Inducing Readable Oblique Decision Trees

1st Antonin Leroux
craft ai
Paris, France
antonin.leroux@craft.ai

2nd Matthieu Boussard
craft ai
Paris, France
matthieu.boussard@craft.ai

3rd Rémi Dès
craft ai
Paris, France
remi@craft.ai

***Index Terms*—Oblique Decision Tree, Decision trees, Explainable AI, Linear discriminant analysis, Machine Learning**

***Abstract*—Although machine learning models are found in more and more practical applications, stakeholders can be suspicious about the fact that they are not hard-coded and fully specified. To foster trust, it is crucial to provide models whose predictions are explainable. Decision Trees can be understood by humans if they are simple enough, but they suffer in accuracy when compared to other common machine learning methods. Oblique Decision Trees can provide better accuracy and smaller trees, but their decision rules are more complex. This article presents MUST (Multivariate Understandable Statistical Tree), an Oblique Decision Tree split algorithm based on Linear Discriminant Analysis that aims to preserve explainability by limiting the number of variables that appear in decision rules.**

I. Introduction

A Decision Tree (DT) classifies the data by partitioning the feature space into sub-spaces. In the final tree structure all sub-spaces correspond to a leaf of the tree. Each tree node recursively splits the data into sub-spaces following a decision rule. Each leaf is then labeled to predict the class of the samples in the corresponding sub-space.

The most popular DT generation algorithms like CART [1], C4.5 [2] or OC1 [3] generate a top-down DT by applying recursively a splitting function on the data. At each node the splitting function is applied on the remaining data to find the optimal decision rule. The algorithm stops when the output in the reached sub-space is pure or almost pure. The main differences between algorithms appear on the splitting method.

Splits can be binary, as in CART or QUEST [4], or non-binary as in C4.5 and FACT [5]. In all the following a *numerical* feature is a variable taking its values in $\mathbb{R}$ while a *categorical* feature has discrete and unordered values. When splitting on numerical features, decisions rules generally come in the form $x > c$or $\leq c$, where x is a numerical feature value from each sample and c is determined by the learning algorithm during the learning step. This binary linear boundary can be formulated as a hyperplane of the feature space and splits the samples on both sides (CART, C4.5, QUEST).

In the linear case, there are two kinds of algorithms. CART and C4.5 focus on axis parallel or univariate split, while QUEST or CRUISE [6] extend the search to oblique or multivariate splits. An axis-parallel split splits on one single feature value; it produces a hyperplane orthogonal to one axis while being parallel to all the others.

The parallel method was the first to be thoroughly explored. In this case an extensive search through all potential splits is actually possible (see Sec.II) and this guarantees that we find the optimal split. Furthermore, univariate splits are easily understandable and following the decision rules from the root to the leaves easily gives a good way to see how each feature contributes to the classification while giving a sense of order between all the different features.

In general, multivariate splits algorithms can achieve similar accuracy to parallel ones while producing smaller trees (see Sec.IV). Size is crucial whenever a human has to analyze the learned the decision tree, as larger trees can become prohibitively more difficult to grasp. However the naive search of all oblique splits is NP-Complete [7], making it intractable for large datasets. The use of heuristics is necessary to induce oblique decision trees.

If we want to produce an easily explainable DT, i.e. a DT that a human can understand, it is critical to have a reduced number of nodes and leaves. So even though oblique boundaries are harder to understand than simple univariate split, it is interesting to study the tradeoff between the gain in size and this split function complexity. When speaking about the size of a tree, we will always refer to the number of leaves.

To reach a better accuracy, [8], [9] choose non-linear decision rules. NDT [8] propose to build a linear boundary in an extended feature space. Extra features are computed non-linearly from the original ones (for instance multiplication of two original features). Extending the Omnivariate decision trees ideas from [10], the approach proposed in [9] can have both linear and non-linear decision rules. A choice between the two is made at each node with a classifiability measure. But non-linear models are often much more difficult to understand as their decision rules can take complicated forms. They also have a tendency to overfit the data and show limited results when confronted to noise. The computational cost is increased as well. In [8] for instance, the additional features add two additional costs: the computation of the new attributes and the extra cost brought by a higher number of features.

If the data have categorical values, CART and C4.5 treat these values differently from the numerical features. To preserve binary splits, CART's decision rules are in the form $x \in A$ or $x \notin A$, while C4.5's are of the type $x = a$. QUEST and CRUISE transform features into numerical features using

Disciminant coords. It seems to be the only way to avoid the selection bias caused by having different types of features in the same tree [4]. However, like any method that transforms original features, this choice damages the understandability of the produced trees.

In this work, we present a split algorithm based on Linear discriminant analysis (LDA), which is an efficient method to compute oblique splits. The contributions of this article are:

1) A feature selection algorithm that helps to produce simpler decision rules,
2) for the single feature split case, a quick estimation algorithm that performs an univariate split search,
3) a full oblique split algorithm that combines all of these elements.

The goal of this paper is to produce easily explainable tree. We are using some of the dimensions of interpretability [11] to define what is explainable. We want a user to be able to understand the logic of the whole generated model (*global interpretability*). The background and the expertise of the users has a direct impact on their perception of a model (*Nature of user expertise*); we consider this user a domain expert, but not an ML expert. Thus, a combination variables may be easier to read for an expert than a succession of splits. Also, we limit the number of variables in an oblique split to 3 to be able to make a spatial visualization. Finally we select the number of nodes in the tree as a complexity measure.

The remaining Sections of this paper are organized as follows: Section II presents related work. Section III explains the proposed methods. The results and comparison with other algorithms are presented in Section IV, Section V concludes the paper.

II. Related Work

Like this work, others also aim at making a tradeoff between explicability, accuracy and computational efficiency in order to provide algorithms usable in practice. This paper focuses on decision trees, but one can note that this approach is not limited to trees. This is the case for instance in [12], where the authors build probabilistic rule lists that are sequences of IF-THEN rules.

In the following, we will denote by n the number of samples and by d the number of features.

This paper deals with finding new splitting methods, thus this section gives more details about related split algorithms. Even though non-linear split functions do exist, we will focus in the rest of this paper only on linear splits.

The best split is defined as a split maximizing a gain function. The aim is to find the global maximum of the function mapping all the possible splits with the gain. In the parallel axis split case the number of split is only nd. That is why CART and C4.5 can perform an exhaustive search of all splits in $O(dn\log(n))$. When extending the search to oblique hyperplanes the number rises critically. In [13] an upper bound of $2^d\binom{n}{d}$ is given for the number of splits.

As mentioned in Section **I**, we need strategies to perform the search in a reasonable time. There are two categories of algorithms performing this search: Optimization Algorithms (II-A) and Heuristic-based Algorithms (II-B).

A. Optimization Algorithms

The algorithms presented in this section use optimization to recursively converge to the optimal solution. They are based on gradient descent-like techniques. The gain function mentioned below is computed with the repartitions of all the classes in the the different subtrees.

The first major oblique tree was CART-LC, Breiman's CART extension to oblique splits [1]. This algorithm is based on recursive perturbation of hyperplanes to reach a maximum of the gain. Using hill climbing technique to find the best oblique split, CART-LC often gets stuck in local maximums of the gain function.

SADT algorithm was then introduced [14]. Instead of moving the initial hyperplane toward the local maximum, the SADT algorithm uses random jumps to find the best split. This choice proves to be quite good at avoiding local extrema but requires to try a large number of random hyperplanes before finding a suitable choice.

The OC1 algorithm [3] mixed ideas from both CART-LC and SADT. Once a local extremum is reached, random jumps are used to get out of it. It remains one of the major references in terms of performance for oblique decision trees.

In [15], evolutionary and genetic algorithms are used to boost the performance of OC1. This improves accuracy but does not bring a better asymptotic complexity.

The downside of all the above algorithms is time complexity. They perform the search in $O(dn^2log(n))$ at each node.

B. Heuristic-based algorithms

The algorithms presented below rely on a heuristic to find the best hyperplane. In [4]–[6], [16] a statistical approach is used, while [17]–[19] are using the global geometrical structure of the data to find the best split.

The HHCART [17] algorithm uses the classes covariance matrix eigenvectors to find a general direction for the splitting hyperplane. Then an axis-parallel split research is performed on the reflected space of the data with a Householder matrix.

The binary tree algorithm of [18] relies at each node on an SVM classifier to find two clustering hyperplanes. These hyperplanes are chosen to maximize the distance to one class while minimizing the distance to the other class. The original SVM algorithm requires to solve a convex quadratic problem, so other algorithms follow similar ideas with less constraints such as Geometric decision tree [19] to reach a lesser computational complexity.

Statistical-focused approaches often use Linear Discriminant Analysis (LDA, see Sec.III-B), such as QUEST [4], FACT [5], CRUISE [6] or Fisher's Tree [16].

The Fisher's Tree algorithm shares similarities with the proposed algorithm in its general principles (Sec.III). We go further as we propose an algorithm able to deal also with categorical feature and we add a feature selection algorithm in order to improve the understandability of the resulting classifier.

III. MUST: Multivariate Understandable Statistical Tree

This section presents the proposed algorithm, called MUST (Multivariate Understandable Statistical Tree). The following explanations are only addressing the 2-class case; Sec.III-H will give further details on the adaptation of the algorithm to the multi-classing problem. In subsection Sec.III-B we explain the algorithm that performs the split search for numerical features.

The proposed algorithm can also deal with categorical attributes. The CRIMCOORD transformation [4], [6], [17] that creates numerical features from categorical attributes shows good results. However, it is quite complicated to interpret an hyperplane that uses purely artificial features. This is why in this paper we will treat categorical values like in [2]. At each non terminal node we compare a binary split on the continuous features against a possibly non binary split on categorical values. This helps the user understand the resulting decision tree.

A. Definitions and Notations

We note n the number of samples, d the number of features and K the number of classes (in the following sections, $K = 2$) in dataset C.

A sample x can be defined as a point in $\mathbb{R}^d$ plus a list of categorical values and the class of the sample. Usually when we use the x notation it refers to the vector of numerical feature values $x = \{x_1, \ldots, x_d\}$. The two classes are labeled C_+ and C_-.

The means of the samples in each class are respectively noted m_+ and m_-. The difference between the two means is:

$$\Delta_m = m_+ - m_- \tag{1}$$

Δ_m represents the average direction and distance between the classes, but does not provide information on how their samples are distributed.

For a set C of continuous samples x, the scatter matrix Σ is defined as follows:

$$\Sigma = \sum_{x \in C} (x - m)(x - m)^T \tag{2}$$

Where m is the mean of the samples. The scatter matrix gives an indication of the directions along which the samples are aligned the most. Σ_+ and Σ_- are the scatter matrices for samples of class C_+ and C_- respectively. We could obtain a covariance matrix by normalizing by the number of samples. Multiplyting factors on scatter matrixes are of no consequence in this work so we generally use the simplest form.

As we said earlier, for a binary oblique split we need a splitting hyperplane. A hyperplane can be defined using two distinct elements. First, the orientation of the hyperplane that is given by a normal vector. We will note this normal vector w. Once we have this direction we need to decide the hyperplane's position in the feature space. For that we will need a another value noted w_0. The equation of the hyperplane is then: $w^T x = w_0$. The resulting decision rule can be written as:

$$w^T x > w_0 \quad \text{or} \quad w^T x \leq w_0 \tag{3}$$

The next section presents MUST's procedure to compute w.

B. LDA

The algorithm to find w is adapted from Linear Discriminant Analysis (LDA), which gives a good heuristic to find a splitting hyperplane. The first steps of the algorithm follow the same pinciples as in Fisher's Tree [16]. According to LDA w is defined as:

$$w = \text{argmax}_w \frac{w^T \Sigma_b w}{w^T \Sigma_w w} \tag{4}$$

where $\Sigma_b = \Delta_m \Delta_m^T$ is called the between-class scatter matrix and $\Sigma_w = \Sigma_+ + \Sigma_-$ is called the total within-class scatter matrix.

This formula comes from the idea that the projections of the 2 classes on the line defined by w need to be as far from each other as possible (maximize between-class scatter) while being less scattered (minimize within-class scatter).

This formula can be differentiated with respect to w to give the following solution:

$$w = \Sigma_w^{-1} \Delta_m \tag{5}$$

The problem with LDA is that it requires the within-class scatter matrix Σ_w to be invertible. We will see in Section **III-F** a way to solve this problem.

Once the normal vector is found, we will use a CART-like procedure to determine w_0. Indeed LDA gives a good heuristic to select a normal vector among the numerous possibilities. Then, searching thoroughly for the optimal w_0 will help improve precision while only adding a little computational complexity.

C. Gain function

The algorithm requires a gain function that can handle both binary and non-binary splits. Due to certain bias [2]'s C4.5 uses an adjusted gain ratio based on entropy and a split information coefficient. But this adjustment also suffers from certain bias (see [20]). That is why we use the [20]'s balanced gain ratio as our gain function preset.

D. Feature selection

Since the goal of this paper is to create easily explainable decision trees, we want to make oblique splits easier to interpret. Indeed, if there are 10 numerical features for example, a person would find it difficult to comprehend the intuitive meaning of a tree's decision rule described as a linear relation between 10 features. That is why we decided to perform a feature selection to reduce the number of features used for the split. We will try to reduce to 1, 2 or 3 features in order to be able to plot the splitting hyperplanes and the samples if needed, and to keep the number of features small enough that the decision rules can be grasped.

We need to pick the features that contribute the most to the hyperplane proposed by the LDA method. From here, s will be the number of features selected.

After computing w with the formula described in Eq.5, we select the s features with the maximum contribution to the

score $\frac{w^T \Sigma_b w}{w^T \Sigma_w w}$ (Eq.4). We evaluate the contribution $contrib$ of feature i with the formula:

$$contrib_i = \frac{|w_i \Delta_{m_i}|^2}{\Sigma_{w_{i,i}} w_i^2} \tag{6}$$

following the exact same heuristic where only the ith component of w is not null.

With Eq.6 we select the s features with the highest contribution. Then we compute the new w using Eq.5 on the feature space reduced to just those s features. The extra computation cost is very low as we have already computed all the coefficients of the means and scatter matrices.

Besides helping the user to understand the model, the feature selection process can help to remove features that do not actually contributes to the classification. Such features are only adding noise and cause a loss of accuracy in the prediction. We see in the Sec.IV-B that this feature selection process can even produce trees that perform better.

Finally, as we use very small values of s, the feature selection has another interesting property as it reduces the number of times where the small samples problem (described in Sec.III-F) occurs.

Even with a limited number of attributes involved in the linear boundary, a multivariate decision rule remains harder to understand than a univariate one. Thus we will handle the particular case $s = 1$ in Seq.III-E. Moreover as we will see in Seq.IV, axis parallel splits algorithms can sometimes be more efficient and accurate.

In the specific case where we want to select one single feature (and perform a parallel-axis split), the LDA heuristic is of no use as it may be actually quicker to test all possible splits. As described in Sec.III-I the complexity of LDA is nd^2 and the complexity of univariate splits is $dnlog(n)$. However with a way to estimate quickly and quite reliably the potential of each feature, we can perform a feature selection faster and more accurately than the one provided by our algorithm, providing a good alternative to the naive univatiate methode.

E. Parallel split fast estimator

In this section we propose a method to perform a quick search for an axis-parallel split. In the case where $s = 1$ and there are two classes, this faster algorithm replaces the feature selection algorithm that we described in Sec.III-D. This method uses a heuristic to find a splitting value that separates the classes well, instead of computing the gain function for every possible candidate. It differs from binning techniques found XGBoost [21] or LightGBM [22] since those techniques approximates various split position for every variable (the bins), where here, we propose to directly find a single estimation of the best split value fo every variable.

We are looking for a splitting hyperplane that is orthogonal to a feature axis. First, we will look at each feature $F_i, i \in \{0, \ldots, d\}$ and consider w with exactly one non-null component $w_i = 1$. We will choose a splitting value w_{0_i} that best separates the classes along this feature. The considered decision rule is: $x_i > w_{0_i}$ or $x_i \leq w_{0_i}$.

In the following, m_{+_i} and m_{-_i} are the means on feature F_i of the samples in classes C_+ and C_- respectively. v_{+_i} and v_{-_i} are the variances for the respective classes on feature F_i.

As a working approximation, we consider that the bulk of the samples of class C_+ (respectively C_-) have their values on F_i in the interval $[m_+ - v_+, m_+ + v_+]$ (respectively $[m_- - v_-, m_- + v_-]$).

We use the following notations to indicate soft borders of the classes on F_i:

$$p_{sup_i} = \begin{cases} m_{+_i} - v_{+_i} & \text{if } m_{+_i} > m_{-_i} \\ m_{-_i} - v_{+_i} & \text{otherwise} \end{cases} \tag{7}$$

$$p_{inf_i} = \begin{cases} m_{-_i} + v_{-_i} & \text{if } m_{+_i} > m_{-_i} \\ m_{+_i} + v_{+_i} & \text{otherwise} \end{cases} \tag{8}$$

And we define p_{edge_i} as the edge of the class with the smallest variance, on the side of the other class:

$$\begin{cases} p_{edge_i} = m_{-_i} + v_{-_i} & \text{if} \quad v_{+_i} > v_{-_i}, m_{+_i} > m_{-_i} \\ p_{edge_i} = m_{-_i} - v_{-_i} & \text{if} \quad v_{+_i} > v_{-_i}, m_{+_i} \leq m_{-_i} \\ p_{edge_i} = m_{+_i} - v_{+_i} & \text{if} \quad v_{+_i} \leq v_{-_i}, m_{+_i} > m_{-_i} \\ p_{edge_i} = m_{+_i} + v_{+_i} & \text{if} \quad v_{+_i} \leq v_{-_i}, m_{+_i} \leq m_{-_i} \end{cases} \tag{9}$$

The formula for w_{0_i} is:

$$w_{0_i} = \begin{cases} \frac{p_{sup_i} + p_{inf_i}}{2} & \text{when} \quad p_{sup_i} > p_{inf_i} \\ p_{edge_i} & \text{otherwise} \end{cases} \tag{10}$$

This comes from the observation that when the two classes are almost separated, a good purity will come from a separation in the middle (see III-E). However if the two classes overlap much then the best purity gain will come from a split that isolates a very pure part of one class while having a balanced or relatively pure other part (III-E). For each feature F_i, we

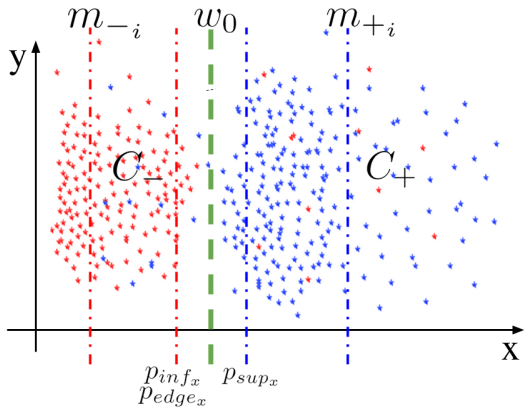

Fig. 1. Parallel split estimator: well separated classes.

found a splitting value w_{0_i}. Then, we compute the associated gain Γ_i to select the feature F_u with the highest gain. The gain is computed using a typical purity metric such as information gain.

$$u = \underset{i \in \{0,\ldots,d\}}{\text{argmax}} \quad \Gamma_i \tag{11}$$

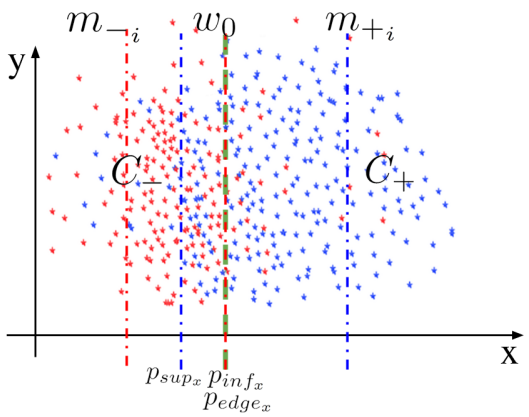

Fig. 2. Parallel split estimator: overlapping classes.

So we find the splitting feature F_u that best maximizes the gain with its splitting value w_{0_u}. To go back to the hyperplane notation, w is such that $w_u = 1$ and $w_i = 0$ for $i \in \{0, \ldots, d\}$, $i \neq u$. w_0 is such that w_{0_u} is as above, and $w_{0_i} = 0$ for $i \in \{0, \ldots, d\}$, $i \neq u$.

F. Small Samples

As pointed out in the end of Sec.III-B, the LDA method requires the Σ_w matrix to be invertible. It is not invertible if the dimensionality of the samples is less than d. This happens if features or samples are linearly dependent, and is sure to arise in the small sample case when $n < d$.

In this case we perform a search for the optimal w following ideas described in [23]. Using the fact that $\frac{f(x)}{g(x)}$ and $\frac{f(x)}{g(x)+f(x)}$ reach their maximum for the same point x_0, [23] have proved that when Σ_w does not have a maximal rank then the w given by Eq.4 is in $E_0 = \ker \Sigma_w$. So we need to compute the vector w in E_0 that maximizes $w^T \Sigma_b w$. To do that we project the samples on E_0 using the projection matrix QQ^T. With $d_0 =$dim E_0, we define $Q = [\alpha, \ldots, \alpha_{d_0}]$ a matrix of size (d, d_0), the α_i being the spanning vectors of E_0. Then w is made from the s eigenvectors with the largest eigenvalue of the adjusted matrix $QQ^T \Sigma_b (QQ^T)^T$.

This solution works every time Σ_w is not invertible and is not just limited to the small sample case.

G. The MUST Algorithm

We call MUST the algorithm that computes the split at each iteration during the generation of a decision tree. A full tree generation based on MUST can follow the steps of any iterative decision tree generation algorithm with a gain function. Alg.1 presents the oblique split algorithm, using an existing gain computation function $gain$. Alg.2 explains the global splitting algorithms that is performed at each node to find the decision rule. Alg.3 and Alg.4 respectively introduce the feature selection process performed during Alg.1 for any value of s (different from or equal to 1).

Input: data C, number for feature selection s
Output: hyperplane (w, w_0), gain
Initialize $m_+, m_-, \Sigma_+, \Sigma_-$ with Eq.2.
$\Sigma_w = \Sigma_+ + \Sigma_-$
if Σ_w is invertible **then**
  Compute w with (5)
else
  Compute w using the small samples (Sec.III-F)
end if
if $s = 1$ **then**
  Compute w with one feature using Alg.4
else
  Compute w with selected feature using Alg.3
end if
Let C_w be $\left\{w^T x | x \in C\right\}$ sorted
Let $C_{w_0} = \left\{\frac{p_i + p_{i+1}}{2} | p_i, p_{i+1} \in C_w, 0 \leqslant i < n\right\}$
Let $w_0 = \underset{w_0 \in C_{w_0}}{\text{argmax}}\ \text{gain}(w_0, C)$
return (w, w_0, gain)

Algorithm 1: Oblique Split Algorithm

H. Adjustment for multi-class problems

The algorithm as we described it cannot be used if there are more than 2 classes. In this case, the original LDA algorithm uses a different between-class scatter matrix Σ_b. With m the total mean of C and $m_k k \in \{0, \ldots, K\}$ the means of the K classes, we have:

$$\Sigma_b = \sum_{k=1}^{K} (m_k - m)(m_k - m)^T \tag{12}$$

This formula does not bring any additional complexity compared with the 2-class problem and would be a first option to extend the 2-class case. This is used in [4]'s QUEST.

The second option would be to use a clustering algorithm like 2-means to separate the samples into two groups, on which to apply the 2-class method. A cheap way is to apply clustering on the means of the different classes. A more expensive but more accurate method is to apply clustering on all the samples. Then we can isolate two groups of classes based on the repartitions of

Input: data C= $\{(x, x_c, y)\}$
Output: Decision rule D
Initialize $H = (w, w_0)$, gain given by Alg.1
Let D be the binary decision rule from H
for categorical feature F_c **do**
  Let gain$_c$ = gain(F_c, C)
  if gain$_c$ $>$ $gain$ **then**
    gain=gain$_c$
    D becomes the decision rule defined a split on F_c
  end if
end for
return D

Algorithm 2: Split Algorithm

Input: w, per-class means difference Δm, scatter Σ_w
Output:new hyperplane w_s
Let $V = \{f(i, w, \Sigma_w, \Delta m) | 0 \leq i < d\}$
where f is defined in (6)
Let V_s = {indexes of the s biggest elements in V}
Let Σ_{w_s}, Δm_s be the scatter and mean difference in the reduced space defined by V_s
Compute w_s with (5)
return w_s
Algorithm 3: Feature Selection Algorithm

Input: means m_+, m_-, scatter matrices Σ_+, Σ_-, data C
Output: Univariate split hyperplane w_u
Initialize gain=0, i_{max}=0
for $0 \leq i \leq d$ **do**
  Let w_0^i be computed with (10)
  Let gain$_i$ = gain(F_i, w_{0_i}, C)
  if gain$_i >$ gain **then**
    gain=gain$_i$ and $i_{max} = i$
  end if
end for
Let w_u be the vector with 1 in the i_{max}-th component and 0 everywhere else
return w_u
Algorithm 4: Quick Estimation Algorithm

the classes in the two cluster. Bringing the multi-class problem back to a two class problem is quite common [17]–[19].

All propositions take an asymptotic nd times, but the step involving all samples is much longer than the others. Still, considering the potential improvement in accuracy, we opted for the last choice: use clustering on the samples to go back to the 2-class problem.

I. Complexity analysis

In this subsection we will give the complexity cost of our splitting algorithm, performed at each non-terminal node. The global complexity of generating full trees is difficult to estimate in general depending on the other steps, but efficient heuristics often build smaller trees that are globally quicker to compute. This part will be empirically evaluated in Sec.IV with the tree size statistics.

The complexity of LDA is very advantageous when the number of samples is large compared to the number of features. Indeed, the main cost is the computation of the scatter matrices which is made in $O(nd^2)$. The means are calculated in $O(nd)$.

In our algorithm, we add $O(nlog(n) + dn)$ which is the computation cost of finding the optimal w_0 with information gain or a similar function as purity function: $O(dn)$ to project the samples on w, $O(nlog(n))$ to sort the n projections, and then another $O(n)$ to find the optimal w_0 by comparing gains. This last complexity can be achieved by going through the sorted table of projected sample values while keeping in memory the number of each class right and left of the considered hyperplane. As the information gain can be entirely computed from these numbers, by doing it in the right order we avoid recomputing the gain function with a n cost complexity for each split.

In the multi-class case, if we denote K the number of classes, we have a Kn complexity to find w_0.

Thus we have an asymptotic $O(nd^2 + nlog(n) + Kn)$ global complexity to find the oblique split at each node.

We need to add the cost to perform the computation of the splitting gain on categorical attributes as in C4.5. If we note d_c the number of categorical feature at the current node, the cost of this step is $d_c n$. Let's note that once a split is done on a categorical feature, that feature will not need to be considered for subsequent splits.

It is also relevant to note that considering the d^2 component of the global complexity, it is quite important to avoid making d large by adding additional numerical features derived from quantitative attributes.

For the quick estimation of univariate splits, assuming that each gain estimation Γ_i takes $O(n)$ operations, the whole process costs $O(dn)$. Once the split feature is selected we need $O(nlog(n))$ (see above) to find w_0. The total quick estimation process takes a total $O(dn + nlog(n))$. The naive algorithm that explores all univariate splits can be made in $O(dnlog(n))$.

TABLE I
REAL CONTINUOUS DATASETS DOWNLOADED FROM UCI.

DATA SET	$d(d_c)$	K	n
HEART	13(0)	2	270
INDIAN DIABETES (PIMA)	8(0)	2	768
GLASS	9(0)	6	214
WINE	13(0)	3	178
SURVIVAL	3(0)	2	306
LETTER	16(0)	26	20000
LIVERS (BUPA)	6(0)	2	345
BALANCE SCALE	4(0)	3	625
STATLOG	14(8)	2	690
INCOME	14(8)	2	32561
BANK	16(9)	2	45211

IV. RESULTS

In this section we present empirical results to evaluate the performance of MUST. We generated trees following a method similar to C4.5 [2] using MUST splits. We tested our algorithm with real data sets coming from the UCI repository [24]. All estimations are made with 10 5-fold cross validations to estimate the average size and accuracy of the produced tree.

Tab.I shows the values of d (number of features), n (number of samples) and K (number of classes) for the different data sets used. d_c is the number of categorical attributes.

A. Comparison Full parallel Search vs Quick Estimation

Tab.II confronts the results obtained by the Quick Estimation algorithm (QE) to an algorithm that perform a full search (FS) for parallel split. That FS algorithm is perfectly similar to the QE we introduced in Sec.III-E except that it tests all possible

TABLE II
ACCURACIES AND TREE SIZE FOR QE VS FS (PARALLEL SPLIT)

DATA SET	DT	AVG. ACC.	AVG. SIZE
HEART	QE	**82.0**± 2.4	5.6± 0.2
	FS	74.0± 1.9	5.9± 2.0
LETTER	QE	85.9± 0.1	1022.8± 132.8
	FS	**86.8**± 0.00	924.0± 71.6
GLASS	QE	**68.9** ± 4.5	13.0± 0.3
	FS	68.0± 5.4	15.3± 0.8
WINE	QE	**93.8** ± 3.2	5.4± 0.1
	FS	91.9± 4.7	5.3± 0.1
SURVIVAL	QE	72.9± 1.5	3.5± 0.5
	FS	**73.3** ± 1.8	3.7± 0.8
PIMA	QE	**74.6**± 0.5	6.9± 0.3
	FS	74.4± 0.6	9.9± 0.6
BUPA	QE	64.7± 3.4	8.4± 2.3
	FS	**65.3**± 2.4	10.1± 0.7

TABLE III
ACCURACIES AND TREE SIZE, PROPOSED VS OTHER METHODS(CATEGORICAL DATASETS)

DATA SET	DT	AVG. ACC.	AVG. SIZE
BANK	QUEST	90.1 ± 0.1	27.0± 15.2
	HHCART	**90.4**± 0.1	44.4± 14.2
	MUST-1	**90.4**± 0.1	117.6±100.0
STATLOG	QUEST	85.7± 0.9	6.1± 3.6
	HHCART	**85.8** ± 0.7	6.5± 3.0
	MUST-2	85.4±0.0	2.0±0.0
INCOME	QUEST	83.9± 0.2	68.0± 23.1
	HHCART	**85.5** ± 0.2	59.5± 19.7
	MUST-1	85.0±0.1	228.0±150.8

univariate split before choosing the one having the best gain. The results are shown in Tab.II.

Our QE algorithm reaches similar performances in term of size and accuracy. As we know that QE is quicker than FS, the results shown in the Tab.II allow us to conclude that our heuristic for parallel split search is efficient.

B. Results for continuous features

Tab.IV shows the performance of our MUST algorithm compared with HHCART and some variations of OC1. The values are directly extracted from [17]. It confronts HHCART results with the OC1 algorithm and OC1-AP (for axis parallel split). The HHCART value is the best from the two HHCART (A and D) algorithms proposed in [17]. MUST-F is the notation for the MUST algorithm without the feature selection process. MUST-s is the MUST algorithm used with the feature selection, s being the number of feature selected. As we said earlier, small values of $1 \leq s \leq 3$ are used. These results are also useful to estimate the efficiency of our feature selection algorithm.

HHCART, OC1 and OC1-AP or assessed with ten 5-fold cross validation. The impurity function is the Twoing Rule [1] and the trees are adjusted with cost-complexity pruning [1] with 10% of the samples. For OC1 the number of restarts and number of jumps were set to 20 and 5.

On datasets that are well classified by parallel splits such as letter we see that MUST-1 reaches an accuracy close to OC1-AP with a significantly smaller tree. But all the other oblique versions of MUST obtain a quite low accuracy compared to HHCART and OC1. This behavior can be explained by the high value of K for letter. Indeed, we use a method that reduces the multi-class problem to a two class problem. The split search is performed on the two artificial clusters of class. Thus, the split generated is naturally less accurate. HHCART does not performs such transformation and consider all classes individually to find the optimal split. It brings a cost in complexity but enables the HHCART algorithm to reach higher accuracy.

Regarding the feature selection we observe that the results depend on the datasets. As we mentioned, in letter only the MUST-1 performs well. Results on Wine, Balance and Bupa are improving as the number of feature rises. On Glass and Survival the tendency is reversed. This can be explained as sometimes there are features that are not relevant to classify the samples. In this case, a full split on all features will be less accurate. On the contrary when all features are useful, ignoring them causes a decrease of the precision.

Even when our feature selection process does not improve the accuracy of MUST-F, it often still achieves to reach accuracy higher than HHCART or OC1. So it can be used to make a good tradeoff between accuracy and explainability.

C. Results for Categorical Attributes

For datasets containing categorical attributes, in Tab.III, we took results from the article [17] to compare MUST with HHCART and QUEST [4]. Considering the fact that the feature selection process is only operating on numerical feature we will only show the results for the best of our MUST algorithms.

In Tab.III we see with Income and Bank datasets that the choice we made about categorical features is problematic as it makes bigger trees. It is not surprising as, when splitting on a categorical attributes, we create as many sub-nodes as attributes values. For instance, if we replace the binary root node by a node inducing four splits we can potentially double the tree size.

The accuracy of the produced trees is in the range of HHCART and QUEST. Thus MUST algorithm achieves to reach an acceptable accuracy while preserving the original form of the categorical features.

V. CONCLUSION

In this paper we presented a way to produce efficient and easily explainable classification decision trees. This new algorithm, called MUST, can deals with both categorical and numerical features. We first proposed an oblique split algorithm and then introduced ways to perform a feature selection. This reduction helps shortening the resulting Decision rule in order to make it more explainable. We also proposed a quick estimation algorithm in the specific case of univariate splits. The different versions of MUST achieves a reasonable

TABLE IV
CLASSIFICATION ACCURACIES AND TREE SIZE PROPOSED VS OTHER METHODS. FOR CONTINUOUS DATASETS

DATA SET	DT	AVG. ACC.	AVG. SIZE
LETTER	MUST-1	85.9± 0.0	1022.8± 132.8
	MUST-2	80.2± 0.3	1134.6± 460.1
	MUST-3	80.8± 0.1	1156.6± 378.3
	MUST-F	80.2 ± 0.0	960.1± 279.8
	HHCART	83.1± 0.3	1135.9± 122
	OC1	83.6± 0.4	1197.2± 88.9
	OC1-AP	**86.3**± 0.3	1611.7±60.0
GLASS	MUST-1	**68.9** ± 4.5	13.0± 0.3
	MUST-2	65.7 ± 1.7	12.9± 1.0
	MUST-3	63.9± 5.3	13.9± 1.7
	MUST-F	57.0 ± 4.2	13.5± 0.5
	HHCART	61.9±3.0	10.1±2.3
	OC1	61.1± 3.5	10.8±4.3
	OC1-AP	64.6± 3.9	14.6±8.7
WINE	MUST-1	93.82± 3.15	5.4± 0.1
	MUST-2	94.9± 0.7	4.5± 0.0
	MUST-3	95.2± 0.4	4.9± 0.1
	MUST-F	**97.4** ± 1.0	3.0± 0.0
	HHCART	91.3± 1.6	3.4± 0.3
	OC1	89.2± 2.1	3.5±0.3
	OC1-AP	89.2± 4.6	4.6±0.6
SURVIVAL	MUST-1	72.9± 1.5	3.5± 0.5
	MUST-2	72.8± 0.9	2.0± 0.0
	MUST-F	72.6 ± 1.8	2.5± 0.1
	HHCART	**73.5**± 1.5	5.3± 2.7
	OC1	71.0± 2.1	6.4±3.5
	OC1-AP	71.9± 1.5	10.7±6.5
PIMA	MUST-1	74.6± 0.5	6.9± 0.3
	MUST-2	75.5± 0.1	2.0± 0.0
	MUST-3	**76.0** ± 0.6	3.2± 0.3
	MUST-F	75.9 ± 0.3	3.1± 0.1
	HHCART	72.9± 1.3	10.8± 4.4
	OC1	73.4± 1.0	9.2±5.4
	OC1-AP	73.6± 1.4	15.9±8.7
BUPA	MUST-1	64.7± 3.4	8.4± 2.3
	MUST-2	67.2± 3.9	11.6± 1.4
	MUST-3	67.3 ± 3.4	12.5± 1.8
	MUST-F	**68.6** ± 2.9	11.8± 4.4
	HHCART	64.1± 2.6	6.5± 1.5
	OC1	66.9± 2.2	8.9±6.1
	OC1-AP	64.7±2.5	13.2±10.5
BALANCE	MUST-1	77.4± 1.3	15.4± 1.8
	MUST-2	83.9± 2.9	14.8± 3.0
	MUST-3	85.6 ± 0.7	15.8± 2.4
	MUST-F	91.2 ± 0.1	3.0± 0.0
	HHCART	**93.7** ± 1.3	7.9± 1.3
	OC1	91.9± 0.9	8.7±3.4
	OC1-AP	78.2±1.3	37.5±16.8
HEART	MUST-1	82.0± 2.4	5.6± 0.2
	MUST-2	79.4± 1.5	4.6± 0.2
	MUST-3	80.0 ± 1.0	4.6± 0.4
	MUST-F	**82.4** ± 1.4	2.0± 0.0
	HHCART	75.8 ± 2.8	7.8± 2.6
	OC1	77.1± 2.5	3.6±1.0
	OC1-AP	76.3 ±2.3	6.7±2.4

computational time, reaching a quasilinear time complexity in the number of samples.

This work demonstrates interesting result for classification problems, so the natural next step is to achieve the same tradeoff between computation time, accuracy and explainability for regression trees so we can provide an unified algorithm for both problems.

REFERENCES

[1] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. New York, NY: Chapman and Hall, 1984.

[2] J. R. Quinlan, "Induction of decision trees." *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[3] S. K. Murthy, S. Kasif, and S. Salzberg, "The oc1 decision tree software system." 1993.

[4] W.-Y. Loh and Y.-S. Shih, "Split selection methods for classification trees." *Statistica sinica*, vol. 7, no. 4, pp. 815–840, 1997.

[5] W.-Y. Loh and N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis." *J. Amer. Statist. Assoc.*, vol. 83, pp. 715–728, 1988.

[6] W.-Y. Loh and H. Kim, "Classification trees with unbiased multiway splits," *J. Amer. Statist. Assoc.*, vol. 96, pp. 598–604, 2001.

[7] L. Hyafil and R. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976.

[8] A. Ittner and M. Schlosser, "Non-linear decision trees-ndt," *ICML*, vol. Citeseer, pp. 252–257, 1996.

[9] Y. Li, M. Dong, and R. Kothari, "Classifiability-based omnivariate decision trees." *IEEE Transactions on Neural Networks*, vol. 16, no. 6, pp. 1547–1560, 2005.

[10] O. Yildiz and E. Alpaydin, "Omnivariate decision trees," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1539–1546, 2001.

[11] R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti, "A survey of methods for explaining black box models," *CoRR*, vol. abs/1802.01933, 2018.

[12] H. Yang, C. Rudin, and M. Seltzer, "Scalable bayesian rule lists," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017, pp. 3921–3930.

[13] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees." *J Artif Intell Res*, vol. 2, no. 1, pp. 1–32, 1994.

[14] D. H. S. K. S. Salzberg, "Induction of oblique decision trees." *IJCAI*, pp. 1002–1007, 1993.

[15] E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Transaction on Evolutionary Computation*, vol. 7, no. 1, pp. 54–68, 2003.

[16] A. Lopez-Chau, J. Cervantes, L. Lopez-Garcia, and F.-G. Lamont, "Fisher's decision tree," *Expert Systems with Applications*, vol. 40, no. 16, pp. 6283–6291, 2013.

[17] D. Wickramarachchi, B. Robertson, M. Reale, C. Price, and J. Brown, "Hhcart: An oblique decision tree," 2015.

[18] D. G. G. Madzarov and I. Chorbev, "A multi-class svm classifier utilizing binary decision tree," *Informatica,*, vol. 33, pp. 233–241, 2009.

[19] N. Manwani and P. Sastry, "2012. geometric decision tree." *IEEE Transactions on Systems, Man, and Cybernetics*, vol. Part B: Cybernetics 42, no. 1, pp. 181–192, 2012.

[20] A. Leroux, M. Boussard, and R. Dès, "Information gain ratio correction: Improving prediction with more balanced decision tree splits," *ArXiv e-prints*, Jan. 2018.

[21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794.

[22] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154.

[23] L.-F. Chen, H. Liao, J.-C. Lin, and G. Yu, "A new lda-based face recognition system which can solve the small sample size problem." *Pattern Recognition*, vol. 33, pp. 1713–1726, 2000.

[24] M. Lichman, "UCI machine learning repository," 2013.