Received xxxxxx xx, xxxx, accepted xxxxxx xx, xxxx. Date of publication xxxx xx, xxxx, date of current version xxxx xx, xxxx.

*Digital Object Identifier 10.1109/ACCESS.201x.xxxxxxx*

# A Novel Hyperparameter-free Approach to Decision Tree Construction that Avoids Overfitting by Design

**RAFAEL GARCíA LEIVA, ANTONIO FERNANDEZ ANTA (Senior Member, IEEE), VINCENZO MANCUSO (Member, IEEE), PAOLO CASARI (Senior Member, IEEE)**

IMDEA Networks Institute, 28918 Madrid, Spain

Corresponding author: Rafael García Leiva (e-mail: rafael.garcia@imdea.org).

**ABSTRACT** Decision trees are an extremely popular machine learning technique. Unfortunately, overfitting in decision trees still remains an open issue that sometimes prevents achieving good performance. In this work, we present a novel approach for the construction of decision trees that avoids the overfitting by design, without losing accuracy. A distinctive feature of our algorithm is that it requires neither the optimization of any hyperparameters, nor the use of regularization techniques, thus significantly reducing the decision tree training time. Moreover, our algorithm produces much smaller and shallower trees than traditional algorithms, facilitating the interpretability of the resulting models.

**INDEX TERMS** Decision Trees, Regularization, Interpretability, Kolmogorov Complexity

## I. INTRODUCTION

Decision trees are a highly popular machine learning technique that has been successfully applied to solve a number of practical problems in areas such as natural language processing, information extraction, data mining, and pattern recognition [1]. Decision trees are also the foundation of more advanced machine learning methods, such as random forests or boosted trees [2].

One of the most challenging tasks for decision tree-building algorithms is to decide when to stop the tree growing process. Most of the tree building methods produce very complex models that overfit training data. Overfitted trees not only have poor predictive capabilities on newpreviously unseen data, but also can be exceedingly difficult to interpret, which is a key barrier against the adoption of these models in practical applications [3]. A common approach to avoid overfitting in decision trees is early stopping, which forces the construction algorithm to stop before the tree becomes too complex. Popular stopping criteria include limiting the maximum depth of the tree, requiring a minimum number of sample points at leaf nodes, or computing the accuracy gain yielded by new nodes [1]. However, these heuristics require the optimization of (possibly multiple) hyperparameters, which makes the training process computationally expensive.

An alternative approach to avoid overfitting in decision trees is to apply a post-processing pruning algorithm that removes those branches that contribute the least to the accuracy of the final model [4]. Cost-complexity [2] is a common pruning technique, based on applying a cost function that takes into account how well the tree fits the training data and a regularization factor based on how complex the tree is. If we denote by $E(T)$ the fitness of a tree $T$, and by $L(T)$ its complexity, the cost function $C(T)$ could be given by

$$C(T) = E(T) + \lambda L(T), \quad (1)$$

where $\lambda$ is a configurable parameter that controls the trade-off between fitness and complexity. In practice, the accuracy of the model is typically used as a fitness metric, and the number of leaf nodes in the tree as a complexity metric. Although, on average, cost complexity pruning produces better results than limiting the growth of the tree via early stopping [5], this technique requires the optimization of the hyperparameter $\lambda$.

In this paper we propose a decision tree construction algorithm with early stopping properties, that does not require hyperparameter tuning or pruning in order to get good accuracy and trees of small size.

### A. RELATED WORK

Earlier algorithms for the construction of decision trees were based on a exhaustive search of the space of possible trees (see [6] for a historical review of the discipline). However, due to the computational complexity of those algorithms, alternative approaches were investigated. Most of the modern algorithms used in practice are based on greedy search techniques (see for example [7]), that significantly reduce the training time, at the cost of producing sub-optimal trees.

Among the algorithms to build decision trees, the Iterative Dichotomiser 3 algorithm, ID3 [8], is based on an information gain splitting criteria, which stops when all the training samples belong to a single target value, or when no possible split has a positive information gain. Classification And Regression Trees, CART [9], builds binary trees based on a growing splitting criteria and applying a cost-complexity pruning. CHi-squared Automatic Interaction Detector, CHAID [10], relies on an adjusted significance tests to find the least significant values with respect to the target attribute, and the process is repeated until no significant pairs are found. C4.5 [11] is an evolution of ID3 that uses gain ratio as a splitting criteria, and ceases to grow the tree when the number of instances to split is below a given threshold. It includes an accuracy based post-pruning. The Quick Unbiased Efficient Statistical Tree algorithm, QUEST [12], uses quadratic discriminant analysis for splits, the stopping is based on statistical tests, and the resulting trees are pruned based on cross-validation. Multivariate Adaptive Regression Splines, MARS [13], is based on a two-phase (forward and backward pass) algorithm that uses regression functions approximated using linear splines and their tensor products. PUBLIC [14] is based on minimizing the sum of the length of the shortest computer program that can print the data given as input to the tree, and the length of the shortest program that can print the tree. Recent trends in tree building algorithms include deep trees [15] of hundred of levels, and oblique trees [16] with more advanced if-else conditions. A detailed comparison of decision trees algorithms can be found in [17], and an up-to-date review including other classification methods in [18].

### B. CONTRIBUTIONS

Unlike in the literature surveyed above, the main contribution of this work is a novel algorithm to construct decision trees that, by design, do not overfit the training data. The algorithm is based on a novel cost function that is used to decide when to stop the building process, rather than to indicate how to prune the generated tree. Our algorithm does not depend on the optimization of hyperparameters, thus considerably reducing the training time. This means that, with our algorithm, we can process a very significant amount of data with reduced complexity, e.g., by using the full dataset for training, or by enacting a training / validation split if a better estimate of the real accuracy of the resulting model is sought.

The models generated with our algorithm are significantly smaller in terms of number of nodes, and more shallow than the models generated by other decision tree building algorithms; at the same time, the obtained models present similar, and sometimes better accuracy. Shallower models allow us to make faster predictions when they are used as part of large ensembles of trees, and largely improve interpretability, e.g., when examined by domain experts. Additional advantages of our new algorithm include lower sensitivity to the presence of errors in the training dataset and to the non-linearity of the hyper-boundaries between different classes.

## II. NOTATION AND BACKGROUND

### A. NOTATION

Let $\mathcal{X}$ be a dataset composed of $n$ training input vectors $x_i \in \mathbb{R}^m$, were $1 \leq i \leq n$, and let $y \in \mathcal{G}^n$ be a category vector, where $\mathcal{G}$ is the set of classification labels ($\mathcal{G} = \{0, 1, \ldots, G\}$). The problem at hand is to find the best model $f$, from a given family of models $\mathcal{F}$, such that $f(x_i) = y_i$ for many $x_i \in \mathcal{X}$. That is, we are interested in solving a supervised classification problem. We are interested in the capability of the model $f$ to generalize to previously unseen data, that is, to correctly classify input vectors extracted from a universe $\mathcal{U}$ of input vectors, not necessarily included in the training dataset $\mathcal{X}$. For this reason, the best solution is usually not the overfitted model $f$ that guarantees $f(x_i) = y_i$ for all $x_i \in \mathcal{X}$.

### B. KOLMOGOROV COMPLEXITY

The novel cost function introduced in this paper to construct decision trees is based on the concept of Kolmogorov complexity [19]–[21], also known as Algorithmic Information Theory. The application of Kolmogorov complexity to the search of optimal statistical models is implemented by the Minimum Description Length (MDL) principle [22] and the Minimum Message Length (MML) [23]. In particular, minimum length techniques have been applied to the problem of inferring decision trees in [24], later on clarified and extended in [25], as well as in [26] as a technique for pruning; in [14] the PUBLIC algorithm based on the MML is described. Although the underlining concepts behind our own cost function are the same (namely, that learning is equivalent to the capability to compress), our approach is very different from the ones described above.

According to the Kolmogorov complexity, the amount of information of an object, encoded as a finite string, is given by the length of the shortest computer program that is able to print (output) that string. Kolmogorov complexity does not require to know the set of valid strings in advance or to make assumptions about their probability distribution. Therefore, it provides a universal definition of the amount of information contained in an object. However, the Kolmogorov complexity is a non-computable quantity [27], hence it must be approximated in practice.

The Kolmogorov complexity of a string $s$ composed by symbols from a fixed alphabet $\Sigma$, denoted by $K(s)$, is defined as[1]

$$K(s) = \min_{p,v} \left\{ |p| + |v| \, : \, U(p,v) = s \right\}, \tag{2}$$

where $U$ is a universal computer, $p$ is a program in a prefix-free language interpreted by $U$,[2] and $v$ is the input to the program. It can be shown that the Kolmogorov complexity $K(s)$ of a string $s$ does not depend on the programming language used [27], i.e., any reasonable and sufficiently powerful computer language provides the same description length, up to a fixed additive constant that depends on the selected language, but not on the string itself.

Sometimes, the description of a string can be greatly reduced if we assume the knowledge of another string. The conditional Kolmogorov complexity of a string $s$ given the string $s'$, denoted by $K(s|s')$, is defined as[3]

$$K(s|s') = \min_{p,v} \left\{ |p| + |v| \, : \, U(p, \langle v, s' \rangle) = s \right\}. \tag{3}$$

## III. DECISION TREE CONSTRUCTION ALGORITHM

### A. KEY IDEA

In this paper, we provide an algorithm to construct decision trees that, by design, does not overfit training data, and that has no hyperparameters to be optimized. To achieve this, the algorithm must automatically understand when growing the decision tree adds needless complexity, and must measure such complexity in a way that is commensurate to some prediction quality aspect, e.g., inaccuracy. We argue that a natural way to achieve the above objectives is to define both the inaccuracy and the complexity (conveyed by the so-called surfeit, introduced later on) using the concept of Kolmogorov complexity [19].

The key insight behind the application of Kolmogorov complexity to machine learning is that the more patterns we can find in a dataset, the more we have learned about the data. This means both identifying the dataset features that best explain the outcome, and to devise a model that provides this explanation without exceeding complexity. Moreover, data compression is about finding and exploiting patterns and regularities in the data. A practical implementation that puts together the key features of Kolmogorov complexity and data compression follows the lines of the Minimum Description Length principle [22], that proposes to minimize the length of the model plus the length of the data given the model, namely $L(M) + L(D \mid M)$, where $M$ denotes a model and $L(\cdot)$ returns the length of its argument. Unfortunately, this approach tends to favor simple models, and is not widely used in practice due to this limitation.

In this work, we propose to replace the minimization of $L(M) + L(D \mid M)$ by a multiobjective optimization problem [28] that reconciles two conflicting goals: to minimize the complexity of the model, and to minimize the inaccuracy of the model's predictions. Moreover, instead of working directly with the length of models $L(M)$ we propose to work with their surfeit, which evaluates the unnecessary complexity of a model used to represent a given dataset. The surfeit can be computed as the length of the model minus the length of the minimum computer program that can print the dataset, i.e., the Kolmogorov complexity of the data. Namely, this is $L(M) - K(D)$. Using the surfeit avoids the rejection of correct but complex models. Our approach is validated in practice by means of its application to a family of decision tree models.

### B. OVERVIEW OF THE ALGORITHM

We describe now the algorithm (called Minimum Surfeit and Inaccuracy, or MSI for short) to build a decision tree given a training dataset $\mathcal{X}$ with the help of the pseudocode reported in Algorithm 1. The algorithm is based on a breadth-first tree traversal, see for example [7]. The algorithm requires a function called bestSplit(), that returns the best way to split a given subset of the training data into two subsets, and a second function called costFunc(), that provides a quantitative evaluation of the cost of a tree in terms of complexity and accuracy. The details of these two functions are given in Sections III-C and III-D, respectively. A third function Forecast() computes the most likely class of a given subset of the training data. Algorithm 1 is based on two nested loops: the external **while** loop keeps a set $Candidates$ of the candidate tree nodes (leaves) to grow, whereas the internal **for each** loop finds the best such node from which the tree should be grown further. The latter operation requires to check all possible options and select the one that minimizes the cost of the resulting tree. The exit point in the algorithm is at the end of the **while** loop, where the current tree $T$ is returned if there are no more candidate nodes to further grow the tree, or the nodes in the set $Candidates$ generate trees that do not reduce the cost. The nodes of the tree are represented with the *TreeNode* data structure composed of the elements (1) *LChild*, that points to the left child node in the tree, (2) *RChild*, that points to the right child node, (3) *Data*, that contains the subset of training data of the node, (4) *Split*, which is a pair composed by the feature and a threshold used for the data splitting, and (5) *Class*, which is the most likely class for this node (based on its *Data*). A tree node could be either an internal node having a split criterion and two children, or a leaf node with a predicted class and no child.

The main difference between our algorithm and other decision tree algorithms is in the **for each** loop. In traditional algorithms, the order in which the branches are evaluated is irrelevant. However, since our algorithm could stop the growing process at any point, at each iteration we explicitly select the best candidate node to grow the tree.

[1] $|x|$ denotes the number of symbols, or length, of a string $x$.
[2] A programming language is prefix-free if no program can be a prefix of another program.
[3] $\langle v, s' \rangle$ denotes the concatenation of the strings $v$ and $s'$

**Algorithm 1** Minimum Surfeit and Inaccuracy (MSI)

```
1: procedure BUILDTREE($data$)
2:     Create TreeNode $r$; $r.LChild, r.RChild, r.Split \leftarrow none$; $r.Data \leftarrow data$; $r.Class \leftarrow$ FORECAST($data$)
3:     Create Tree $T$; $T.Root \leftarrow r$; $bestCost \leftarrow$ COSTFUNC($T$)
4:     $Candidates \leftarrow \{r\}$
5:     while $Candidates \neq \emptyset$ do
6:         $bestSol \leftarrow none$
7:         for each $\ell \in Candidates$ do
8:             $\Theta \leftarrow$ BESTSPLIT($\ell.Data$)
9:             if $\Theta = \emptyset$ then $Candidates \leftarrow Candidates \setminus \{\ell\}$    ▷ Discard $\ell$ as candidate to grow the tree.
10:            else
11:                Create TreeNode $\ell L, \ell R$    ▷ Tree nodes $\ell L, \ell R$ will be the children of $\ell$.
12:                $\ell L.LChild, \ell L.RChild, \ell L.Split, \ell R.LChild, \ell R.RChild, \ell R.Split \leftarrow none$
13:                $(dataL, dataR) \leftarrow$ SPLIT($\Theta, \ell.Data$); $\ell L.Data \leftarrow dataL$; $\ell R.Data \leftarrow dataR$
14:                $\ell L.Class \leftarrow$ FORECAST($dataL$); $\ell R.Class \leftarrow$ FORECAST($dataR$)
15:                $\langle \ell.LChild, \ell.RChild, \ell.Split \rangle \leftarrow \langle \ell L, \ell R, \Theta \rangle$    ▷ Grow the tree at node $\ell$.
16:                $C \leftarrow$ COSTFUNCT($T$);    ▷ $C$ is the cost of the tree if grown at $\ell$.
17:                $\langle \ell.LChild, \ell.RChild, \ell.Split \rangle \leftarrow \langle none, none, none \rangle$    ▷ Leave the tree as it was.
18:                if $C < bestCost$ then $bestCost \leftarrow C$; $bestSol \leftarrow (\ell, \Theta, \ell L, \ell R)$
19:        if $bestSol \neq none$ then
20:            $(\ell, \Theta, \ell L, \ell R) \leftarrow bestSol$
21:            $\langle \ell.LChild, \ell.RChild, \ell.Split \rangle \leftarrow \langle \ell L, \ell R, \Theta \rangle$
22:            $Candidates \leftarrow Candidates \setminus \{\ell\} \cup \{\ell L, \ell R\}$
23:        else return $T$    ▷ No candidate reduces the cost of the tree.
24:    return $T$    ▷ No more candidates.
```

### C. SPLITTING CRITERION

A decision tree is an algorithm that recursively partitions the training vectors of $\mathcal{X}$ in such a way that the same values $y_i$ are grouped together. Given a subset $Q \subseteq \mathcal{X}$ we have to find the optimal split for $Q$. A split is a pair $\theta = (j, w)$ were $1 \leq j \leq m$ is an index and $w \in \mathbb{R}$ a threshold. A split partitions the set $Q$ into two disjoint subsets $Q_l = \{x_i \in Q : x_{ij} \leq w\}$, and $Q_r = Q \backslash Q_l$. We use the minimal weighted entropy as the splitting criterion.[4] The weighted entropy of a split, denoted by $\tilde{H}$, is defined as:

$$\tilde{H}(Q, \theta) = \frac{d(Q_l)}{d(Q)} H(Q_l) + \frac{d(Q_r)}{d(Q)} H(Q_r) \tag{4}$$

where $d(S)$ is the number of elements (or diameter) of set $S$, and $H(S)$ is the entropy of $S$, i.e., $H(S) = -\sum_{x \in S} p(x) \log p(x)$, being $p(x)$ the probability of getting $x$ if we select a random element from $S$. The function bestSplit$(Q)$ returns the best split $\theta^\star$ of a given subset of data $Q$, defined as $\theta^\star = \arg\min_\theta \tilde{H}(Q, \theta)$. If $\theta^\star$ does not split $Q$, bestSplit$(Q)$ returns $\emptyset$.

### D. COST FUNCTION

For every possible branch to grow, we have to compute how good the resulting tree would be if we add the new nodes, compared to the same tree without them. In this section we introduce a novel cost function to evaluate and compare candidate trees. A problem with traditional cost functions for tree evaluation is that they are based on incommensurable quantities, such as the accuracy of the model and the number of nodes. Our goal here is to introduce two new metrics that are conceptually equivalent to the traditional ones while still being commensurable, i.e., based on the same units. We have also designed them to have the same scale, in order to enable a direct comparison. These two metrics will convey how badly the tree fits the data (inaccuracy), and how unnecessarily complex the model is (what we call *surfeit*).

Our measure of the inaccuracy of a model $M$ for a dataset $\mathcal{X}$ will be the length of the shortest computer program that, given as input the model $M$, is able to print the dataset $\mathcal{X}$. We seek to measure how difficult (in terms of the size of a program, not its running time) it is to fix the errors introduced by the model. Intuitively, in this sense, it is more difficult to fix a model that makes one hundred different mistakes than a model that makes one hundred times the same mistake. Formally, the inaccuracy of a dataset $\mathcal{X}$ and a candidate model $M$ is

$$\mathcal{I}(\mathcal{X}, M) = \frac{K(\mathcal{X}|M)}{K(\mathcal{X})}, \tag{5}$$

where $K(\mathcal{X}|M)$ is the conditional Kolmogorov complexity of the dataset $\mathcal{X}$ given the model $M$. The normalization factor $K(\mathcal{X})$, the Kolmogorov complexity of the dataset $\mathcal{X}$, is introduced to guarantee that both the inaccuracy and the surfeit have the same scale. We recall that Kolmogorov complexity is a non-computable quantity. Therefore, we ap-

[4]Other metrics, like Gini impurity or information gain could be used for the same purpose.

proximate $\mathcal{I}(\mathcal{X}, M)$ by the ratio $|Comp(E)|/|Comp(\mathcal{X})|$, where $|Comp(E)|$ is the length of the compressed version of the subset $E \subset \mathcal{X}$ composed by those points that have been misclassified by the tree, and $|Comp(\mathcal{X})|$ is the length of the compressed version of the full dataset.

To assess the model's complexity, we seek to measure the amount of surfeit introduced by our current model with respect to the shortest possible model. Formally, the surfeit of a dataset $\mathcal{X}$ and a candidate model $M$ is

$$\mathcal{S}(\mathcal{X}, M) = 1 - \frac{K(\mathcal{X})}{|M|}. \tag{6}$$

The length of the shortest possible model for the dataset $\mathcal{X}$ is given by its Kolmogorov complexity $K(\mathcal{X})$. The normalization factor is given by the length of the current model being evaluated, $|M|$. Formally, exceedingly short models $M$ such that $|M| < K(\mathcal{X})$, are not considered candidate models. Again, because $K(\mathcal{X})$ is a non-computable quantity, in practice we approximate the surfeit of a model by its redundancy, that is, by computing $1 - |Comp(M)|/|M|$, where $|Comp(M)|$ is the length of a compressed version of a string describing the tree $M$. We remark that the shortest possible model $M$ for a dataset $\mathcal{X}$ must be incompressible, but an incompressible model for $\mathcal{X}$ is not necessarily the shortest possible one. That is, it might happen that a model is not redundant but still presents some surfeit.

Both quantities, inaccuracy and surfeit, have to be combined into a single value by applying a function

$$\mathcal{N}(\mathcal{X}, M) = g(\mathcal{I}(\mathcal{X}, M), \mathcal{S}(\mathcal{X}, M)). \tag{7}$$

Since both the inaccuracy and the surfeit are relative quantities, a natural way to combine them is via their harmonic mean. Other candidate functions (arithmetic mean, geometric mean, Euclidean distance, product, and addition) will be evaluated in the following, in order to confirm that the selected function provides the best result among these.

## IV. PRACTICAL IMPLEMENTATION

The abstract concept of Kolmogov complexity is usually approximated in practice with compression algorithms [27]. In this work we have tested three different algorithms: the Lempel-Ziv-Markov chain algorithm, LZMA [29] that uses dictionaries for compression, zlib [30] that combines dictionaries with Huffman encodings, and `bz2` a compressor based on the Burrows-Wheeler transform [31]. All compressors have been configured to use the maximum compression level allowed, in order to avoid problems due to small window size buffers [32].

For the representation of a tree as a string we use the following template:

```
def tree{[attrs]}:
    if [attr] <= [thresh]:
        return [label] || [subtree]
    else:
        return [label] || [subtree]
```

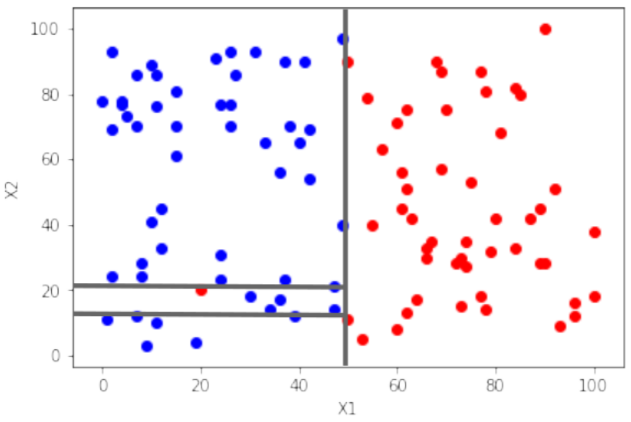

Figure 1: Synthetic dataset with splits computed by CART.

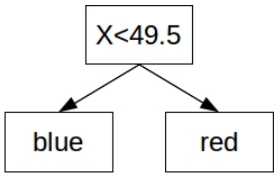

Figure 2: Decision tree obtained by our algorithm.

Where `[attrs]` is the list of attributes used, and only those used in the model,[5] `[attr]` is a single attribute represented by the letter `X` followed by a number (e.g. `X1`), `[thresh]` is the threshold used for the split, `[label]` is one of the valid labels from the set $\mathcal{G}$, and `|| [subtree]` means that the `return` statement can be replaced by another level of `[if - else]` conditions. We could have used a much shorter description of trees by replacing word tokens with symbols, e.g., via the ternary conditional operators `?` and `:` used in modern programming languages, or by dropping the `return` statement. This would produce shorter trees, but the complexity of the models would remain the same, up to an additive constant that does not depend on the models themselves. Since, e.g., the harmonic mean compares relative values instead of absolute ones, this additive constant can be safely ignored.

## V. RESULTS

In this section we evaluate our new algorithm, and compare its performance against the well-known algorithms CART, C4.5, CHAID, and PUBLIC. In order to measure prediction performance of the tree classifiers, we used the mean accuracy, defined as one minus the mean error. Other performance parameters taken into account include the total size of the tree, and the maximum depth.

Figure 1 shows a synthetic dataset consisting of 100 random points lying on a two-dimensional plane, where all the points with an $X1$ attribute smaller than 50 are colored blue,

[5]If the dataset contains many attributes, listing all of them when dealing with very short models would make the length of the model's header greater than the length of the body.

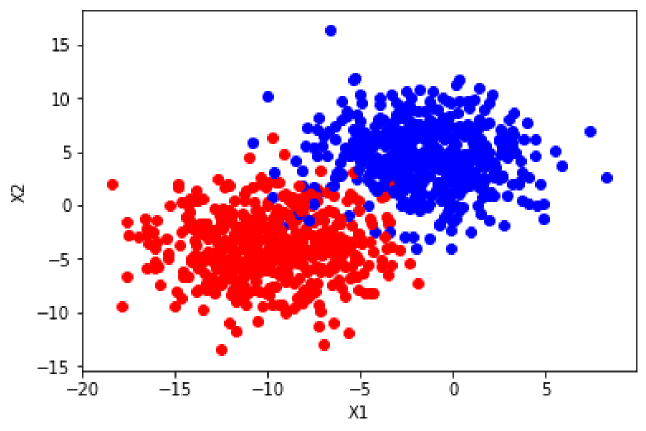

Figure 3: Example of isotropic Gaussian blobs (std. dev. = 3).

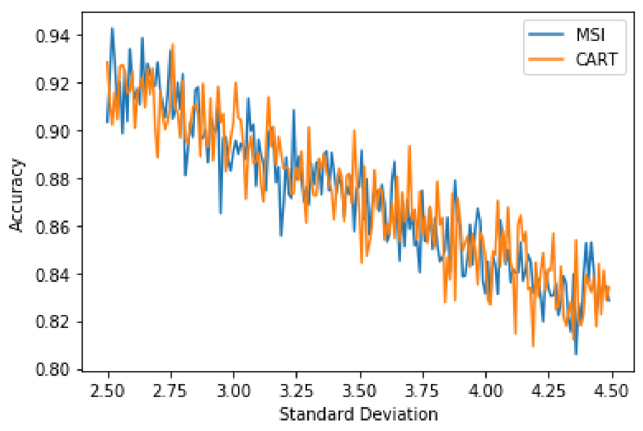

Figure 4: Accuracy for isotropic Gaussian blobs.

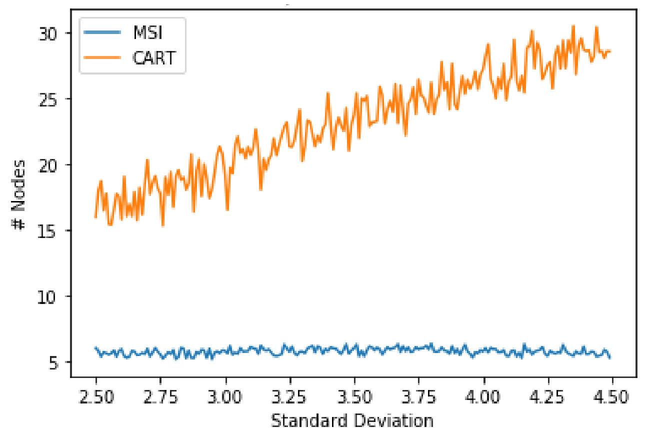

Figure 5: Number of tree nodes for isotropic Gaussian blobs.

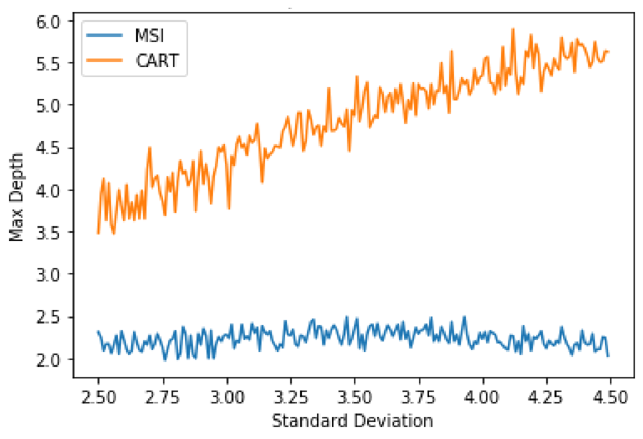

Figure 6: Maximum tree depth for isotropic Gaussian blobs.

and the rest as red. We artificially introduce a red point, simulating a measurement error, in the blue area. The gray lines correspond to the classification performed by the CART algorithm, as implemented by the scikit-learn toolkit [33]. CART will not stop until all the points have been correctly classified, so we require to have at least five points in order to make a node split, a de facto standard suggested, for example, in Section 9.2.2 of [2] (for a more theoretical justification of this minimum value see the expected count condition at [34]).

The tree obtained by applying our algorithm to Figure 1 can be seen in Figure 2. Our algorithm does not try to model the error point, since the gain due to an increment in the accuracy does not compensate the redundancy introduced in the model. Recall that our algorithm stops growing the tree when the total cost function of the tree, based on the measures of inaccuracy and surfeit, does not decrease when adding new nodes to the tree. Our algorithm presents a lower sensitivity to the errors found in datasets, at least if the number of errors is small compared to the number of valid points.

A second experiment with synthetic datasets is depicted in Figure 3. There, we create two isotropic Gaussian blobs that partially overlap. We start with a standard deviation of $2.5$ for each cluster, so they are easy to separate, and we increase the standard deviation in increments of $0.01$, until we reach $4.5$, which causes significant overlaps. For each value of the standard deviation, we run the experiment $100$ times and we compute the average accuracy for the two algorithms using different datasets for training and testing. For this experiment, the hyperparameter “minimum number of samples per leaf node” of the CART algorithm has been optimized in order to achieve the maximum accuracy (in this case, the best value was obtained with a minimum size of $26$ samples). The results of this experiment are shown in Figure 4. On average, our algorithm provides the same accuracy ($0.872$ on average) as the optimized version of the CART algorithm.

For each iteration of the experiment, we have also computed the average number of nodes, including internal and leaf nodes, required by the models to properly classify the clouds in the dataset. The results of this measurement are show in Figure 5. Our algorithm requires an average of $5.7$ nodes compared to $23$ nodes for the CART algorithm. Moreover, our algorithm is more stable than CART, in the sense that it produces models of similar complexity when it gets similar input datasets (a standard deviation of $0.3$ compared to $3.9$ for CART).

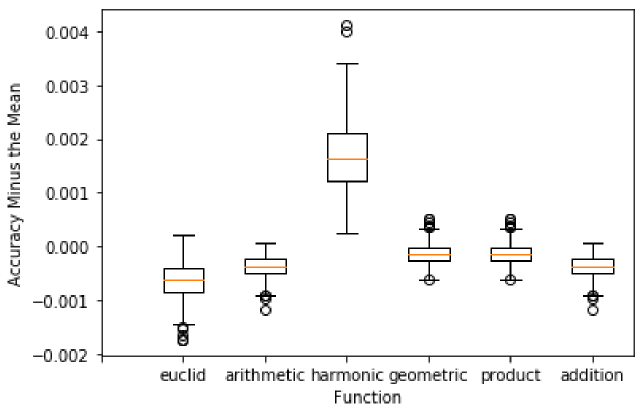

Figure 7: Evaluation of cost functions.

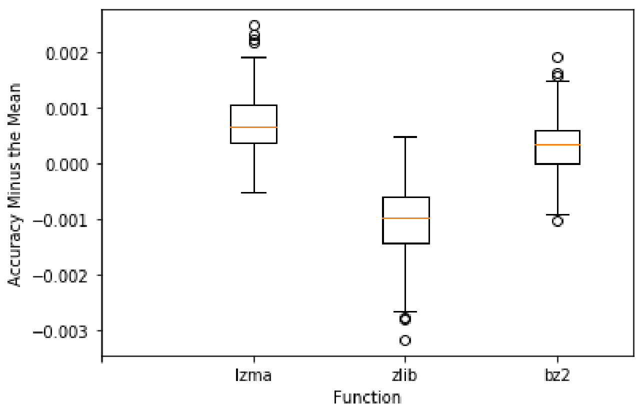

Figure 8: Evaluation of different compressors.

In Figure 6 we show the maximum depth of the tree, defined as the longest path from the root of the tree to any of its leaves. The maximum depth of the tree is a good measure of the maximum time it will require for the model to provide a classification. Our algorithm has an average depth of 2.2 nodes, whereas the average depth yielded by the CART algorithm is 4.8 nodes. We emphasize that the CART algorithm requires to optimize a configuration hyperparameter in order to obtain these optimal results, whereas the algorithm we propose, by design, does not require this optimization at all.

Before proceeding, we would like to stress that neither the cost function nor the compressor are hyperparameters. We demonstrate this by showing that the choice of different cost functions and compressors leads to substantially similar results. In Figure 7, we apply our algorithm to the discussed partially overlapping isotropic Gaussian blobs, and evaluate different alternatives for the definition of the cost function $\mathcal{N}(\mathcal{X}, M)$ in (7): arithmetic mean $(\mathcal{I}(\mathcal{X}, M) + \mathcal{S}(\mathcal{X}, M))/2$, geometric mean $(\mathcal{I}(\mathcal{X}, M) \times \mathcal{S}(\mathcal{X}, M))^{1/2}$, harmonic mean $2/(\mathcal{I}(\mathcal{X}, M)^{-1} + \mathcal{S}(\mathcal{X}, M))^{-1})$, Euclidean distance $(\mathcal{I}(\mathcal{X}, M)^2 + \mathcal{S}(\mathcal{X}, M)^2)^{1/2}$, sum $\mathcal{I}(\mathcal{X}, M) + \mathcal{S}(\mathcal{X}, M)$, and product $\mathcal{I}(\mathcal{X}, M) \times \mathcal{S}(\mathcal{X}, M)$. The figure shows the extremely limited difference between the different functions (for an easier interpretability of the results, the mean has been subtracted from the values). Similarly, Figure 8 shows the performance of our algorithm when using the LZMA, zlib, and bz2 compressors. We observe that all of them yield similar performance. From the above results, we conclude that the performance our algorithm is independent of the specific choice made for either implementation aspect.

In the following, we employ the bz2 as the compressor to approximate the computation of the Kolmogorov complexity, and the harmonic mean as the cost function. The latter has the additional advantage of comparing relative values instead of absolute ones, and therefore makes it possible to get rid of additive constants related to the choice of a specific template for the representation of the tree, as discussed in Section IV.

Finally, we have compared the performance of our algorithm with the performance of four popular decision trees algorithms, CART, C4.5, CHAID, and PUBLIC, with a collection of real datasets. More specifically, we have selected 12 well known datasets from the UCI Machine Learning Repository [35], among those with the largest amount of data. The selected ones are: diagnosis of breast cancer (`cancer`), optical recognition of handwritten digits (`digits`), predicting protein localization sites in gram-negative bacteria (`yeast`), classification of NASA space shuttle data (`shuttle`), classification of blocks in web pages (`page`), segmentation of outdoor images (`image`), predicting the age of abalones from physical measurements (`abalone`), predicting the quality of red and white variants of Portuguese wine (`wine`) [36], filter spam emails (`spam`), wall-following robot navigation (`wall`), classification of land use based on Landsat satellite images (`landsat`), and distinguishing signals from background noise in the MAGIC gamma telescope images (`magic`). For each dataset, we have repeated the experiment 100 times[6], by randomly selecting the training (70%) and testing (30%) subsets at each iteration.

In Figure 9 we compare the accuracy of the resulting models obtained by applying CART, C4.5, CHAID, as well as our own new algorithm to the above real datasets. In 3 of the 12 datasets, our algorithm provides better accuracy than CART. In the remaining 9 cases, the accuracy is fully comparable, and less than 2% smaller than the best of the other algorithms, on average. In Figure 11 we provide a comparison of the depth of the resulting models. Our algorithm yields a shallower tree than all other algorithms, except for datasets `Cancer`, `Digits`, and `Image`, confirming the non-overfitting properties of our tree construction process. In Figure 10, we show a comparison of the number of nodes of the resulting models. Only for one of the datasets (`Digits`), does our model produce a slightly larger tree that those generated by CART, C4.5, CHAID and PUBLIC. In most of the cases, the trees generated by our algorithm have up to three orders of magnitude fewer nodes.

Finally, we have compared our algorithm with a post-processing pruning algorithm applied to the tree obtained by

[6]In order to limit the computational complexity, the number of repeated experiments was limited to less than 100 for the CHAID algorithm when applied to some particularly heavy datasets.

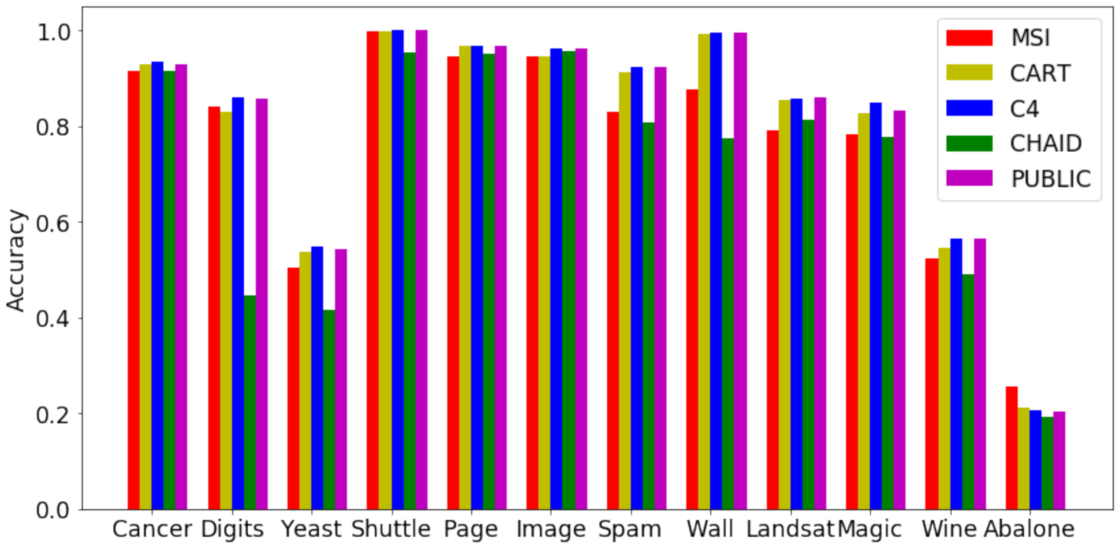

Figure 9: Accuracy over example datasets.

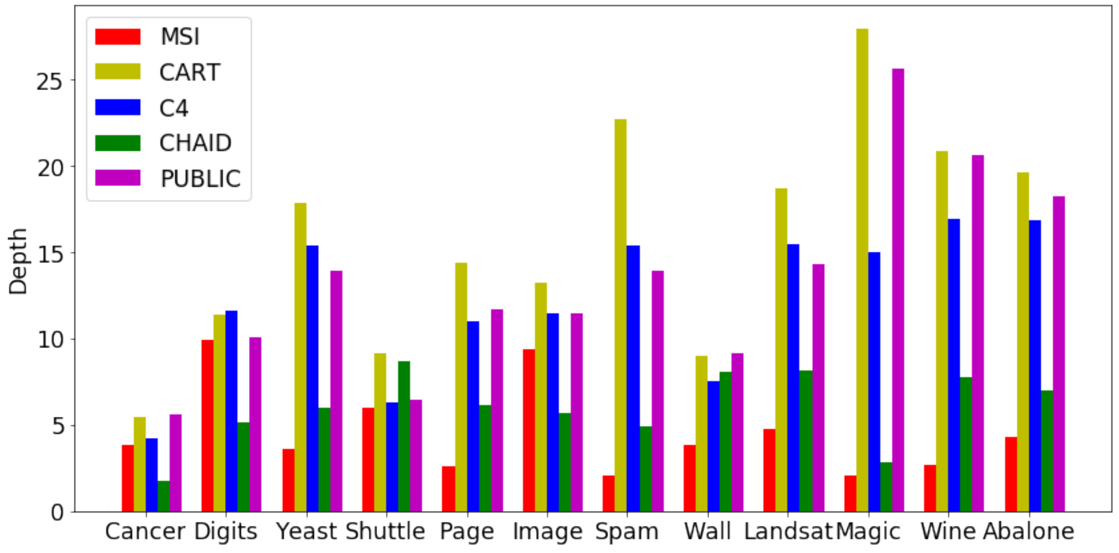

Figure 10: Depth of models over example datasets.

CART. In particular, we have applied a cost-complexity pruning guided by a cross-validation to the `Shuttle` dataset, as implemented by the rpart library in R. The optimal value of the cost-complexity metric is achieved for a tree of 59 nodes, whereas our algorithm, without requiring the optimization of any hyperparameters, obtains a tree of 28 nodes. Both trees achieve the same accuracy.

## VI. LESSONS LEARNED

The algorithm proposed in this paper has been designed to find a compact model that describes well a dataset (high accuracy) without over-fitting the data. The experiment described in Figure 1 suggest that our algorithm has lower sensitivity than CART to the errors found in datasets, at least if the number of errors is small compared with the number of valid points. The experiment of Figure 3 shows that the algorithm tends to produce simpler models when the classes that compose the dataset are not linearly separable. These two situations, errors and non-linearity, are common causes of model overfitting when using decision trees. In general, the CART algorithm produces much more complex models than our algorithm in those situations, even when configured to avoid overfitting as much as possible.

As we can see in Figures 9, 10 and 11, our new algorithm produces trees with a significantly smaller number of nodes (and depth) than those produced with standard algorithms used in practice, namely CART, C4, CHAID and PUBLIC, without significantly decreasing the accuracy. The experiments have been performed with a collection of datasets resulting from real experiments in order to test the applicability

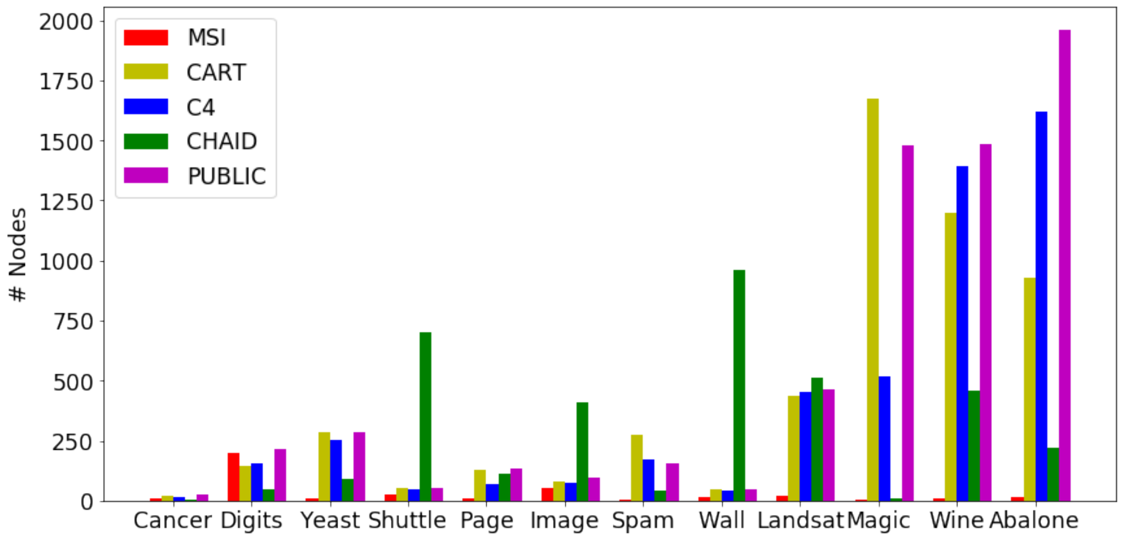

Figure 11: Number of nodes over example datasets.

of the new algorithm to practical problems. Given the size of the trees produced, our algorithm is the ideal method to apply in those cases where the interpretability of the results is critical, or where there is a large risk of model overfitting.

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a new algorithm to build decision trees based on the compressibility of candidate models and the compressibility of the errors generated by those models. The main advantage of the new algorithm is that it does not overfit the training data by design, and additionally it does not require external or ad-hoc procedures to control the overoptimization of the produced model. Moreover, our algorithm does not require the optimization of any hyperparameters. Experimental validation with synthetic and real datasets demonstrates that the accuracy of the new algorithm is similar to the accuracy of well-known decision tree algorithms, like CART, C4.5, CHAID and PUBLIC. Our results show that the proposed algorithm produces models with a considerably smaller number of nodes, without any substantial accuracy decrease.

Future work includes extending the cost function to include a goodness measure for the attributes used at the nodes of the trees, e.g., based on how correlated the values of these attributes are with the target values. The early stopping properties of our algorithm can also be applied to other machine learning techniques, e.g., to find optimal deep neural network architectures.

## References

[1] R. Lior et al., Data mining with decision trees: theory and applications. World scientific, 2014, vol. 81.
[2] J. Friedman, T. Hastie, and R. Tibshirani, The elements of statistical learning. Springer series in statistics New York, 2001, vol. 1.
[3] M. Wu et al., "Beyond sparsity: Tree regularization of deep models for interpretability," arXiv preprint arXiv:1711.06178, 2017.
[4] M. Bohanec and I. Bratko, "Trading accuracy for simplicity in decision trees," Machine Learning, vol. 15, no. 3, pp. 223–250, 1994.
[5] B. D. Ripley, Pattern recognition and neural networks. Cambridge university press, 1996.
[6] W.-Y. Loh, "Fifty years of classification and regression trees," International Statistical Review, vol. 82, no. 3, pp. 329–348, 2014.
[7] T. H. Cormen, Introduction to algorithms. MIT press, 2009.
[8] J. R. Quinlan, "Induction of decision trees," Machine learning, vol. 1, no. 1, pp. 81–106, 1986.
[9] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees." Kluwer Academic Publishers, New York, 1984.
[10] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," Applied statistics, pp. 119–127, 1980.
[11] J. R. Quinlan, "C4.5: Programming for machine learning," Morgan Kauffmann, vol. 38, p. 48, 1993.
[12] W.-Y. Loh and Y.-S. Shih, "Split selection methods for classification trees," Statistica sinica, pp. 815–840, 1997.
[13] J. H. Friedman, "Multivariate adaptive regression splines," The annals of statistics, pp. 1–67, 1991.
[14] R. Rastogi and K. Shim, "Public: A decision tree classifier that integrates building and pruning," in VLDB, vol. 98, 1998, pp. 24–27.
[15] D. Ignatov and A. Ignatov, "Decision stream: Cultivating deep decision trees," in Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on. IEEE, 2017, pp. 905–912.
[16] M. A. Carreira-Perpinán and P. Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in Advances in Neural Information Processing Systems, 2018, pp. 1219–1229.
[17] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," Machine learning, vol. 40, no. 3, pp. 203–228, 2000.
[18] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, "An up-to-date comparison of state-of-the-art classification algorithms," Expert Systems with Applications, vol. 82, pp. 128–150, 2017.
[19] R. J. Solomonoff, "A formal theory of inductive inference. part i and ii," Information and control, vol. 7, no. 1, pp. 1–22, 1964.
[20] A. N. Kolmogorov, "Three approaches to the quantitative definition of information'," Problems of information transmission, vol. 1, no. 1, pp. 1–7, 1965.
[21] G. J. Chaitin, "On the simplicity and speed of programs for computing infinite sets of natural numbers," Journal of the ACM (JACM), vol. 16, no. 3, pp. 407–422, 1969.
[22] P. D. Grünwald, The minimum description length principle. MIT press, 2007.
[23] C. S. Wallace, Statistical and inductive inference by minimum message length. Springer Science & Business Media, 2005.

[24] J. R. Quinlan and R. L. Rivest, "Inferring decision trees using the minimum description lenght principle," Information and computation, vol. 80, no. 3, pp. 227–248, 1989.
[25] C. S. Wallace and J. Patrick, "Coding decision trees," Machine Learning, vol. 11, no. 1, pp. 7–22, 1993.
[26] M. Mehta, J. Rissanen, R. Agrawal et al., "Mdl-based decision tree pruning." in KDD, vol. 21, no. 2, 1995, pp. 216–221.
[27] M. Li and P. Vitányi, An introduction to Kolmogorov complexity and its applications. Texts in Computer Science. Springer, New York,, 2008, vol. 9.
[28] K. Miettinen, Nonlinear multiobjective optimization. Springer Science & Business Media, 2012, vol. 12.
[29] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on information theory, vol. 23, no. 3, pp. 337–343, 1977.
[30] P. Deutsch, "Deflate compressed data format specification version 1.3," IETF RFC 1951, Tech. Rep., May 1996.
[31] M. Burrows and D. J. Wheeler, "Technical report 124: A block-sorting lossless data compression algorithm," 1994.
[32] M. Cebrián Ramos, M. Alfonseca, and A. Ortega, "Common pitfalls using the normalized compression distance: What to watch out for in a compressor," Communications in information and systems, 2005.
[33] F. Pedregosa et al., "Scikit-learn: Machine learning in python," Journal of machine learning research, vol. 12, no. Oct, pp. 2825–2830, 2011.
[34] D. S. Starnes, D. Yates, and D. S. Moore, The practice of statistics. Macmillan, 2010.
[35] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[36] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," Decision Support Systems, vol. 47, no. 4, pp. 547–553, 2009.