly predictive but also more interpretable classification tools.

Author contributions All three authors have contributed evenly to the different aspect of the research and edition of the paper.

Funding Funding for open access publishing: Universidad de Sevilla/CBUA. This research has been partially supported by the Spanish Ministerio de Ciencia y Tecnología, Agencia Estatal de Investigación, and Fondos Europeos de Desarrollo Regional (FEDER) via project PID2020-114594GB-C21, Junta de Andalucía projects FEDER-US-1256951, P18-FR-1422, CEI-3-FQM331, B-FQM-322-UGR20, and AT 21_00032; Fundación BBVA through project NetmeetData: Big Data 2019, UE-NextGenerationEU (ayudas de movilidad para la recualificación del profesorado universitario), and the IMAG-Maria de Maeztu grant CEX2020-001105-M /AEI /10.13039/501100011033.

Data availability All data are publicly available at UCI repository.

Code availability All codes are available upon request to the authors.

Declarations

Conflict of interest Not applicable.

Ethics approval Not applicable.

Consent to participate All the authors consent to participate and to appear in the list of authors.

Consent for publication Not applicable.

References

Agarwal, N., Balasubramanian, V. N., & Jawahar, C. (2018). Improving multiclass classification by deep networks using DAGSVM and triplet loss. *Pattern Recognition Letters, 112*, 184–190.

Allwein, E. L., Schapire, R. E., & Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research, 1*, 113–141.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Bahlmann, C., Haasdonk, B., & Burkhardt, H. (2002). Online handwriting recognition with support vector machines-a kernel approach. In *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition* (pp. 49–54). IEEE.

Baldomero-Naranjo, M., Martínez-Merino, L. I., & Rodríguez-Chía, A. M. (2020). Tightening big Ms in integer programming formulations for support vector machines with ramp loss. *European Journal of Operational Research, 286*(1), 84–100.

Baldomero-Naranjo, M., Martínez-Merino, L. I., & Rodríguez-Chía, A. M. (2021). A robust SVM-based approach with feature selection and outliers detection for classification problems. *Expert Systems with Applications, 178*(115), 017.

Benítez-Peña, S., Blanquero, R., Carrizosa, E., et al. (2019). Cost-sensitive feature selection for support vector machines. *Computers & Operations Research, 106*, 169–178.

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning, 106*(7), 1039–1082.

Bertsimas, D., & Dunn, J. W. (2019). *Machine learning under a modern optimization lens*. Dynamic Ideas LLC.

Blanco, V., Japón, A., & Puerto, J. (2020). Optimal arrangements of hyperplanes for SVM-based multiclass classification. *Advances in Data Analysis and Classification, 14*(1), 175–199.

Blanco, V., Puerto, J., & Rodriguez-Chia, A. M. (2020). On lp-support vector machines and multidimensional kernels. *The Journal of Machine Learning Research, 21*, 469–497.

Blanco, V., Gázquez, R., & Ponce, D., et al (2022a) A branch-and-price approach for the continuous multifacility monotone ordered median problem. *European Journal of Operational Research*.

Blanco, V., Japón, A., & Puerto, J. (2022). A mathematical programming approach to binary supervised classification with label noise. *Computers & Industrial Engineering, 172A*(108), 611.

Blanco, V., Japón, A., & Puerto, J. (2022). Robust optimal classification trees under noisy labels. *Advances in Data Analysis and Classification, 16*, 155–179.

Blanquero, R., Carrizosa, E., Molero-Río, C., et al. (2021). Optimal randomized classification trees. *Computers & Operations Research, 132*(105), 281.

Breiman, L., Friedman, J., & Olshen, R. et al. (1984). Classification and regression trees.

Carreira-Perpiñán, MA., & Tavallali, P. (2018). Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (pp. 1219–1229). Curran Associates Inc., Red Hook, NY, USA, NIPS'18.

Carrizosa, E., Molero-Río, C., & Morales, D. R. (2021). Mathematical optimization in classification and regression trees. *Top, 29*(1), 5–33.

Chen, G., & Ge, Z. (2019). SVM-tree and SVM-forest algorithms for imbalanced fault classification in industrial processes. *IFAC Journal of Systems and Control, 8*(100), 052. https://doi.org/10.1016/j.ifacsc.2019.100052

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

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21–27.

Crammer, K., & Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research, 2*, 265–292.

Demirović, E., & Stuckey, PJ. (2021). Optimal decision trees for nonlinear metrics. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 3733–3741).

Demirović, E., Lukina, A., Hebrard, E., et al. (2022). Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research, 23*(26), 1–47.

Dietterich, T. G., & Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research, 2*, 263–286.

Du, M., Liu, N., & Hu, X. (2019). Techniques for interpretable machine learning. *Communications of the ACM, 63*(1), 68–77.

Eitrich, T., & Lang, B. (2006). Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of Computational and Applied Mathematics, 196*(2), 425–436.

Firat, M., Crognier, G., Gabor, A. F., et al. (2020). Column generation based heuristic for learning classification trees. *Computers & Operations Research, 116*(104), 866.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning.

Gan, J., Li, J., & Xie, Y. (2021). Robust SVM for cost-sensitive learning. *Neural Processing Letters*, 1–22

Gaudioso, M., Gorgone, E., Labbé, M., et al. (2017). Lagrangian relaxation for SVM feature selection. *Computers & Operations Research, 87*, 137–145.

Guermeur, Y., & Monfrini, E. (2011). A quadratic loss multi-class SVM for which a radius-margin bound applies. *Informatica, 22*(1), 73–96.

Günlük, O., Kalagnanam, J., Menickelly, M. et al. (2018). Optimal decision trees for categorical data via integer programming. arXiv preprint arXiv:1612.03225.

Harris, T. (2013). Quantitative credit risk assessment using support vector machines: Broad versus narrow default definitions. *Expert Systems with Applications, 40*(11), 4404–4413.

Hastie, TJ., & Tibshirani, RJ. (2017). Generalized additive models.

Hu, H., Siala, M., & Hebrard, E. et al. (2020). Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*.

Hu, X., Rudin, C., & Seltzer, M. (2019). Optimal sparse decision trees. *Advances in Neural Information Processing Systems*, *32*.

Jiménez-Cordero, A., Morales, J. M., & Pineda, S. (2021). A novel embedded min-max approach for feature selection in nonlinear support vector machine classification. *European Journal of Operational Research, 293*(1), 24–35.

Kašćelan, V., Kašćelan, L., & Novović Burić, M. (2016). A nonparametric data mining approach for risk prediction in car insurance: A case study from the Montenegrin market. *Economic Research-Ekonomska istraživanja, 29*(1), 545–558.

Lauer, F., & Guermeur, Y. (2011). MSVMpack: a multi-class support vector machine package. *The Journal of Machine Learning Research, 12*, 2293–2296.

Lee, Y., Lin, Y., & Wahba, G. (2004). Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association, 99*(465), 67–81.

Letham, B., Rudin, C., McCormick, T. H., et al. (2015). Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics, 9*(3), 1350–1371.

Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning* (pp. 4–15). Springer.

Lin, J., Zhong, C., & Hu, D. et al. (2020). Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning* (pp. 6150–6160). PMLR.

Majid, A., Ali, S., Iqbal, M., et al. (2014). Prediction of human breast and colon cancers from imbalanced data using nearest neighbor and support vector machines. *Computer Methods and Programs in Biomedicine, 113*(3), 792–808.

Meyer, D., Dimitriadou, E., & Hornik, K. et al. (2015). Misc functions of the department of statistics, probability theory group (formerly: E1071). Package e1071 TU Wien.

Montañana, R., Gámez, J. A., & Puerta, J. M., et al. (2021). Stree: A single multi-class oblique decision tree based on support vector machines. In E. Alba, G. Luque, & F. Chicano (Eds.), *Advances in Artificial Intelligence* (pp. 54–64). Cham: Springer International Publishing.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research, 2*(1), 1–32.

Narodytska, N., Ignatiev, A., & Pereira, F. et al. (2018). Learning optimal decision trees with SAT. In *Ijcai* (pp. 1362–1368).

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research, 12*, 2825–2830.

Quinlan, J. (1996). *Machine learning and id3*. Morgan Kauffman.

Quinlan, R. (1993). *C4. 5: Programs for machine learning*. Elsevier.

Radhimeenakshi, S.: (2016) Classification and prediction of heart disease risk using data mining techniques of support vector machine and artificial neural network. In: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), IEEE, pp 3107–3111

Tang, X., & Xu, A. (2016). Multi-class classification using kernel density estimation on k-nearest neighbours. *Electronics Letters, 52*(8), 600–602.

van den Burg, G., & Groenen, P. (2016). GenSVM: A generalized multiclass support vector machine. *Journal of Machine Learning Research, 17*, 1–42.

Verhaeghe, H., Nijssen, S., Pesant, G., et al. (2020). Learning optimal decision trees using constraint programming. *Constraints, 25*(3), 226–250.

Verwer, S., & Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence* (pp. 1625–1632).

Weston, J., & Watkins, C. (1999). Support vector machines for multi-class pattern recognition. In *Esann* (pp. 219–224).

Yu, J., Ignatiev, A., & Stuckey, PJ. et al. (2020). Computing optimal decision sets with SAT. In *International Conference on Principles and Practice of Constraint Programming* (pp. 952–970). Springer.