INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MÉXICO

SCHOOL OF ENGINEERING AND SCIENCES

A novel functional tree for class imbalance problems

A dissertation by

LEONARDO MAURICIO CAÑETE SIFUENTES

Submitted to the
School of Engineering and Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science

Atizapán de Zaragoza, Estado de México
5th December, 2022

Instituto Tecnologico y de Estudios Superiores de Monterrey

Estado de México Campus

The committee members, hereby, certify that have read the dissertation presented by **Leonardo Mauricio Cañete Sifuentes** and that it is fully adequate in scope and quality as a partial requirement for the degree of Doctor of Philosophy in Computer Science.

Dr. Raúl Monroy Borja
Tecnológico de Monterrey, Campus Estado de México
Principal Advisor

Dr. Miguel Angel Medina Pérez
AEROENGY
Co-Advisor

Eduardo Morales Manzanares
Instituto Nacional de Astrofísica, Óptica y Electrónica

Andrés Eduardo Gutiérrez Rodríguez
MAHLE Shared Services

Francisco Cantú Ortiz
Tecnológico de Monterrey, Campus Monterrey

Santiago Conant Pablos
Tecnológico de Monterrey, Campus Monterrey

Dr. Rubén Morales Menéndez
Dean of Graduate Studies
School of Engineering and Sciences
Tecnológico de Monterrey, Campus Monterrey

Atizapán de Zaragoza, Estado de México, 05th December, 2022

Declaration of Authorship

I, Leonardo Mauricio Cañete Sifuentes, declare that this thesis proposal titled, 'A novel functional tree for class imbalance problems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

Abstract

A novel functional tree for class imbalance problems

by Leonardo Mauricio Cañete Sifuentes

Decision trees (DTs) are popular classifiers partly because they provide models that are easy to explain and because they show remarkable performance. To improve the classification performance of individual DTs, researchers have used linear combinations of features in inner nodes (Multivariate Decision Trees), leaf nodes (Model Trees), or both (Functional Trees). Our general objective is to develop a DT using linear feature combinations that outperforms the rest of such DTs in terms of classification performance as measured by the Area Under the ROC Curve (AUC), particularly in class imbalance problems, where one of the classes in the database has few objects compared to another class.

We establish that, in terms of classification performance, there exists a hierarchy, where Functional Trees (FTs) surpass Model Trees, that in turn surpass Multivariate Decision Trees. Having shown that Gama's FT, the only FT to date, has the best classification performance, we identify limitations that hinder its classification performance.

To improve the classification performance of FTs, we introduce the Functional Tree for class imbalance problems (FT4cip), which takes care in each design decision to improve AUC. The decision of what pruning method to use led us to the design of the AUC-optimizing Cost-Complexity pruning algorithm, a novel pruning algorithm that does not degrade classification performance in class imbalance problems because it optimizes AUC. We show how each design decision taken when building FT4cip contributes to classification performance or to simple tree models.

We demonstrate through a set of tests that FT4cip outperforms Gama's FT and excels in class imbalance problems. All our results are supported by a thorough experimental comparison in 110 databases using Bayesian statistical tests.

Acknowledgements

Thanks to my advisors Dr. Raúl Monroy and Dr. Miguel Angel Medina-Pérez for their guidance through my studies. Thanks to my professors and members of the GIEE Machine Learning for their contribution to my formation as a researcher. Thanks to Bárbara Cervantes for her support and for her feedback to this research. Finally, thanks to all my family and friends for their support.

The research reported here is supported by a CONACYT grant.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Decision trees (DTs) are popular classifiers that have evolved for nearly sixty years as stand-alone and base classifiers for ensembles. One reason for the popularity of DTs is that their models are considered interpretable. There is a trend toward using white-box models with high accuracy because experts need to understand the models, and it is mandatory to explain the results in several practical problems [57]. Another advantage is that stand-alone DTs have reasonably good classification performance and fast computation speed [54]. Tree-based ensembles are popular because they achieve highly competitive classification results; in a 2017 survey [93], Random Forest [16] and XGBoost [22] are among the top-ranked algorithms; in a 2021 survey [76] using synthetic databases, bagged CART and Gradient Boosted Trees were the top-ranked algorithms in problems with few features and a high number of objects.

The original and most known type of DT is the Univariate Decision Tree (UDT), and probably the most famous UDT is CART [15]. UDTs are the easiest type of DT to interpret; however, they have the lowest classification performance among the family of DTs.

One way to improve the classification performance of UDTs is to allow DTs to use more than one feature in the condition for splitting a node. The family of trees that allow feature combinations, usually linear combinations, is composed of Multivariate Decision Trees (MDTs) [15], Model Trees (MTs) [46], and Functional Trees (FTs) [34]. MDTs allow feature combinations in inner nodes, MTs allow feature combinations in leaf nodes, and FTs allow feature combinations in all nodes. Previous works show that MDTs outperform UDTs in classification performance [17].

Although trees allowing feature combinations have better classification performance over UDTs, many algorithms have shortcomings when dealing with multi-class and class imbalance problems. The problem of class imbalance is critical because many real-world problems have class imbalance and it is not taken into account by many classifiers. Databases with class imbalance have a low proportion of objects of any class when compared to others [2]. Classifiers that do not take into account class imbalance may classify almost every object with the majority class and achieve high performance according to measures such as accuracy. However, in many class imbalance problems, we are interested in correctly classifying objects of the minority class. Therefore, it is important to design classifiers that consider the class imbalance problem and evaluate them using appropriate measures for these problems, such as the Area Under the ROC curve (AUC).

In this work, we present the first FT designed to deal with class imbalance problems of two or more classes, which we call the Functional Tree for class imbalance problems (FT4cip). FT4cip achieves an improved performance over Gama's FT [34], the only FT to date, measured with the Area Under the ROC Curve (AUC). The improved performance of FT4cip in class imbalance problems is mainly due to the use of a split evaluation function that considers class imbalance (Twoing) and a novel pruning algorithm that optimizes AUC.

1.1 Motivation

Although trees using linear combinations have shown to outperform UDTs in classification performance [17], the best DTs cannot be easily identified due to a generalized lack of proper statistical comparison of newer DT works with previous literature. We have alleviated this problem by publishing the most extensive survey on MDTs to date, which includes a statistical comparison of 19 MDTs in 57 databases [18].

To compare FT4cip against the top-performing DT, as measured by the Area Under the ROC Curve (AUC), we first show that Functional Trees surpass Model Trees, that in turn surpass Multivariate Decision Trees. We apply Bayesian statistical tests in 110 databases to identify if a classifier outperforms another or if they are equivalent in terms of AUC. Previously, Gama's FT was the only of its kind [34] and did not outperform the top model tree, the Logistic Model Tree (LMT) [46]. However, we will show that the latest implementation of Gama's FT does outperform LMT.

Although we will show that Gama's FT has the highest classification performance among the trees using linear feature combinations, it has limitations that hinder its classification performance. One of the main limitations of Gama's FT is that it does not consider class imbalance.

To deal with Gama's FT limitations, each design decision for FT4cip aims to maximize AUC. Wa aim to maximize AUC because it is a measure that takes into account class imbalance. We searched for the best strategies available in the literature or proposed our own if this achieved further improvement of AUC. The design decisions considered for a new FT are how to generate candidate splits, how to evaluate candidate splits, when to stop splitting a node, how to approach multi-class problems, what (if any) pruning method to use, what split types are allowed, what (if any) feature selection method to use, and what (if any) classifier to use at the leaves. We showed that each design decision taken is important through an experiment in which we modified a single design decision to match LMT or Gama; by doing so, the classification performance decreases, or the resulting model or algorithm is more complex than the one for FT4cip.

1.2 Recent applications of Functional Trees

Gama's FT was published on 2004 [34] and no other Functional Tree has been published to date. However, multiple authors have successfully applied Gama's FT in classification problems in recent years. In Section 2.4, we show that multiple MDT algorithms have been introduced after Gama's FT was published. We believe that this disparity in the development of FTs and MDTs is due a lack of exposure of FTs in the community because, as we show in Chapter 5, FTs achieve higher classification performance than MDTs.

Functional Trees have been applied in recent years in problems of landslide [70] susceptibility, flash flood susceptibility [4], avalanche susceptibility [67], phishing website detection [6], and attrition prediction in a population of very preterm infants [81].

In [70], an ensemble of Gama's FT was used to predict landslide susceptibility. The ensemble of Gama's FT outperformed a single FT, NBTree [45], CART [15], and Bagging [15].

In [6], Gama's FT and ensembles based on it were compared in the problems of website phishing detection against Naive Bayes, Sequential Minimal Optimization [5], SVM, and Decision Table. As a single tree, Gama's FT achieved better AUC than the rest of the classifiers. The ensembles of Gama's FT improved the classification performance of a single tree.

In [81], several classifiers were compared in the problem of predicting loss of participants (attrition) at subsequent follow-ups in a study of very preterm infants. Incremental models were trained at four different follow-ups. Overall, Random Forest achieved the best performance in all follow-ups. However, Gama's FT was ranked second in the fourth follow-up.

1.3 Objectives and contributions

The general objective of this research is to develop a DT using linear feature combinations that outperforms the rest of such DTs in terms of classification performance as measured by the AUC, particularly in class imbalance problems. To do so, we first show that the family of DTs using linear feature combinations has an order in classification performance, with functional trees at the top. Therefore, we designed a new functional tree, which we compared with the top-performing one (Gama's FT). Our specific objectives are:

- Demonstrate that the top-performing model tree, LMT, outperforms all MDTs of our recent survey.
- Demonstrate that Gama's FT, the only functional tree, has better classification performance than LMT.
- Design a functional tree that achieves high classification performance in class imbalance problems, that is FT4cip.
- Demonstrate that, in general, FT4cip has better classification performance, in terms of AUC, than Gama's FT.
- Demonstrate that the classification performance of FT4cip, in terms of AUC, is particularly good in problems with class imbalance, when compared to Gama's FT.

In this thesis, we make two key contributions. First, we demonstrate that, in terms of classification performance, there exists a hierarchy, where functional trees surpass model trees, that in turn surpass multivariate trees. This result is supported by a set of statistical tests run over a thorough experimental comparison.

Second, we introduce a new functional tree, the Functional Tree for class imbalance problems (FT4cip), especially designed for class imbalance problems. We show that FT4cip has better classification performance than Gama's FT, the previous top-performing DT. Furthermore, we show that FT4cip excels in class imbalance problems.

A third contribution is the introduction of the AUC-optimizing Cost-Complexity pruning. This novel pruning algorithm is one of the elements that allows FT4cip to outperform Gama's FT.

1.4 Organization

The rest of this thesis is organized as follows. In Chapter 2, we show how to build functional trees and identify the key design decisions involved. We begin by showing the common structure of DTs and how to build them; then, we discuss the family of DTs that uses feature combinations; finally, we give a general algorithm for building FTs.

In Chapter 3, we present a protocol for a fair comparison of DT algorithms. We describe the experiments, where we compare DTs, carried out in this work. We describe the Bayesian statistical test used to compare DTs, the databases used, and the evaluation measure.

In Chapter 4, we introduce our Functional Tree for class imbalance problems (FT4cip). We begin by describing the choices taken for each design decision identified in Chapter 2. Then, we present the algorithm for training FT4cip. Next, we show the results of our first experiment, which evaluates the importance of the design decision of FT4cip. Finally, we make an analysis of the computational complexity and runtime of FT4cip.

In Chapter 5, we give the results of our second and third experiments, which show that functional trees surpass model trees, that in turn surpass multivariate trees, in terms of classification performance (AUC). Then, our fourth and fifth experiments show that FT4cip outperforms Gama's FT in classification performance, especially in class imbalance problems.

Finally, we give our conclusions in Chapter 6. In this chapter we outline future work and list the research papers published as result of this work.

Chapter 2

Induction of functional trees

Gama [34] designed Functional Trees (FTs) as the most general type of Decision Tree (DT) that uses linear combinations to separate objects of different classes; however, Gama's FT is the only one of its type to date. The most popular DTs, such as CART [15] and C4.5 [71], are specific cases of an FT without linear combinations; these types of trees are called Univariate Decision Trees (UDTs). The Multivariate Decision Tree (MDT) and Model Tree (MT) are two possible DTs that result from limiting the nodes where linear combinations can be used in an FT.

Through this chapter, we will identify the design decisions involved in building FTs, some of which are common to all DTs. This will allow us to introduce our new FT, FT4cip, in a well-organized way, and to identify later what design decisions of FT4cip contribute more to its classification performance.

We begin this chapter by describing the typical structure of DTs, how to build one, and the design decisions involved in building any DT. A specific example of building a simple DT is shown in Section 2.1.1. Then, Section 2.2 describes the family of DTs that use linear combinations. We discuss why no new FTs have been developed in Section 2.4. Finally, Section 2.3 gives a general algorithm for building FTs.

2.1 Common structure and construction of Decision Trees

Decision trees are commonly represented as directed graphs, as shown in Figure 2.1. In a classification problem, we are given a training database $\mathbf{D}$, with n objects and m features. When training, the decision tree starts as a single *root node*, marked at the top of Figure 2.1, which contains all objects in $\mathbf{D}$. The nodes connected to any node in the tree are called *children*; conversely, the node to which the children are connected is called the *parent node*. The arcs connecting the parent and children are called *branches*.

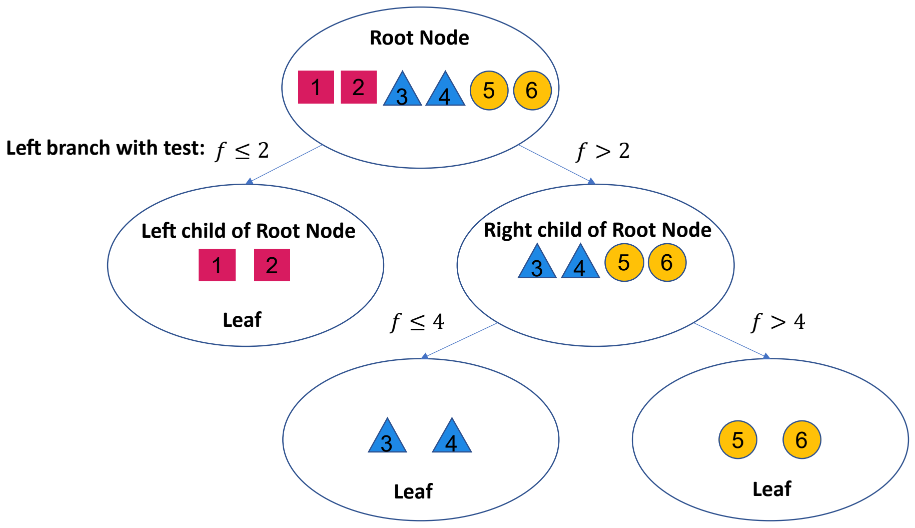

FIGURE 2.1: Decision Tree for an example database. We have labeled the root node, its children, an example of a branch, and the leaves.

A DT induction algorithm generates children by *splitting* the parent node, unless a stopping criterion is met (e.g. all object in the node belong to the same class). Each child is assigned a subset of the objects from the parent node, with the subsets forming a partition of the parent's objects. A function that partitions the objects into children is called a *split*. The *split* is represented as a collection of tests that define the partition. The branches of a tree are tagged with the corresponding test. For example, the tests $(color(x) = blue, color(x) \neq blue)$, where x is an object, represent a split that partitions the objects into a subset of blue objects and a subset of non-blue objects. The corresponding partition is $(\{x : color(x) = blue\}, \{x : color(x) \neq blue\})$.

Most DTs generate only two children per parent, known as binary DTs. Non-binary DTs may have more than two children per parent and can be transformed into binary DTs. In this work, we assume that we work with binary DTs.

A DT induction algorithm will recursively split nodes unless a stopping condition is met. Once a stopping condition is met in a node, it is marked as a *leaf*. A partition of database **D** is formed when collecting the subsets of objects in all leaves of the tree.

Algorithm 2.1 shows a simple way of training a DT. The following notation is used in the algorithm:

D. Database as an array of size $n \times m+1$. The first m columns correspond to non-class features; the additional feature is the class.

n. The number of objects in the database.

m. The number of features in the database.

$\mathcal{N}_t$. Node t.

$\mathbf{D_t}$. The objects at node $\mathcal{N}_t$.

$\mathcal{S}$. A candidate split.

`eval`$(\mathbf{D_t}, \mathcal{S})$. Split evaluation function, which evaluates the split $\mathcal{S}$ in node $\mathcal{N}_t$. The return value is a non-negative real number.

`partition`$(\mathbf{D_t}, \mathcal{S})$. Function that generates the left and right children $\mathcal{N}_l, \mathcal{N}_r$ and assigns to them the subset of objects from $\mathbf{D_t}$ according to split $\mathcal{S}$.

Algorithm 2.1 Basic algorithm for building a decision tree.

Input:
$\mathcal{N}_t$. Node t.
$\mathbf{D_t}$. The objects at node $\mathcal{N}_t$.
`eval`$(\mathbf{D_t}, \mathcal{S})$. Split evaluation function, which evaluates the split $\mathcal{S}$ in node $\mathcal{N}_t$. The return value is a non-negative real number.

```
BuildTree($\mathbf{D_t}$, $\mathcal{N}_t$, eval):
 1: if $\mathcal{N}_t = \emptyset$ then ▷ 1. Create a root node and assign it all objects in the training database.
 2:     $\mathcal{N}_t$ = root
 3:     $\mathbf{D_t} = \mathbf{D}$
 4: end if
 5: if a stop condition is met then
 6:     return $\mathcal{N}_t$          ▷ 2a. Mark the node as a leaf if a stop condition is met.
 7: end if
                                     ▷ 2b. If no stop condition is met, this node will be split.
 8: $S$ = GenerateCandidates($\mathbf{D_t}$)          ▷ 3. Generate a set of candidate splits.
 9: $\mathcal{S}^* = \arg\max_s$ eval($\mathbf{D_t}, s$), $s \in S$    ▷ 4. Select the split that maximizes the evaluation
    function.
10: $N_l, N_r$ = partition($\mathbf{D_t}, \mathcal{S}^*$)
11: BuildTree($\mathbf{D_l}, N_l$, eval)
12: BuildTree($\mathbf{D_r}, N_r$, eval)
13: return $N_t$
```

From the main steps of our algorithm, marked as comments, we identify three design decisions common to all DTs:

1. **How to generate candidate splits.** For numerical features, candidate splits are generated for each feature $f \in F$, usually of the form $(\{x : f(x) \leq v\}, \{x : f(x) > v\})$, where $f(x)$ is the value of feature f for object x, and $v \in \Re$ is known as the split point. Generally, DTs use exhaustive search to generate splits by varying v. Some exhaustive algorithms

test all values of feature f present in the database. In contrast, other exhaustive algorithms reduce the number of tests by sorting the objects by feature f and testing only when there is a class change between contiguous objects. As an alternative, DTs may use the class distribution given a feature to do an analytical split.

2. **What split evaluation function to use.** The split evaluation function gives a numerical evaluation of how well a split discriminates between classes. In this text, we will assume that our objective is to maximize the split evaluation function. Some authors use impurity measures that give higher values when children have objects of mixed classes (i.e., the nodes are impure). However, the minimization of impurity measures can be easily converted into a maximization problem. Ties are usually solved arbitrarily; for example, the first split with the highest evaluation is selected.

3. **What stop conditions to use.** The most common stop condition is that a node has only objects of one class (i.e., the node is pure). Another widespread stop condition is that the number of objects is below a minimum value; algorithms that do analytical splits may set a minimum number of objects due to the use of statistical tools.

Once the tree is built, we can use it to classify new objects. At each node, starting from the root, the new object x is tested with the tests tagging the branches. Once object x gives `true` for a test, it is directed to the corresponding child. The process is repeated until a leaf is reached. Once the object reaches a leaf, it is classified using the leaf's most common class (majority class).

Two additional design decisions that are not shown in the algorithm are:

4. **Approach to multi-class problems.** Some DT induction algorithms are designed to deal with multi-class classification problems, but there are others that are not. From the DT algorithms that work with multi-class problems, some of them need to transform the problems into two-class problems.

5. **What pruning method to use.** Pruning is an optional step, carried out after building a DT. Pruning is used to reduce the size of the tree while trying to preserve classification performance.

2.1.1 A basic example of Decision Tree induction.

In Figure 2.2, we show a basic example of how to build a Decision Tree. Our example database has six objects, a single feature f, and three classes. Objects with values $f(x) \in \{1, 2\}$ belong

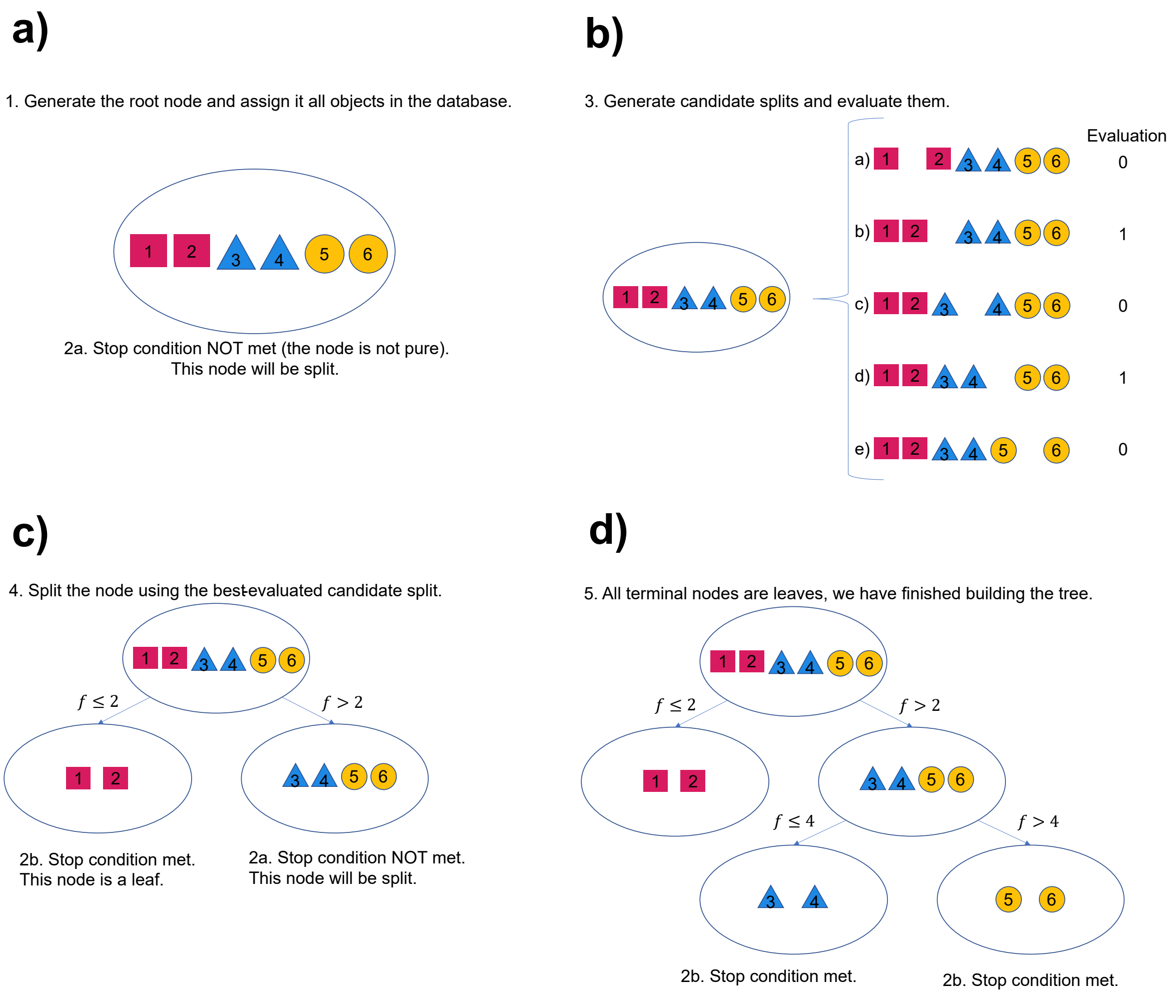
a)
1. Generate the root node and assign it all objects in the database.
1 2 3 4 5 6
2a. Stop condition NOT met (the node is not pure).
This node will be split.
b)
3. Generate candidate splits and evaluate them.
Evaluation
a) 1 | 2 3 4 5 6 0
b) 1 2 | 3 4 5 6 1
c) 1 2 3 | 4 5 6 0
d) 1 2 3 4 | 5 6 1
e) 1 2 3 4 5 | 6 0
c)
4. Split the node using the best-evaluated candidate split.
f ≤ 2
f > 2
2b. Stop condition met.
This node is a leaf.
2a. Stop condition NOT met.
This node will be split.
d)
5. All terminal nodes are leaves, we have finished building the tree.
f ≤ 4
f > 4
2b. Stop condition met.
2b. Stop condition met.

FIGURE 2.2: Basic example of DT induction.

to the magenta square class, objects with values $f(x) \in \{3, 4\}$ belong to the blue triangle class, and objects with values $f(x) \in \{5, 6\}$ belong to the yellow circle class.

For our example, the three design decisions are simple. To generate candidate splits, we use an exhaustive search. Our split evaluation function assigns a value of 1 to splits that generate a pure node; otherwise, it assigns a value of 0. Our only stop condition is that a node must be pure.

Following Algorithm 2.1, we begin by generating the root node and assigning all objects of our training database (Figure 1.1a). Since no stop condition is met, we continue to generate candidate splits in line 8. In Figure 1.1b, we see the five candidate splits and their evaluations with a tie for the best split between split b) and d). We arbitrarily select split b).

We show the tree resulting from the first split in Figure 1.1c. Following our algorithm, we recursively call the `BuildTree` function with each child node. Since the left-hand side child meets our stop condition (it is pure), it is marked as a leaf in line 6 and is not split. However, the right-hand side child does not meet our stop condition, so it is split following the same steps to split the root node. Splitting the right-hand side child generates two pure children, resulting in the tree in Figure 1.1d. All the terminal nodes are leaves at this point, so the DT construction ends. We can verify that the subsets of objects in the leaves form a partition of our database.

2.2 Family of Decisions Trees that use feature combinations

The most popular DTs are Univariate Decision Trees (UDTs), which use a single feature to split objects. The example trees of the previous section are UDTs. UDTs divide the feature space with axis-parallel hyperplanes. However, UDTs generate suboptimal splits when a single non-axis-parallel hyperplane may separate objects from different classes.

To improve the classification performance of DTs, authors have introduced DTs with feature combinations. Three types of DTs using feature combinations exist, distinguished by where the feature combinations occur. Multivariate Decision Trees (MDTs) allow splits combining features in inner nodes. Model Trees (MTs) allow feature combinations in leaf nodes by having a classifier, such as a logistic regression model, in each leaf that decides the class of objects falling in the leaf. Finally, Functional Trees (FTs) are a generalization of MDTs and MTs, allowing feature combinations in both inner and leaf nodes. The three types of trees improve classification performance compared to UDTs. Figure 2.3 shows an example of each tree.

A linear combination is the most common feature combination used in MDTs, MTs, and FTs. Tests with linear combinations may involve a subset of features. Let us assume that the database has F features, and we are given a subset of features $F' \subseteq F$. The tests with linear combinations take the form $\sum_{f \in F'} w_f f(x) \leq v$ or $\sum_{f \in F'} w_f f(x) > v$, where each $w_f \in \Re$ is a weight coefficient for feature f, $f(x)$ is the value of feature f for object x, and $v \in \Re$ is the split point. In contrast with UDTs, in addition to finding the split point v, MDTs, MTs, and FTs must search for the weight coefficients w_f for each feature f involved in the linear combination. Splits that use feature combinations in inner nodes are known as *multivariate splits*.

We now discuss how MDTs and MTs add design decisions that we need to consider when building a DT.

2.2.1 Multivariate Decision Trees

Introducing multivariate splits with MDTs adds more design decisions when building a DT. The first design decision is whether to use multivariate splits. The literature supports using

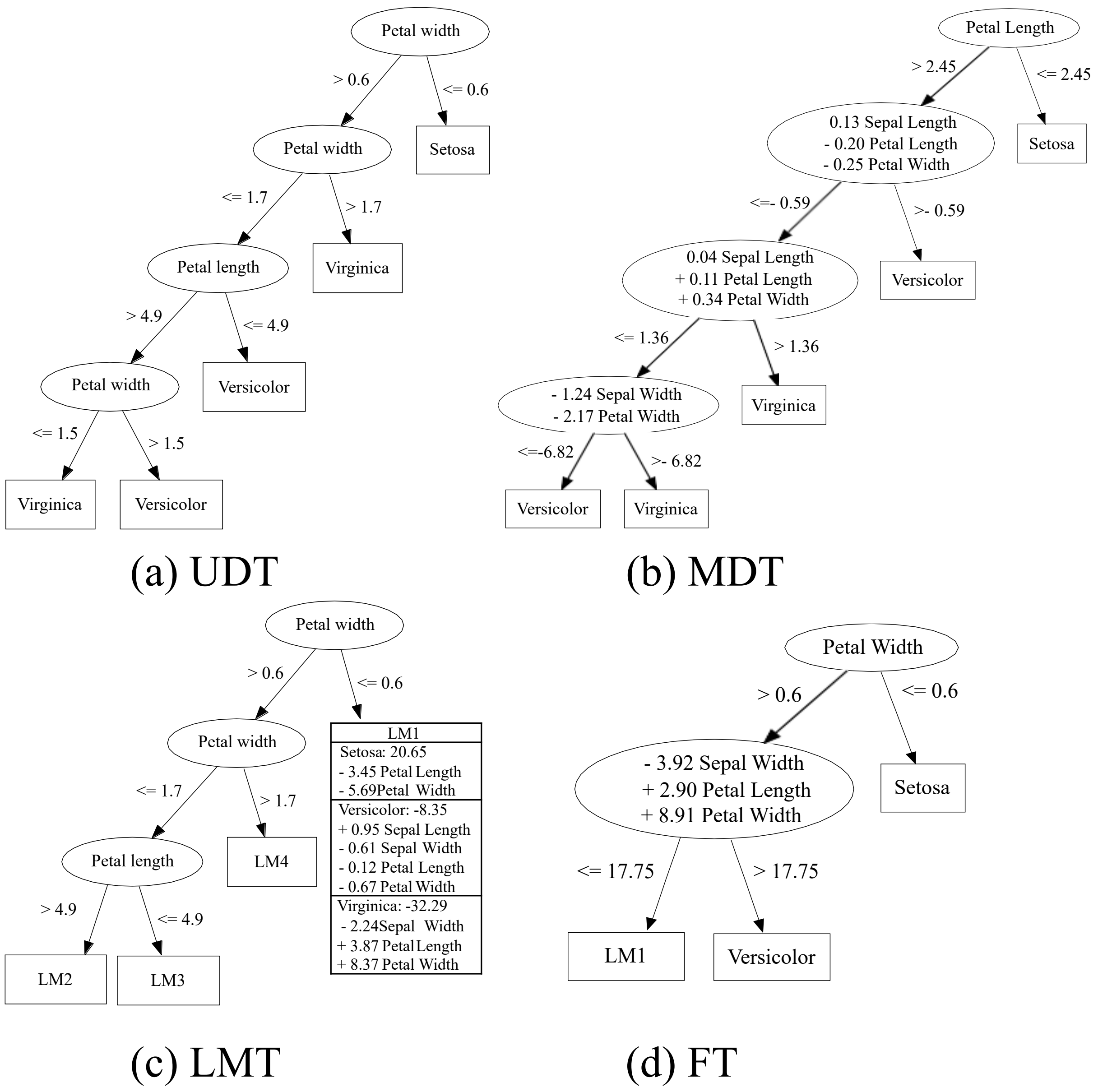

Figure 2.3: Decision trees for the Iris database [29]. (a) Univariate Decision Tree (UDT); all splits use a single feature, and the leaves are labeled with the majority class. (b) Multivariate Decision Tree (MDT); unlike the UDT, some splits have feature combinations. (c) Model Tree; unlike the UDT, the leaves have a classifier. In this example, the leaves have logistic models (LM), resulting in linear feature combinations, as seen in the expanded model (LM1). (d) Functional Tree (FT); this is a generalization of the previous trees. The FT can use both univariate and multivariate splits, and the class in the leaves can be chosen with the majority class or with a classifier. All trees were computed via Weka's classifiers [30].

multivariate splits since multiple authors have shown that MDTs achieve better classification performance than UDTs by using multivariate splits [68] [48].

There are several algorithms to build MDTs, with more than 30 algorithms introduced between 1977 and 2022. Yildiz *et al.* [92] proposed a taxonomy that groups the algorithms according to split type, approach to multi-class problems, how they find the weight coefficients w, how they find the split point v, branching factor, and split evaluation function. These are additional design decisions to consider. We have already discussed the branching factor, approach to multi-class problems, and split evaluation function in Section 2.1. We now discuss the rest of the design decisions and include the feature selection strategy to be used, if any:

1'. How to generate candidate splits In Section 2.1, we described this as one of the design decisions common to all DTs. When multivariate splits are considered, we must specifically decide how to search for the linear split's weight coefficients and split point.

  1.1. Search for w. The search for the weight coefficients w can be either analytical or iterative. Analytical algorithms generally consider the distribution of classes

  1.2. Search for v. The search for the split point v can also be analytical or iterative.

6. Split type. There are three possible split types: univariate, multivariate with linear combinations, and multivariate with non-linear combinations. To be considered an MDT, the algorithm must search for feature combinations. However, some MDT induction algorithms use different split types in different nodes.

7. Feature selection. MDT induction algorithms use or do not use feature selection for multivariate splits. The algorithms that use feature selection find multivariate splits using subsets of features. Most feature selection algorithms rely on a greedy search. For example, Sequential Forward Selection (SFS) begins with an empty set of features $F' = \emptyset$; then, it adds a feature one at a time, provided that a split improves the evaluation function when using the feature in conjunction with all features already in F'. Brodley *et al.* [17] describe other prominent feature selection algorithms used in MDTs.

Our recent survey on MDTs compared 19 algorithms in 57 databases [20]. The top-performing MDT was the Multi-class Hellinger Linear DT (MHLDT). MHLDT achieved this result by using both univariate and multivariate linear splits, using feature selection (Sequential Forward Selection), working directly with multi-class problems, and using an analytical method to find the weight coefficients w.

2.2.2 Model Trees

Model Trees aim to combine the advantages of decision trees and other classifiers, such as logistic regression models [46]. The most popular model tree is the Logistic Model Tree (LMT) [46], which builds a UDT using C4.5 and, while building the tree, fits logistic regression models in each node using the LogitBoost algorithm. The LogitBoost algorithm is an iterative algorithm that adds a feature to the logistic regression model or modifies an already present feature. LMT uses the iterative nature of LogitBoost to use the model of a parent node as an initial solution, so logistic models in a parent node are refined in children using only the objects falling in each child. Finally, LMT prunes the tree through Cost-Complexity pruning, which takes a trade-off between tree size and misclassification rate. The resulting tree classifies objects using the regression models at the leaves instead of the majority class.

The first design decision introduced by Model Trees is whether or not to use a classifier at the leaves. The second design decision is what classifier to use. When first published, the classification performance of LMT was better than that of MDTs and similar to that of Gama's FT, thanks to the logistic regression models generated by the LogitBoost algorithm. This makes for a strong case for using models at the leaves and using LogitBoost to generate the models.

8. Use leaf models. We must decide whether to use leaf models and, if so, how to generate them. When using leaf models, we assume that a logistic regression model is built with LogitBoost, since it has shown the best classification performance.

The similar performance between the first implementation of Gama's FT and LMT may discourage us from building the more complex FTs. However, the current implementation of Gams's FT in Weka achieves better classification performance than LMT, as we will show in Section 5.2.

2.3 Functional Tree construction

Functional Trees (FTs) combine the advantages of MDTs and model trees by allowing both multivariate splits and other classification models at the leaves. The first and only FT was presented by Gama [34]. This section describes the general steps for building a Functional Tree in Algorithm 2.2.

When building FTs, we need to take into account all design decisions considered for building general DTs, MDTs, and MTs. We have not identified any design decision specific to FTs.

The first difference between DT construction in Algorithm 2.1 and FT construction in Algorithm 2.2 is that a classification model, such as a logistic model with LogistBoost, is built at each node in line 5. The second difference is that we generate multivariate splits in lines 10 to 15, possibly using feature selection. The feature selection algorithms usually use the evaluation of the best split found so far, $\mathcal{S}^*$, and the current candidate splits S to decide when to stop.

Algorithm 2.2 General algorithm for building a Functional Tree.

Input:

$\mathcal{N}_t$**.** Node t.

$\mathbf{D_t}$**.** The objects at node $\mathcal{N}_t$.

`eval`$(\mathbf{D_t}, \mathcal{S})$**.** Split evaluation function, which evaluates the split $\mathcal{S}$ in node $\mathcal{N}_t$. The return value is a non-negative real number.

`FeatureSelection`(F, F')**.** Feature selection algorithm that returns a subset of features from F, given that a subset of features F' has already been selected. If an algorithm does not use feature selection, we assume that this function returns F when $F' = \emptyset$, and $\emptyset$ for any other value of F'.

```
BuildTree($\mathbf{D_t}$, $\mathcal{N}_t$, eval):
 1: if $\mathcal{N}_t = \emptyset$ then
 2:     $\mathcal{N}_t = \texttt{root}$
 3:     $\mathbf{D_t} = \mathbf{D}$
 4: end if
 5: $\mathcal{LM}_t = \texttt{LogitBoost}(\mathcal{N}_t)$        ▷ Build a logistic regression model.
 6: if a stop condition is met then
 7:     return $\mathcal{N}_t$
 8: end if
 9: $\mathcal{S}^* = \emptyset$
10: $\mathcal{S} = \emptyset$
11: $F' = \emptyset$
12: while $F' = \texttt{FeatureSelection}(F, F')$ do        ▷ Loop while the feature selection function
    returns candidate feature subsets.
13:     $S = \texttt{GenerateCandidates}(\mathbf{D_t}, \mathrm{F}')$        ▷ Generate multivariate candidate splits with
    features $F'$.
14:     $\mathcal{S}^* = \arg\max_s \texttt{eval}(\mathbf{D_t}, s), s \in S \cap \{\mathcal{S}^*\}$
15: end while
16: $N_l, N_r = \texttt{partition}(\mathbf{D_t}, \mathcal{S}^*)$
17: BuildTree($\mathbf{D_l}$, $N_l$, eval)
18: BuildTree($\mathbf{D_r}$, $N_r$, eval)
19: return $N_t$
```

2.4 Why there are no new FT algorithms?

In a recent survey, we gave an overview of 37 MDT induction algorithms published between 1977 and 2022 [20]. Although MDT induction algorithms have been introduced in recent years, no new FT has been introduced after Gama's FT. In Table 2.1, we show a timeline of publications for DTs with linear combinations. Although many MDTs have been published after Gama's FT, no new FT algorithms has been introduced.

One possible reason for the lack of new FTs is that researchers have decided that their disadvantages outweigh the advantage of improved classification performance over MDTs. We do not believe this to be the case, as the main disadvantage of FTs compared to MDTs is that they add complexity by adding linear combinations at the leaves that may include many features.

Year	Algorithm
1977	Friedman [31]
1984	CART-LC [15]
1991	LMDT [17]
1992	CTNNFE [35]
1993	SADT [37]
1994	OC1 [68]
1997	QUEST [52]
1998	BMDT [50]
1999	Ltree, Lgtree & Qtree [33]
1999	APDT [78]
2000	Dipolar [13]
2000	FAT & MOC1 [8]
2001	CRUISE [44]
2001	Omnivariate [91]
2003	LDTS [47]
2004	Gama's FT [34]
2005	LMT [46]
2005	SURPASS [48]
2005	LDT [92]
2008	Cline [3]
2009	GUIDE [53]
2009	Geometric [64]
2010	VDT & CDT [10]
2011	oRF [65]
2013	FDT [56]
2014	HBDT [80]
2015	HHCART [88]
2015	Zhag's MPSVM [93]
2017	Optimal [9]
2017	OmniGa [61]
2017	SBT & PT [51]
2018	Efficient [85]
2019	DTsvm [69]
2019	MHLDT [18]
2019	HHCART(G) [89]
2020	BDTKS [86]

TABLE 2.1: Publication timeline of DTs with linear combinations. We have highlighted Gama's FT and LMT, the rest of the algorithms are MDTs.

However, most MDT induction algorithms do not include feature selection for the splits with linear combinations, in which case the complexity of the MDTs is equivalent to an FT.

A more plausible reason that there are no new FTs is that they are not widely known in the community. This may be due to a problem of fragmentation in DT literature identified by Rusch *et al.* [72] where many authors that propose DT algorithms are not aware or choose to ignore similar works. In Figure 2.4, we show that authors compare new MDTs against at most a couple of other ones; the most comprehensive study, compares their MDT against five others.

Furthermore, only CART-LC, OC1, QUEST, and LMDT are used as a reference for comparison more than twice. Note that most MDTs have never been used in any experimental comparison. Furthermore, the MDTs were not compared against Gama's FT.

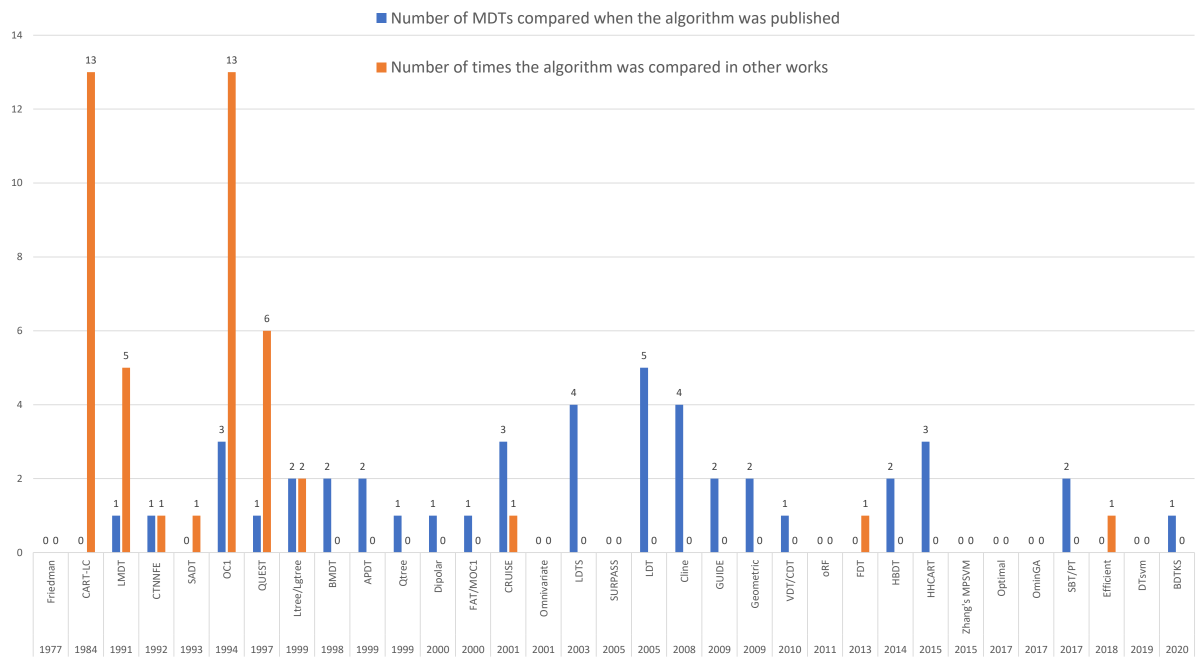

FIGURE 2.4: Comparisons between MDT algorithms. For each algorithm, the blue bar represents the number of rival algorithms the authors used in their comparison study. The orange bar represents the number of times the algorithm is compared in papers by other authors.

2.5 Chapter conclusions

In this chapter, we described basic DTs and how to build them. We identified five common design decisions in DTs: how to generate candidate splits, what split evaluation function to use, what stop conditions to use, how to approach to multi-class problems, and what pruning method to use.

We also described the family of DTs that uses feature combinations, which is composed of Functional Trees (FTs), Multivariate Decision Trees (MDTs) and Model Trees (MTs). We identified three additional design decisions for the family of DTs that uses feature combinations: what split types to use, what feature selection method to use, and what classification model to use at the leaves. Identifying these design decisions allows us to make informed decisions by searching the literature for the best choices for improving classification performance.

As mentioned before, multiple authors have shown that MDTs outperform UDTs. However, MTs have not been compared with the most recent top-performing MDT induction algorithms. Furthermore, the top-performing MT (LMT) has not been compared with the latest version of Gama's FT, which now also uses the LogitBoost algorithm that previously allowed LMT to maintain a similar performance. In the following section, we will show that LMT outperforms all MDTs tested in our recent survey and that Gama outperforms LMT.

Chapter 3

A protocol for a fair comparison of DT algorithms

Our main goal is to show that our new classifier, FT4cip, has better classification performance than the classifiers in the family of DTs that use linear feature combinations (described in Section 2.2). We also want to evaluate how our design decisions contribute to FT4cip's classification performance.

In Chapter 4, we introduce FT4cip, detailing each design decision. In order to evaluate the importance of each design decision to classification performance, we generate variations of FT4cip and compare them to the original algorithm. Each variation is related to a design decision. When comparing a variation to the original FT4cip with statistical tests, we can measure the importance of the design decision to classification performance.

In Chapter 5, we first demonstrate that there exists a hierarchy in classification performance, where functional trees surpass model trees, that in turn surpass multivariate trees. We divide our demonstration in two steps; first, we demonstrate that MTs surpass MDTs, and then we demonstrate that FTs surpass MTs.

Having demonstrated that FTs are at the top of the hierarchy, we compare FT4cip against the only previously published FT (Gama's FT). Our aim is to show that FT4cip has better classification performance than Gama's FT in general, and that FT4cip outperforms Gama's FT in class imbalance problems. We describe with detail the experiments carried out in Section 3.1.

In this work, we only need to compare classifiers in multiple databases pairwise. We use the Bayesian signed-rank test described by Benavoli *et al.* [7]. We describe this test in more detail in Section 3.2. One advantage of this test is that it considers the effect size (i.e., the magnitude of the difference in classification performance). A second advantage is that it gives actual probabilities of when an algorithm outperforms the other or when the algorithms are equivalent.

To make our comparisons statistically sound, we have compared our algorithms in 110 publicly available databases. However, due to the nature of some algorithms, we are forced to make some comparisons in subsets of databases. Section 3.3 describes the 110 databases and the subsets used in our experiment.

Finally, Section 3.4 describes the Area Under the ROC curve (AUC) evaluation measure, which we will use to measure classification performance. We show how to calculate the AUC from a confusion matrix and justify its use.

3.1 Experiments

When comparing a pair of algorithms, we execute them in each database $d \in \mathbf{D}$ using 5-fold Distribution Optimally Balanced-SCV (DOB-SCV). The k-fold DOB-SCV [66] is an alternative to the standard Stratified k-fold cross-validation (SCV). The standard SCV only places an equal number of samples in each fold, while the DOB-SCV additionally tries to keep the data distribution as similar as possible. Lopez *et al.* [55] suggest using k-fold DOB-SCV instead of k-fold cross-validation to avoid having different distributions between testing and training databases. For each algorithm execution $a \in \mathbf{A}$ in database $d \in \mathbf{D}$, we obtain the AUC (see Section 3.4) for each fold and calculate the mean AUC over the five folds.

For each experiment, we take each pairing of algorithms $(a_i, a_j), i \neq j$ considered in the experiment and apply the Bayesian signed-rank test (see Section 3.2) for the subset of databases considered in the experiment. The test gives three probabilities as a result: the probability that a_i is practically better than a_j, in other words, the probability of a_i winning; the probability that a_j is practically better than a_i, in other words, the probability of a_j winning or a_i losing; and the probability that a_i and a_j are practically equivalent, in other words, the probability of a tie.

3.1.1 Experiment 1. Evaluating FT4cip design decisions

The objective of our first experiment is to show that all design decisions behind FT4cip contribute to classification performance, in terms of AUC, or do not reduce the performance with the advantage of keeping the model or algorithm simple.

We created modified versions of FT4cip by changing one of the design decisions to match LMT or Gama. We compare each modified version of FT4cip with the baseline version using the Bayesian signed-rank test in the 110 databases.

When the baseline has better performance than the variation, it means that the design decision we took contributes to improving classification performance. When the baseline and variation have equivalent performance, the choice for our design decision is justified by keeping the model or algorithm simple. The probability that a variation has practically better performance than the baseline is always zero.

3.1.2 Experiment 2. Demonstrating that Model Trees outperform Multivariate Decision Trees $MDT \subset MT$

The objective of this experiment is to show that MTs have better classification performance than MDTs ($MDT \subset MT$). To do so, we show that the Logistic Model Tree (LMT) has better classification performance than the 19 MDTs compared in our recent survey [20]. LMT is compared against each MDT using the Bayesian signed-rank test in a subset of 57 or 40 numerical databases.

We selected the subset of 57 numerical databases because many MDT implementations cannot handle nominal attributes. In addition, since some MDT implementations crashed when running some of the 57 numerical databases, we further reduced the number of databases to 40 for those algorithms.

Unfortunately, we have found it difficult to assess why a specific implementation failed in one database. Some algorithms do not have publicly available source code, and the errors were not ones handled by the developers that could provide some helpful message.

3.1.3 Experiment 3. Demonstrating that Functional Trees outperform Model Trees $MT \subset FT$

The objective of this experiment is to show that FTs have better classification performance than MTs ($MT \subset FT$). To do so, we show that Gama's FT has better classification performance than the top-performing MT, the Logistic Model Tree (LMT). We compare the algorithms with the Bayesian signed-rank test using our whole set of 110 databases.

Let us say that we show that MTs have better classification performance than MDTs ($MDT \subset MT$) and FTs have better performance than MTs ($MT \subset FT$). It follows that FTs have the best classification performance among the family of DTs that use linear feature combinations. Then, if our objective is to maximize classification performance, we only need to compare new algorithms against Gama's FT since it is the only one of its type.

3.1.4 Experiment 4. Demonstrating that FT4cip has better classification performance than Gama's FT

The objective of this experiment is to show that our new classifier, FT4cip, has the best classification performance among the family of DTs with linear feature combinations. After carrying out the previous experiments, we will have shown that Gama's FT is the top-performing classifier of the family. Therefore, we only need to compare FT4cip against Gama's FT. We compare FT4cip against Gama's FT with the Bayesian signed-rank test using our whole set of 110 databases.

Additionally, we show that the performance of FT4cip is even better in databases with class imbalance. To do so, we compare FT4cip against Gama's FT in a subset of imbalanced databases using the Bayesian signed-rank test.

3.1.5 Experiment 5. Comparing FT4cip and Gama's FT in imbalanced databases

The objective of this experiment is to show that FT4cip has particularly better performance in class imbalance problems. In order to do so, we divide our databases in two groups according to their degree of class imbalance. The first group, the balanced databases, consists of databases with up to two objects of the majority class per object of the minority class. The second group, the imbalanced databases, is composed of databases with more than two objects of the majority class per object of the minority class.

We compare FT4cip against Gama's FT with the Bayesian signed-rank test in the subsets of balanced and imbalanced databases. The test will show that FT4cip has better classification performance in the imbalanced databases.

3.2 Bayesian signed-rank test

Benavoli *et al.* [7] describe the different Bayesian tests for comparing classifiers in multiple databases in a tutorial. When comparing two classifiers, instead of a p-value, the Bayesian tests return the distribution of the mean difference of classification performance (AUC in this case) between the two classifiers. To make automatic decisions, Benavoli *et al.* [7] suggest using a lower limit for the probability, such as 0.95.

To compare two algorithms in multiple databases, we used the Bayesian signed-rank test described in the tutorial by Benavoli *et al.* [7]. This test is the Bayesian counterpart to Wilcoxon's test. The Bayesian signed-rank test has two advantages. First, the test gives us actual probabilities of either algorithm outperforming the other and the probability of the classifiers being equivalent. The second advantage is that the test takes into account the effect size.

To understand this test, we briefly describe how to compare two classifiers in a single database. The Bayesian tests described by Benavoli *et al.*[7] are based on three hypotheses: that classifier A is practically better than B, that the classifiers are practically equivalent, and that classifier B is practically better than A. The Bayesian correlated t-test is used to calculate the probabilities of the hypotheses for a specific database.

The Bayesian tests calculate three probabilities: θ_l is the probability that classifier A is practically better than B, θ_r is the probability that classifier B is practically better than A, and θ_e is the probability that the classifiers are practically equivalent. The probabilities $\theta_l, \theta_e, \theta_r$ correspond to the integral of the distribution on different intervals: the region $(-\infty, -r)$, where classifier A is practically better than B; the region (r, ∞), where classifier B is practically better than A; and the region $[-r, r]$, where the classifiers are practically equivalent. The interval $[-r, r]$ is known as the region of practical equivalence (rope). Benavoli *et al.* [7] use $r = 0.01$ for accuracy; we will use the same value for AUC, given the similarity of the measure for balanced databases.

We use the Bayesian signed-rank test to compare the classifiers on multiple databases. For this test, a distribution of the probabilities $\theta_l, \theta_e, \theta_r$ is computed by Monte Carlo sampling. For a given sample, there is a bias towards $\theta_i \in \{\theta_l, \theta_e, \theta_r\}$ if $\theta_i > \max(\theta_j, \theta_k)$, with $\theta_j \in \{\theta_l, \theta_e, \theta_r\}$, $\theta_k \in \{\theta_l, \theta_e, \theta_r\}$ and $\theta_i \neq \theta_j \neq \theta_k$. For example, if all our samples have $\theta_l > \max(\theta_r, \theta_e)$, we conclude with a probability of 1 that classifier A is practically better than classifier B.

We must take care with the interpretation of the Bayesian signed-rank test results. That classifier B is practically better than classifier A with a probability of 1 does not imply that the difference in AUC between classifiers B and A is always greater than 0.01. Instead, this means that the probability θ_r is always greater than both θ_l and θ_e; in other words, there is always a bias towards classifier B winning.

Let us say we conclude that classifier B is practically better than classifier A with a probability equal to 1. This conclusion does not necessarily mean that the difference in AUC between classifier B and A is always greater than 0.01. Instead, this means that the probability θ_r is always greater than both θ_l and θ_e; in other words, there is always a bias towards classifier B winning.

Benavoli *et al.* [7] visualize $\theta_l, \theta_e, \theta_r$ for each sample using a simplex with vertices $\{(1,0,0), (0,1,0), (0,0,1)\}$. In Figure 3.1, we show one for a comparison between the classifiers CRUISE and CL2 on a subset of databases with a high number of features. If a point falls in a vertex, it has $\theta_i = 1$ for the corresponding hypothesis i; in the figure, a point in the left corner is a sample where the

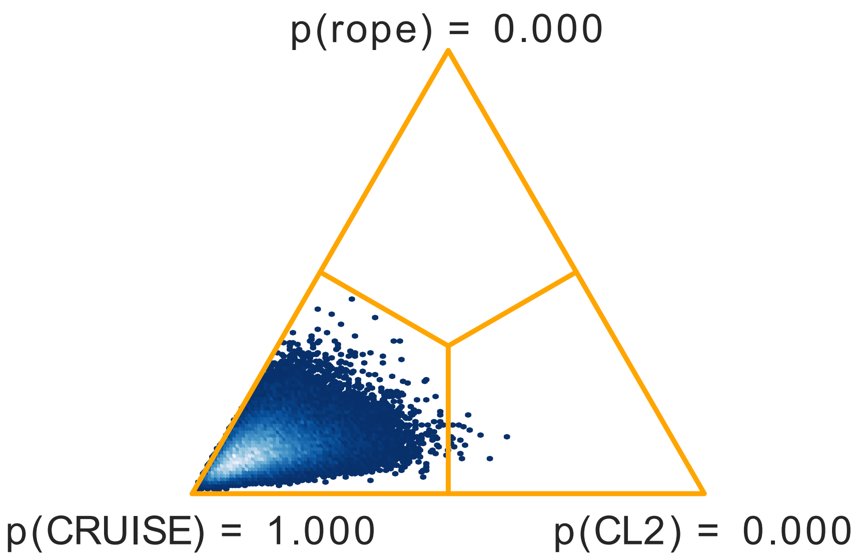

FIGURE 3.1: Example of Bayesian signed-rank test. Comparison between CRUISE and CL2 in a subset of the 57 databases with a high number of features.

difference in AUC between CRUISE and CL2 is greater than 0.01 with a probability equal to 1. There are three regions limited by $\theta_i > \max(\theta_j, \theta_k)$, so the left region corresponds to the case where there is a bias towards CRUISE. In each corner, we show the proportion of samples falling in the corresponding region; since almost all samples fall in the region where CRUISE is better, we have $p(CRUISE) \approx 1$. Some samples fall in the region where CL2 is better; however, the proportion of those samples is smaller than 1×10^{-3}.

3.3 Databases

We have obtained 110 databases from the UCI repository [25]. The databases are diverse, with varying numbers of instances, features, classes, and degrees of imbalance, with or without missing values, and with or without nominal features. We give a complete list of databases in Appendix A.

As described in Section 3.1.2, our survey compared all MDTs in a subset of 40 databases and 15 of 19 MDTs in 17 more databases. We summarize the key characteristics of the databases in Tables 3.1, 3.2, and 3.3, showing the distribution of databases in the different subsets used to compare algorithms in this work.

Since we evaluate the classifiers using k-fold cross-validation, we divide the databases into five folds using Distribution Optimally Balanced-SCV (DOB-SCV) [66]. DOB-SCV tries to keep the data distribution as similar as possible in each fold, which is essential when dealing with class imbalance because the degree of class imbalance in each fold will be close to that of the whole database.

No. of features	No. of databases (110)	No. of databases (57)	No. of databases (40)
< 10	27	19	17
$10 - 100$	72	33	20
> 100	11	5	3

Table 3.1: Distribution of the databases under consideration according to their number of features. We show the distribution for all databases, for a subset of 57 databases used when evaluating some MDTs, and 40 of 57 databases used to evaluate all MDTs.

No. of objects	No. of databases (110)	No. of databases (57)	No. of databases (40)
< 100	3	3	0
$100 - 1,000$	49	29	27
$> 1,000$	58	25	13

Table 3.2: Distribution of the databases under consideration according to their number of objects. We show the distribution for all databases, for a subset of 57 databases used when evaluating some MDTs, and 40 of 57 databases used to evaluate all MDTs.

No. of classes	No. of databases (110)	No. of databases (57)	No. of databases (40)
2	51	25	18
> 2	59	32	22

Table 3.3: Distribution of the databases under consideration according to their number of classes. We show the distribution for all databases, for a subset of 57 databases used when evaluating some MDTs, and 40 of 57 databases used to evaluate all MDTs.

Class imbalance	No. of databases (110)	No. of databases (57)	No. of databases (40)
< 2	40	29	24
$2 - 10$	33	22	11
> 10	37	6	5

Table 3.4: Distribution of the databases under consideration according to their degree of class imbalance, measured as the number of objects of the majority class for each object of the minority class. We show the distribution for all databases, for a subset of 57 databases used when evaluating some MDTs, and 40 of 57 databases used to evaluate all MDTs..

3.4 Evaluation measures

To select an evaluation measure, we must consider the class imbalance problem because many real-world databases are imbalanced. A database is highly imbalanced if it has many objects of one class compared to the rest. In such cases, measures such as accuracy may give a high evaluation value, close to 1, to a classifier that predicts that all objects belong to the majority class. This behavior is undesirable because we are often interested in correctly classifying many objects of the minority class.

We use the Area Under the ROC curve (AUC) to evaluate the classification performance of the classifiers. We use the AUC because many real-world databases are imbalanced, and the AUC is more insensitive to imbalanced databases than other evaluation measures, such as accuracy [74].

We can calculate the AUC from the confusion matrix obtained from the classifier applied to a testing database. The confusion matrix is a $k \times k$ matrix, where k is the number of classes. The rows correspond to actual classes, and the columns to predicted classes. We show an example of a confusion matrix in Table 3.5. From the row Iris-setosa, we conclude that 49 objects are classified correctly as Iris-setosa, and one object is classified incorrectly as Iris-versicolor. Similar remarks hold for the two other classes.

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	49	1	0
Iris-versicolor	0	47	3
Iris-virginica	0	2	48

Table 3.5: Example confusion matrix obtained by the J48 classifier [71] for the iris database.

Let $\mathbf{C}$ be a confusion matrix of size $k \times k$. Each cell c_{ij} in $\mathbf{C}$ counts the number of objects of class i classified as class j.

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{k1} & c_{k2} & \cdots & c_{kk} \end{bmatrix}$$

For discrete classifiers, the AUC measure for two classes is defined using recall and specificity [74]. Let (C_i, C_j), with $i \neq j$, be any pair of classes, where C_i denotes the Positive class, and C_j denotes the Negative class. The number of objects of the Positive class C_i correctly classified, c_{ii}, is the number of True Positives. The number of objects of the Positive class C_i incorrectly classified, c_{ij}, is the number of False Negatives. The number of objects of the Negative class C_j correctly classified, c_{jj}, is the number of True Negatives. The number of objects of the Negative class C_j incorrectly classified, c_{ji}, is the number of False Positives.

The Recall is then defined as the proportion of objects of the positive class correctly classified:

$$\mathtt{r}_{ij}(\mathbf{C}) = \frac{c_{ii}}{c_{ii} + c_{ij}} \tag{3.1}$$

Specificity is defined as the proportion of objects of the negative class correctly classified:

$$\mathtt{sp}_{ij}(\mathbf{C}) = \frac{c_{jj}}{c_{ji} + c_{jj}} \tag{3.2}$$

Finally, the AUC for two classes i, j is defined as the average between recall and specificity:

$$\texttt{auc}_{ij}(\mathbf{C}) = \frac{\texttt{r}_{ij}(\mathbf{C}) + \texttt{sp}_{ij}(\mathbf{C})}{2} \tag{3.3}$$

To extend the definition of AUC to multi-class problems, we take the recommended one versus the others approach [74]. This approach consists of averaging the AUC of all possible pairs of classes as follows:

$$\texttt{auc}(\mathbf{C}) = \frac{1}{\binom{k}{2}} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \texttt{auc}_{ij}(\mathbf{C}) \tag{3.4}$$

3.5 Chapter conclusions

In this chapter, we presented our protocol for making a fair comparison of DT algorithms. We described the five experiments made in this work. In all experiments, we only make pairwise comparisons of the classifiers, so we use the Bayesian signed-rank test to compare them.

With the first experiment, we show that all design decisions taken for FT4cip contribute to improving classification performance, or to keep a simple model or algorithm.

The objective of the second and third experiments is to identify the relative performance of the classifiers that use linear feature combinations. We show that functional trees surpass model trees, that in turn surpass multivariate trees.

Finally, our fourth experiment shows that, in general, FT4cip has better classification performance than Gama's FT. The fifth experiment shows that FT4cip performs even better than Gama's FT in class imbalance problems.

Chapter 4

Improving Functional Trees: The Functional Tree for class imbalance problems

In this chapter, we introduce our new functional tree, the Functional Tree for class imbalance problems (FT4cip). We begin by describing the design decisions for building FT4cip in Section 4.1, and show the full algorithm in Section 4.2.

In Section 4.3, we evaluate our design decisions by showing that changing any of them to match the best-performing classifiers, MHLDT, LMT, or Gama's FT, results in a reduction in classification performance. Next, we evaluate the runtime of FT4cip in Section 4.4.

4.1 Design decisions for building FT4cip

In each of the following subsections, we describe each of the eight design decisions for building Functional Trees identified in Chapter 2. As we recall, we need to define how to generate candidate splits, which evaluation function to use, which stop conditions to use, how to approach multi-class problems, which (if any) pruning method to use, which split types are allowed, which (if any) feature selection method to use, and which (if any) classifier to use at the leaves.

4.1.1 Generating candidate splits

We use the multi-class version of Fisher's linear discriminant (`MFLD`) [18] to split a node because it is used by the top-ranked MDT (MHLDT) on our survey [20]. Additionally, `MFLD` allows us to work directly with multi-class problems.

`MFLD` gives $K - 1$ vectors that maximize class separability, where K is the number of classes [32]. The value of the function which measures class separability, when maximized, is equal to the sum of the corresponding eigenvalues. This means that the eigenvector with the largest eigenvalue, called the dominant eigenvector, is the one that contributes the most to the maximization of the function.

To generate a split with `MFLD`, we project the objects onto the dominant eigenvector, sort them, and find the best split point according to the split evaluation measure. We decided to use the dominant eigenvector as w because when projecting the objects onto it, the classes can be divided better than when projecting the objects onto the rest of the eigenvectors.

After projecting all objects onto w, we sort them and use exhaustive search to find the split point v. We test values for v where two contiguous objects have different classes.

4.1.1.1 Nominal features

To generate splits using nominal features, we choose to do binary splits with a value and complement approach. We generate a candidate split for each possible feature value, with all objects matching the feature value going to the left node and those that do not match going to the right node. For example, with the split $(color = blue, color \neq blue)$, all blue objects go to the left child, and non-blue objects go to the right child. With this decision, we avoid generating unnecessary child nodes when a single value of a nominal feature, but not the rest of the values, helps to discriminate between classes.

The key difference is that LMT uses multi-way splits, creating a child node for each possible value of a nominal feature, and FT4cip uses binary splits.

4.1.2 Split evaluation function

We use Twoing [15] to evaluate candidate splits since it considers class imbalance. As shown in a 2021 survey on split evaluation measures [38], Twoing was the top-ranked split evaluation measure for AUC.

Given a split that divides objects from $\mathbf{C}$ classes into left L and right R nodes, the proportion of objects falling in the left node is p_L and the proportion of objects falling in the right node is p_R. The proportion of objects with class $c \in \mathbf{C}$ falling in the left node is $p_{c,L}$ and the proportion of objects with class $c \in \mathbf{C}$ falling in the right node is $p_{c,R}$. The split evaluation function evaluates split s at node t. The Twoing split evaluation function is defined as:

$$\theta(s,t) = \frac{p_L p_R}{4}[\sum_{c \in \mathbf{C}} |p_{c,L} - p_{c,R}|]^2 \tag{4.1}$$

By maximizing this function, we attempt to have all objects of a class fall in one of the child nodes since the term $|p_{c,L} - p_{c,R}|$ is maximized when the proportion of objects of class c all fall in the left or right node, and the number of objects assigned to each child node is similar because the term $p_L p_R$ is maximized when assigning the same number of objects to each child. For each class, the term $|p_{c,L} - p_{c,R}|$ takes values in the same range $[0, 1]$ regardless of the number of objects of each class, making the Twoing split evaluation function robust to class imbalance.

The key difference is that LMT and Gama's FT use Information Gain as a split evaluation function, while FT4cip uses Twoing.

4.1.3 Stop conditions

Since the stopping conditions did not significantly affect classification performance of FT4cip when pruning the trees, we decided to keep them as simple as possible. We only stop growing the tree if a node is pure. During the pruning procedure, the size of the tree will be reduced.

4.1.4 Approach to multi-class problems

We want to work directly with multi-class problems, because transforming them to two-class problems by using heuristics may result in undesirable class groups. For example, the classes may be grouped in a way that a multivariate split is needed when an univariate split suffices. In contrast to using a heuristic, an exhaustive search of class groups is more time consuming.

FT4cip is able to work directly with multi-class problems thanks to the use of the multi-class version of Fisher's linear discriminant (`MFLD`). LMT and Gama's FT can also work with multi-class problems.

4.1.5 Pruning

We use a novel version of the Cost-Complexity pruning that optimizes AUC to prune the tree. We use this method because the original Cost-Complexity pruning [15] achieves lower AUC values than using no pruning at all, as we show in Section 4.3.6. After all, it optimizes accuracy, which is inappropriate for class imbalance problems. In Section 4.3.6, we show that Cost-Complexity pruning optimizing AUC does not degrade the classifier's performance.

Cost-Complexity pruning estimates the error of a tree as the error in the training database plus a factor α times the subtree size [14]. To optimize the parameter α describing the trade-off between tree size and misclassification rate, five trees are built and pruned with candidates for α through 5-fold cross-validation in a training database. The candidates for α are obtained with a search procedure that computes the distinct values of α that reduce the tree size. The value of α used to prune the tree built with the whole training database is selected by minimizing the average misclassification rate on the five folds.

We noticed that the original Cost-Complexity pruning algorithm reduces classification performance in terms of AUC. Therefore, we propose the AUC-optimizing Cost-Complexity pruning that changes the objective of minimizing the misclassification rate to maximizing AUC.

The key difference is that LMT uses the original Cost-Complexity pruning, which optimizes accuracy, while our modified version optimizes AUC. Gama uses the C4.5 pruning method, which also optimizes accuracy.

4.1.6 Split type

FT4cip can generate univariate or linear multivariate splits. When the best split of one feature uses a nominal feature, the split is univariate. Otherwise, the split may be univariate or linear multivariate. In contrast, LMT only allows univariate splits.

4.1.7 Feature selection

We use Sequential Forward Selection (SFS) as feature selection method. We chose SFS because it is also used by the top-ranked MDT (MHLDT) in our survey [20]. Furthermore, it is easier to find short linear combinations by using SFS, which makes the tree easier to interpret. When MHLDT was used in a contrast pattern-based classifier [18], most splits were univariate or had only two features. Gama's FT does not apply a feature selection method.

4.1.8 Classifier for the leaves

We use the same logistic classifier as LMT and Gama's FT to label the objects falling in a leaf. The logistic models allow LMT to achieve better classification performance than all MDTs from a recent survey [20], as we show in Section 5.1.

The algorithm that trains the logistic models is called LogitBoost [46]. First, a logistic regression model is built for the root node using LogitBoost. Then, the children node refine the parent's logistic model by running additional iterations of the LogitBoost model using only the objects in the node. There is no difference in the classifier at the leaves; we use the same one as LMT and Gama.

4.2 FT4cip training algorithm

In this section, we show how to build our new functional functional tree, FT4cip, in Algorithm 4.1. We describe the general steps for building the tree as comments and give more detailed pseudocode to make it replicable. For readers seeking to use or modify FT4cip, we also provide the implementation as a Weka package and the source code. We use the following notation in the pseudocode of Algorithm 4.1.

$\mathbf{D}$. Database as an array of size $n \times m+1$. The first m columns correspond to non-class features; the additional feature is the class.

n. The number of objects in the database.

m. The number of features in the database.

F. The set of features in the database.

$\mathcal{N}_t$. Node t.

$\mathbf{D_t}$. The objects at node $\mathcal{N}_t$.

$\mathcal{S}$. A candidate split.

$\mathcal{S}_f$. Univariate split at node $\mathcal{N}_t$ with feature $f \in F$.

$\mathcal{S}_{F'}$. Multivariate split at node $\mathcal{N}_t$ with features $F' \subseteq F$.

$\texttt{split}(\mathbf{D_t}, F', \texttt{eval})$. A split function that splits the objects in $\mathbf{D_t}$ using features from F'. The split function uses the split evaluation function `eval` to select between candidate splits. The function returns a split $\mathcal{S}_t$, which may be a univariate split $\mathcal{S}_f$ when $F' = \{f\}$, or a multivariate split $\mathcal{S}_{F'}$.

$\texttt{eval}(\mathbf{D_t}, \mathcal{S})$. Split evaluation function, which evaluates the split $\mathcal{S}$ in node $\mathcal{N}_t$. The return value is a non-negative real number.

$\texttt{partition}(\mathbf{D_t}, \mathcal{S})$. A function that generates the left and right children nodes $\mathcal{N}_l, \mathcal{N}_r$ and assigns to them the subset of objects from $\mathbf{D_t}$ according to split $\mathcal{S}$.

$\texttt{SFS}(\mathbf{D}, F', \mathcal{N}_t, \texttt{eval}, \texttt{split})$. Sequential Forward Selection algorithm.

Algorithm 4.1 FT4cip induction.

Input:

$\mathcal{N}_t$. Node t.

$\mathbf{D_t}$. The objects at node $\mathcal{N}_t$.

`Twoing`$(\mathbf{D_t}, \mathcal{S})$**.** Split evaluation function, which evaluates the split $\mathcal{S}$ in node $\mathcal{N}_t$. The return value is a non-negative real number.

`SFS`(F, F')**.** Sequential Forward selection algorithm that returns a subset of features from F, given that a subset of features F' has already been selected.

`MFLD`**.** Multi-class Fisher's Linear Discriminant split function.

We call `FT4cip`$(\mathbf{D}, F, \emptyset)$ to build a tree.

After building the FT4cip tree, we prune it using AUC-optimizing Cost-Complexity pruning.

`FT4cip`$(\mathbf{D_t}, F, \mathcal{N}_t)$:

```
1:  if $\mathcal{N}_t = \emptyset$ then
2:      $\mathcal{N}_t = \texttt{root}$                                ▷ Initialize the root node
3:  end if
4:  $\mathcal{LM}_t = \texttt{LogitBoost}(\mathcal{N}_t)$              ▷ Build a logistic regression model.
5:  if the node is pure then                                   ▷ We only stop if the node is pure.
6:      return $\mathcal{N}_t$
7:  end if
8:  $S = \texttt{GenerateUnivariateCandidates}(\mathbf{D_t}, F)$.    ▷ Generate univariate splits.
9:  $\mathcal{S}^* = \arg\max_s \texttt{Twoing}(\mathbf{D_t}, s), s \in S$    ▷ Evaluate the splits and keep the best.
10: $f^* = \texttt{Feature}(\mathcal{S}^*)$                        ▷ Store the feature of the best split.
```

Continued on next page.

```
11: if $f^*$ is a numerical feature then              ▷ If the best split uses a numerical feature...
12:     $F' = \{f^*\}$
13:     while $F' = \texttt{SFS}(F, F')$ do      ▷ Generate subsets of candidate features of an increasing
    size, as long as there is an improvement in split quality.
14:         $S = \texttt{MFLD}(\mathbf{D_t}, \mathrm{F}')$                      ▷ Generate multivariate splits.
15:         $\mathcal{S}^* = \arg\max_s \texttt{Twoing}(\mathbf{D_t}, s), s \in S \cup \{\mathcal{S}^*\}$
16:     end while
17: end if
18: $N_l, N_r = \texttt{partition}(\mathbf{D_t}, \mathcal{S}^*)$
19: $\texttt{FT4cip}(\mathbf{D_l}, F, N_l)$
20: $\texttt{FT4cip}(\mathbf{D_r}, F, N_r)$
21: return $N_t$
```

In Section 2.3, Algorithm 2.2 gave a general algorithm for building Functional Trees. We notice that the general algorithm is very similar to the one of FT4cip. However, we now have a specific method for generating multivariate splits (`MFLD`), a specific evaluation function (`Twoing`), and a specific feature selection method (`SFS`).

Lines 1 - 8 of Algorithm 4.1 are mostly the same as the general algorithm, the only difference is that our only stop condition is that we stop when the node is pure. Next, we generate univariate candidate splits with each feature and keep the best in lines 9 - 10.

The next difference in FT4cip is at line 14, where we only begin to generate multivariate candidate splits if the feature used by the best univariate split is numerical. Otherwise, we just use the best split found, which uses a nominal feature.

4.3 Results of Experiment 1. Evaluating FT4cip design decisions

In this section, we evaluate the relative importance of the design decisions described in Section 4.1. To do so, we created modified versions of FT4cip by changing one of the design decisions. The only exception was removing both logistic models and pruning simultaneously in one variation; we did so because the resulting tree is better without pruning.

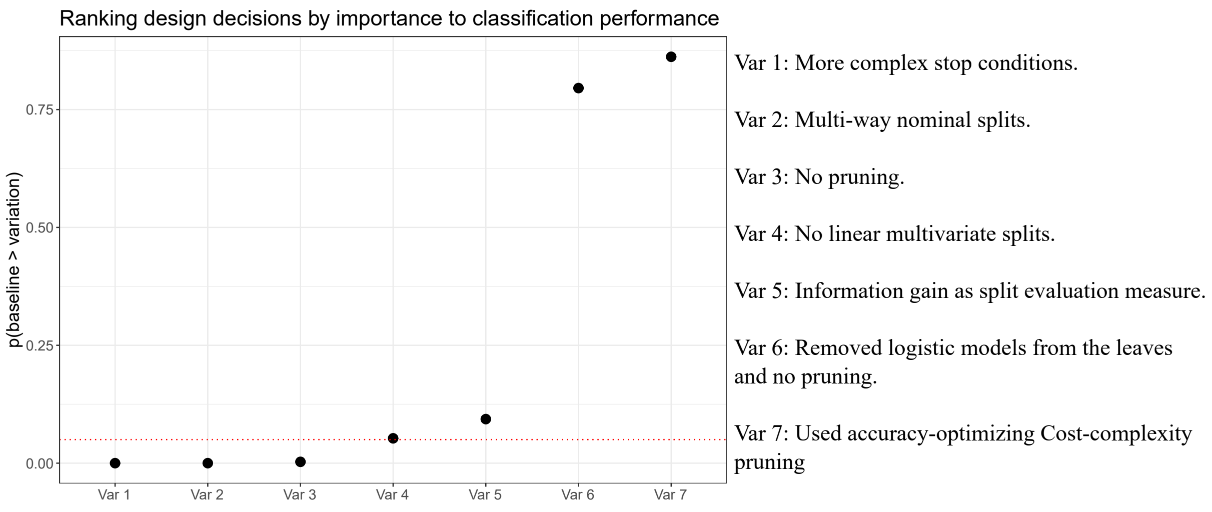

Figure 4.1: Evaluating the relative importance of FT4cip's design decisions. For each design decision described in Section 4.2, we generate a new version of FT4cip. We compare the variations against the baseline version using the Bayesian signed-rank test. We show the probability that the baseline version improves upon each variation. The dashed line at $p(baseline > variation) = 0.05$ helps to notice negligible improvements.

By ranking the modified versions of FT4cip by their probability of losing against the original version of FT4cip (baseline), we can evaluate the relative importance of the design decisions by how much they fall in the ranking. We show this ranking in Figure 4.1, together with the probability that the baseline is practically better than the variation $p(baseline > variation)$. It is sufficient to show this probability because, for all variations, we found that $p(variation > baseline) = 0$.

As an example, from Figure 4.1, since the version using the accuracy-optimizing Cost-Complexity pruning is at the bottom of the ranking, we can conclude that the design decision of using the AUC-optimizing Cost-Complexity pruning is the most important to classification performance. We can also see that the baseline optimizing AUC in pruning is better than the variation that optimizes accuracy in 89.8% of the cases. Conversely, simple stop conditions are the least important to classification performance, and the baseline and variation are practically equivalent. Even when the classification performance is practically equivalent, we can still show an improvement in other measures.

From the results of Figure 4.1, the relative importance for classification performance of our design decisions, from most to least important, is: Using an appropriate pruning method, using logistic models, using linear multivariate splits, using the proper split evaluation measure, using simple stop condition, and using binary splits for nominal features. Now, from Section 4.3.6 to 4.3.2, we give recommendations on each design decision.

4.3.1 Using simple stop conditions

The baseline and the version using LMT's stop conditions are practically equivalent. Since there is no improvement from using more complex stop conditions, we prefer simple stop conditions.

4.3.2 Using binary splits for nominal features

The baseline and the version using multi-way splits are practically equivalent. We prefer a fully binary tree because any multi-way split may be represented as a binary split. Furthermore, when a binary split is enough to perfectly separate one class, multi-way splits still create more than two children, making the tree more complex.

4.3.3 Using linear multivariate splits

The improvement of using linear multivariate splits is not as big as that of previous design decisions. However, we notice that the probability of improvement is not negligible (greater than 0.05), so we still recommend using linear multivariate splits.

4.3.4 Using the appropriate split evaluation measure

Using Information Gain, as LMT does, instead of Twoing results in a decreased classification performance for imbalanced databases. In general, we can see that the baseline version using Twoing is better in 10.5% of the cases, so we recommend using it instead of Information Gain.

4.3.5 Using logistic models

Without using logistic models, FT4cip transforms into an MDT. The FT is more complex because the logistic models use many features; however, the FT is better in 75% of the cases in terms of AUC. To solve the complexity issue, we suggest feature selection in logistic models as future work.

4.3.6 Using an appropriate pruning method

The version using no pruning is practically equivalent to the baseline; however, by using pruning, we can obtain smaller trees that are easier to interpret. Therefore, we recommend using an appropriate pruning method.

We emphasize using an appropriate pruning method because some methods may decrease classification performance. Specifically, the accuracy-optimizing Cost-Complexity pruning used by LMT is worse than using no pruning, according to the difference in ranks (see variations 3 and 7 in Figure 4.1). Since the probability of the baseline being practically better than the variation is 89.8%, we recommend using our AUC-optimizing Cost-Complexity.

4.4 Runtime

In terms of computational complexity, in the worst case, the Sequential Forward Selection Algorithm adds all features to the set of selected features, which involves comparing m candidate splits, then $m-1$ candidate splits, $m-2$, and so on until no candidate features remain. The number of comparisons is $\sum_{i=1}^{m} i = \frac{m(m-1)}{2}$, so the computational complexity is $O(m^2)$. However, given the early stop heuristic of SFS, the number of comparisons is often lower. For example, the average linear combination length of multivariate items from the MHLDT tree [18], which also uses SFS, in the same 110 databases used in this study is 1.5. Therefore, most of the time, the SFS algorithm selected splits with at most two features, which involves about $2m$ comparisons.

Regarding the computational complexity of FT4cip, at each node, we need to consider the logistic regression models and the multivariate splits built. The asymptotic complexity for building logistic regression models is $O(m^2 \cdot n)$, where n is the number of objects, and m is the number of features [46]. To select a candidate split given m features from SFS takes $O(m^3 \cdot n + m \cdot n \log n)$, which reduces to $O(m \cdot n \log n)$ for $m \ll n$. Overall, the computational complexity at a node is $O(m^3 \cdot n \log n)$ when considering the computational complexity of SFS.

In Figure 4.2, we show with a boxplot the training time distribution of FT4cip, LMT, Gama, and the variations of FT4cip described in Section 4.3. Likewise, Figure 4.3 shows the inference time distribution. Surprisingly, Gama is the classifier with the lowest training time, although LMT does not use linear splits.

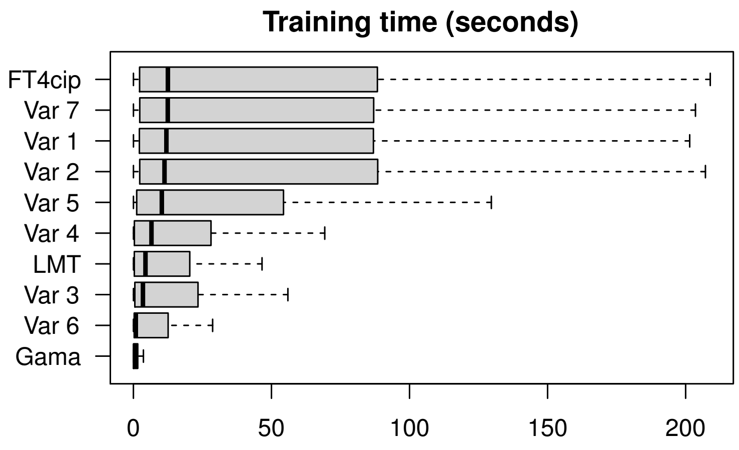

FIGURE 4.2: Training time, measured in seconds, of FT4cip, LMT, Gama, and the variations of FT4cip described in Section 4.3. The algorithms are sorted by their median runtime. The outliers were removed in order to visualize the key statistics shown in the boxplot.

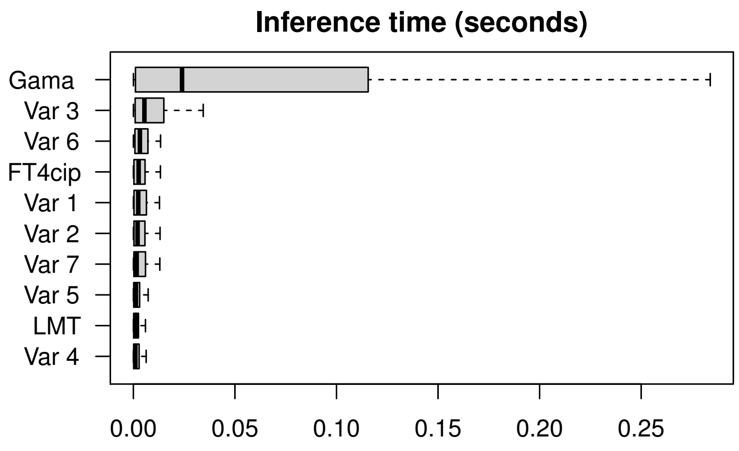

FIGURE 4.3: Inference time, measured in seconds, of FT4cip, LMT, Gama, and the variations of FT4cip described in Section 4.3. The algorithms are sorted by their median runtime. The outliers were removed in order to visualize the key statistics shown in the boxplot.

The most time-consuming algorithm is FT4cip. However, there is little difference between the runtime of FT4cip and the variations using more complex stop conditions, multi-way splits, and accuracy-optimizing Cost-complexity pruning.

Gama is the least time-consuming algorithm, followed by variations that do not use pruning. Therefore, we identify the Cost-complexity pruning algorithm as the most time-consuming step. Since we can see LMT and the FT4cip variation without multivariate splits are the next least time-consuming algorithms, the second most time-consuming step is building FT4cip multivariate splits. The third most time-consuming step is calculating the Twoing split evaluation function, as we see from the runtime drop when using Information Gain instead of Twoing.

4.5 Chapter conclusions

In this chapter, we presented our new functional tree, FT4cip. The key differences from previous works that contribute to the performance of FT4cip are how it generates candidate splits and the pruning method. FT4cip uses the multi-class version of Fisher's linear discriminant (`MFLD`) to generate candidate splits, together with Sequential Forward Selection as feature selection method; this allows FT4cip to work directly with multi-class problems and to generate splits with few features. Our novel pruning method, AUC-optimizing Cost Complexity pruning, does not reduce the classification performance by optimizing the AUC. Other differences simplified the algorithm or resulting tree without affecting classification performance.

The most time-consuming step in FT4cip is Cost-complexity pruning. There is little difference in runtime between the accuracy-optimizing and the AUC-optimizing Cost-complexity pruning. Since the FT4cip variation using accuracy-optimizing Cost-complexity pruning is the worst, we suggest optimizing AUC. Since Gama is the fastest algorithm, it may be worth exploring an AUC-optimizing version of its pruning method.

Although the second most time-consuming step is building multivariate splits with FT4cip, removing multivariate splits from FT4cip degrades its classification performance. We also prefer our multivariate splits over Gama's multivariate splits because we use feature selection, which results in shorter linear combinations and seems to be the reason for improved performance in databases with many features.

One limitation of FT4cip, shared with LMT and Gama, is that the logistic regression models generate linear combinations with many features, making them harder to interpret. Another limitation of LMT and FT4cip is that leaves always have a logistic model, while Gama simplifies the trees by tagging some leaves with the majority class during the pruning procedure.

We conclude that the following design decisions improved the performance of FT4cip: using our AUC-optimizing pruning algorithm, using logistic models at the leaves, using a split evaluation measure that considers class imbalance (Twoing), and using multivariate splits. The rest of the design decisions do not reduce classification performance and simplify the models by pruning, building binary trees, and using simple stop conditions.

Chapter 5

Comparing FT4cip against the DT family that uses linear combinations

This chapter aims to show that FT4cip has the best classification performance among the DTs in the family that uses linear combinations. In order to do so, we first establish an order in the performance of the DTs in the family. Then, we show that FT4cip outperforms Gama's FT [34], which has the best classification performance in the family.

We described the family of DTs with linear combinations comprising Multivariate Decision Trees, Model Trees, and Functional Trees in Section 2.2. However, the literature does not show that any of the three types of DT has better classification performance than the others. We will apply our evaluation protocol to compare MDTs, MTs, and FTs. As a result, we show that MTs outperform MDTs, and FTs have the best performance ($MDTs \subset MTs \subset FTs$).

Over 30 MDT induction algorithms were published between 1977 and 2022. However, due to lack of statistical comparison between the algorithms, their relative performance was unknown. We recently published a survey where we compared 19 MDT algorithms following different approaches to building MDTs [20]. With these results, we can properly compare the top-performing MDTs against the top-performing MT (LMT) [46] and Gama's FT [34].

Gama's FT has already been compared against LMT [46], showing no statistical differences in classification performance. However, the latest implementation of Gama's FT takes advantage of the LogitBoost algorithm used by LMT. Our statistical comparison will show that the latest implementation of Gama's FT does outperform LMT.

The chapter is organized as follow. We begin this chapter with Section 5.1, showing that the top-performing MT, the Logistic Model Tree (LMT), has better classification performance than all MDTs on our survey. These results indicate that MTs outperform MDTs ($MDT \subset MT$). Next, Section 5.2 shows that Gama's FT has better classification performance than LMT. Having shown that FTs outperform MTs ($MT \subset FT$) and also MDTs ($MDT \subset MT \subset FT$), we only compare FT4cip against Gama's FT. Section 5.3 compares the classification performance of FT4cip against Gama in all our databases, and Section 5.4 compares their classification performance in balanced and imbalanced databases.

5.1 Results of Experiment 2. Demonstrating that Model Trees outperform Multivariate Decision Trees $MDT \subset MT$

In this section, we show the results of applying our evaluation protocol to compare LMT against all MDTs from the experimental comparison of our survey. As mentioned in Section 3.1.2, executing some MDTs in some databases resulted in execution errors. Therefore, the comparison of MDTs was done in a subset of databases.

We compared 19 MDT induction algorithm in the review, which follow different strategies for generating multivariate splits. Table 5.1 lists eight algorithms and two families of algorithms, with their strategies for building MDTs. The first family of algorithms, Zhang's MPSVM, is composed of MPSVMlda, MPSVMpca, MPSVMparallel, MPSVMtikhnov, and MPSVMsubspace. Ths second family of algorithms, Cline, is composed of CLMIX, CLDA, CLM, CLLVQ, CL4, and CL2.

As we mentioned in Section 3.1.2, the MDTs were tested in 57 numerical databases. However, CART, OC1, LDT, and Omnivariate were only tested in a subset of 40 databases due to runtime errors.

Algorithm	Search of w	Search of v	Evaluation measure	Multi-class	Split type	Feature selection
QUEST [52]	Discriminant functions	Discriminant functions	-	Transforms the problem	Uni/Lin	-
CRUISE [44]	Discriminant functions	Discriminant functions	-	Works directly	Uni/Lin	-
LDT [92]	Fisher's discriminant	Fisher's discriminant	-	Transforms the problem	Lin	-
Cline [3]	Analytical	Analytical	-	Only two class	Lin	-
Zhang's MPSVM [93]	MPSVM	MPSVM	Gini index	Transforms the problem	Uni/Lin	-
MHLDT [18]	Fisher's discriminant	Exhaustive	Hellinger distance	Works directly	Uni/Lin	SFS
CART-LC [15]	Backfitting	Exhaustive	Impurity	Works directly	Lin	SBE[1]
OC1 [68]	Hill climbing	Exhaustive	Info Gain	Works directly	Uni/Lin	-
Omnivariate [91]	Perceptron	Perceptron	Impurity	Transforms the problem	Uni/Lin/Non	-
OCT [9]	MIO[2]	MIO	Missclassification	Works directly	Uni/Lin	MIO

[1] SBE: Sequential Backward Elimination

[2] MIO: Mixed-integer optimization

TABLE 5.1: MDT algorithms compared and their design decisions. We do not include stop conditions or pruning methods. The top six algorithms are analytical algorithms, and the bottom four are iterative algorithms.

Table 5.2 shows the results of comparing LMT against the MDTs using the Bayesian signed rank-test. We indicate if the comparison between LMT and a given MDT was done in the subset of 40 or 57 databases.

MDT	Number of databases	win	lose
CRUISE	57	0.92246	4.00E-05
MHLDT	57	0.97688	0.00628
QUEST	57	0.9996	0
MPSVMlda	57	0.99968	0.00032
MPSVMpca	57	0.99996	4.00E-05
CLMIX	57	0.99998	2.00E-05
CLDA	57	1	0
MPSVMparallel	57	1	0
CLM	57	1	0
MPSVMtikhnov	57	1	0
OCT	57	1	0
CLLVQ	57	1	0
MPSVMsubspace	57	1	0
CL4	57	1	0
CL2	57	1	0
CART	40	0.9973	0
OC1	40	0.99848	0.00034
LDT	40	0.99996	4.00E-05
Omni	40	1	0

TABLE 5.2: Bayesian signed rank-test results of comparing LMT against each classifier MDT. The second column shows if the comparison was done with all 57 numerical databases without missing values, or with a subset of 40 of those databases. The third column shows the probability of LMT outperforming the MDT in the row. Finally, the fourth column shows the probability that the MDT in the row outperforms LMT. The top algorithms are analytical, and the four bottom algorithms are iterative.

We notice that the probability that LMT has better classification performance than a given MDT is always greater than 0.92. Furthermore, the probability of any MDT outperforming LMT is always smaller than 0.0003. Therefore, we conclude that LMT, a Model Tree, has better classification performance than MDTs.

5.2 Results of Experiment 3. Demonstrating that Functional Trees outperform Model Trees $MT \subset FT$

In this section, we show the results of applying our evaluation protocol to compare Gama's FT against LMT, which is the best-performing MT. As described in Section 3.1.3, this experiment is carried out with all 110 databases.

Figure 5.1 shows the results of comparing Gama's FT against LMT using the Bayesian signed rank-test. Gama's FT has better classification performance than LMT in 94.2% of cases, and LMT only outperforms Gama's FT is of 0.3% of cases. Therefore, we conclude that Gama's FT has better classification performance than LMT.

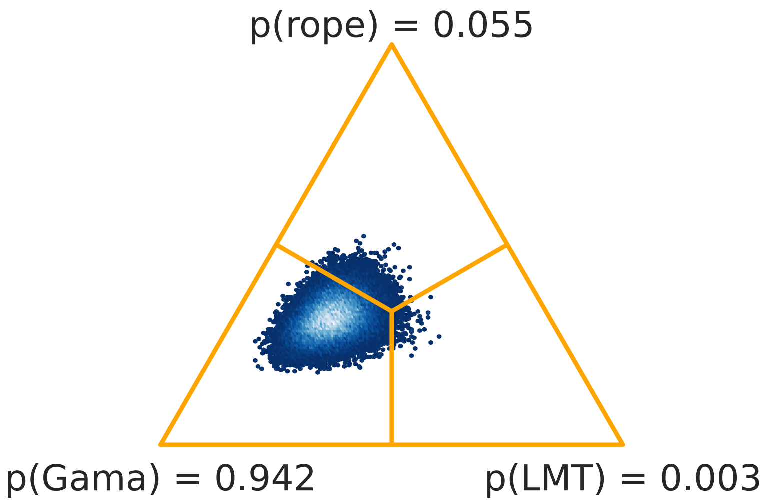

FIGURE 5.1: LMT vs. Gama: results of Bayesian signed-rank test. The mean difference of the AUC is visualized with 150,000 Monte Carlo samples plotted in barycentric coordinates. Brighter areas have more samples falling in them. According to the Bayesian signed-rank test, Gama is better in 94.2% of cases, and equivalent to LMT in 5.5% of cases.

5.3 Results of Experiment 4. Demonstrating that FT4cip has better classification performance than Gama's FT

Figure 5.2 shows the AUC distribution for FT4cip, Gama, and LMT in the 110 databases described in Section 3.3. Since we want to maximize AUC, a small box to the right side suggests good classification performance. FT4cip has the best classification performance in terms of the median AUC. We also notice that the first and third quartiles for FT4cip are higher than for the other classifiers. To confirm that FT4cip has better classification performance, we now show the results of the statistical tests.

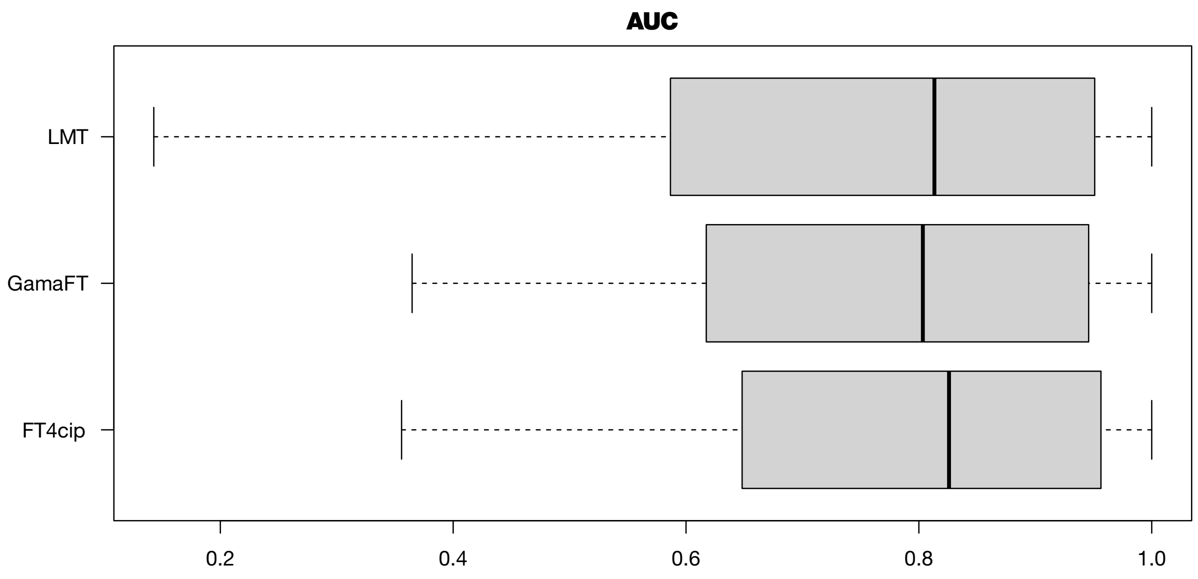

FIGURE 5.2: Boxplot showing the distribution of AUC for FT4cip, Gama's FT, and LMT.

Figure 5.3 compares FT4cip with Gama using the Bayesian signed-rank test. Here, we conclude that FT4cip is practically better than Gama in 27.7% of cases and practically equivalent in 72.3%.

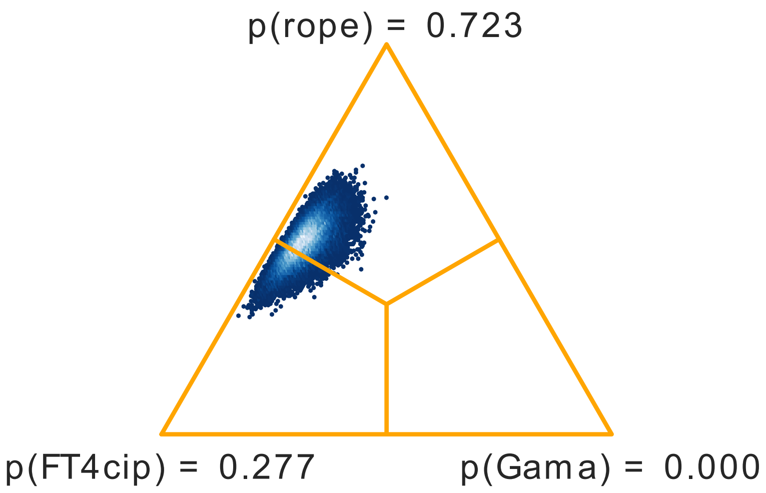

Figure 5.3: FT4cip vs. Gama: results of Bayesian signed-rank test. The mean difference of the AUC is visualized with 150,000 Monte Carlo samples plotted in barycentric coordinates. Brighter areas have more samples falling in them. According to the Bayesian signed-rank test, FT4cip is better in 27.7% of cases, and equivalent to Gama's FT in 72.3% of cases

Our main finding here is that FT4cip has practically better classification performance than Gama's FT according to the Bayesian signed-rank test. Since Gama's FT outperformed the rest of the DTs that use linear combinations, if the main concern is classification performance, we recommend using FT4cip.

5.4 Results of Experiment 5. Comparing FT4cip and Gama's FT in imbalanced databases

Since FT4cip is designed to deal with class imbalance problems, our hypothesis is that it will have better performance than Gama's FT in those problems. To confirm this, we compare FT4cip and Gama in balanced and imbalanced problems.

In Figure 5.4, we compare the performance of FT4cip against Gama's FT in subsets of databases according to their degree of class imbalance. The first subset of databases has up to two objects of the majority class per each of the minority class. The second subset of databases has more than two objects of the majority class per object of the minority class.

We notice that the probability of Gama's FT outperforming FT4cip is smaller than 0.001 in all cases. For databases with low imbalance, the FT4cip and Gama's FT are practically equivalent in 98.6% of cases. However, for databases with high imbalance, FT4cip is practically better

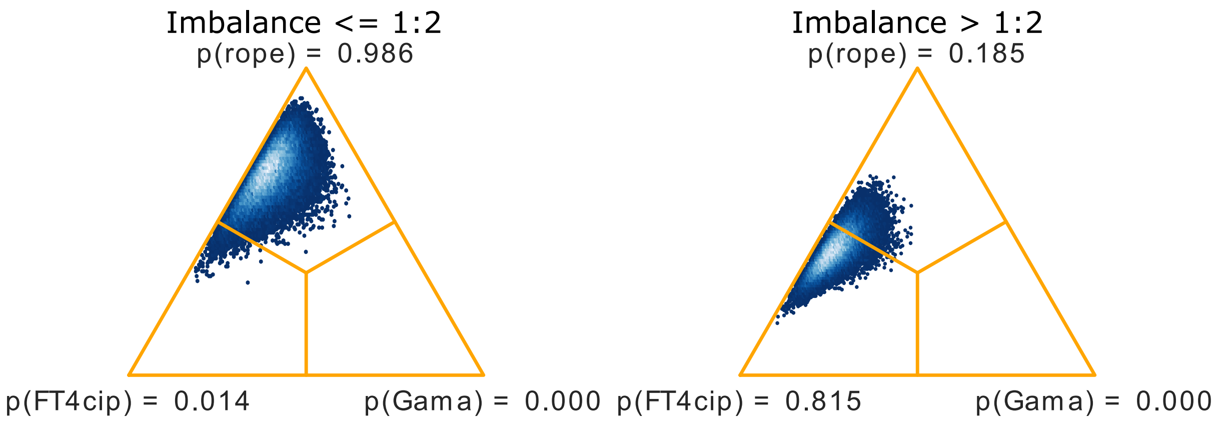

FIGURE 5.4: Comparison of FT4cip and Gama's FT using the Bayesian signed-rank test in subsets of databases by degree of class imbalance. The left plot shows results for databases with up to two objects of the majority class per object of the minority class and the right plot shows results for databases with more than two objects of the majority class per object of the minority class.

than Gama's FT in 81.5% of cases. We conclude that FT4cip outperforms Gama when there are more than two objects of the majority class per object of the minority class.

5.5 Chapter conclusions

The results of our first experiment confirms that LMT outperforms all MDTs tested in our survey. Therefore, we conclude that MTs have better classification performance than MDTs ($MDT \subset MT$).

With our second experiment, we confirm that the latest implementation of Gama's FT outperforms LMT, which is the top-performing Model Tree. Therefore, we conclude that FTs have better classification performance than MTs ($MT \subset FT$).

We have confirmed that there is an order in the classification performance of the DTs in the family of trees that uses linear combinations. MDTs show the lowest classification performance and FTs show the highest classification performance ($MDT \subset MT \subset FT$). Since Gama's FT is the only FT, we only need to compare our new classifier, FT4cip, against it.

Our third experiment showed that, FT4cip outperforms Gama's FT in 27.7% of cases, and the classifiers are equivalent in the remaining 72.3%. This shows a clear improvement of FT4cip over Gama's FT in some databases.

Since FT4cip was designed to deal with class imbalance problems, our hypothesis was that the classification performance of FT4cip compared to Gama's FT should be better in imbalanced databases. The results of our fourth experiment show that FT4cip outperforms Gama's FT in

81.5% of cases when there are more than two objects of the majority class per object of the minority class.

Chapter 6

Conclusions

In this work, we presented a new Functional Tree for class imbalance problems (FT4cip). Our objective was to improve the classification performance of Decision Trees (DTs) that use linear feature combinations, particularly in class imbalance problems. We selected this objective because DTs that use linear feature combinations manage to outperform those that do not, but many have shortcomings that reduce their performance in class imbalance problems.

In this work, we described the family of DTs that uses linear feature combinations, which has three members that allow feature combinations in different nodes. Multivariate Decision Trees (MDTs) allow feature combinations in inner nodes, Model Trees (MTs) allow feature combinations in leaf nodes[46], and Functional Trees (FTs) allow feature combinations in all nodes. However, there was no proper statistical comparison that shows that any of these trees outperforms the others.

Our first contribution is to show that functional trees surpass model trees, that in turn surpass multivariate trees. This way, we can compare FT4cip against the top-performing DT, which is Gama's FT, the only FT.

We identified eight key design decisions for building an FT: how to generate candidate splits, how to evaluate candidate splits, when to stop splitting a node, how to approach multi-class problems, what (if any) pruning method to use, what split types are allowed, what (if any) feature selection method to use, and what (if any) classifier to use at the leaves. From these design decisions Gama's FT uses a suboptimal split evaluation measure and a pruning method that optimizes accuracy, lowering the classification performance in class imbalance problems.

Our second contribution is the introduction of the Functional Tree for class imbalance problems (FT4cip), especially designed for class imbalance problems. To deal with Gama's FT limitations, each design decision for FT4cip aims to maximize AUC. We show with statistical tests that all design decisions taken contribute to classification performance, or they do not hinder the performance while keeping a simpler model or algorithm.

The main advantages of FT4cip over Gama's FT4cip are that we use the method for generating multivariate splits used by the top-performing MDT (the multi-class version of Fisher's linear discriminant), which also uses feature selection to keep short linear combinations [18]; we use Twoing as split evaluation measure, which has shown better classification results when maximizing AUC in the literature [38]; and we introduce a new pruning method that maximizes AUC, the AUC-optimizing Cost-Complexity pruning.

We show through a statistical comparison on 110 databases that FT4cip has better classification performance than Gama's FT. Furthermore, we show that FT4cip excels in class imbalance problems.

6.1 Future work

As future work, we suggest to include FT4cip in DT ensembles to improve their classification performance. We have preliminary results that show that a Random Forest of FT4cip outperforms a Random Forest of MDTs.

Contrast pattern-based classifiers based on DTs, such as PBC4cip [58], have shown great classification performance and produce interpretable models. Specifically, PBC4cip has been designed to deal with class imbalance problems. We suggest to use FT4cip as base classifier in PBC4cip to improve its classification performance.

We also suggest using an effective feature selection method when building logistic models to obtain shorter linear combinations. This would make the FT4cip trees easier to interpret.

To measure how good are our strategies for dealing with class imbalance, we suggest to compare Gama's FT after applying data level approaches for dealing with class imbalance. First, a balanced database is obtained by either oversampling the minority class or undersampling the majority class. Then, Gama's FT can be trained on the balanced database.

6.2 Research papers published

To identify the best DTs in terms of classification performance, we had to deal with the generalized lack of proper statistical comparison of newer DT works with previous literature. Our first step was making the largest survey on MDTs to date, which includes a statistical comparison of 19 MDTs in 57 databases. We published this contribution in:

Leonardo Cañete-Sifuentes, Raúl Monroy, and Miguel Angel Medina-Pérez. A review and experimental comparison of multivariate decision trees. *IEEE Access*, 9:110451–110479, 2021.

In our second paper, we introduced our new Functional Tree, the Functional Tree for class imbalance problems (FT4cip). We showed how a careful design that takes into account class imbalance results in a better classification performance than the top-performing DT (Gama's FT). The results were published in:

Leonardo Cañete-Sifuentes, Raúl Monroy, and Miguel Angel Medina-Pérez. Ft4cip: A new functional tree for classification in class imbalance problems. *Knowledge-Based Systems*, 252:109294, 2022.

Bibliography

[1] Nuno Gonçalo Costa Fernandes Marques de Abreu et al. *Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional.* PhD thesis, 2011.

[2] Aida Ali, Siti Mariyam Shamsuddin, and Anca L Ralescu. Classification with class imbalance problem. *Int. J. Advance Soft Compu. Appl*, 5(3), 2013.

[3] Mehmet Fatih Amasyali and Okan K. Ersoy. Cline: A new decision-tree family. *IEEE Transactions on Neural Networks*, 19(2):356–363, 2008.

[4] Alireza Arabameri, Sunil Saha, Wei Chen, Jagabandhu Roy, Biswajeet Pradhan, and Dieu Tien Bui. Flash flood susceptibility modelling using functional tree and hybrid ensemble techniques. *Journal of Hydrology*, 587:125007, 2020.

[5] Mustafa Aydin and Nazife Baykal. Feature extraction and classification phishing websites based on url. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 769–770. IEEE, 2015.

[6] Abdullateef O. Balogun, Kayode S. Adewole, Muiz O. Raheem, Oluwatobi N. Akande, Fatima E. Usman-Hamza, Modinat A. Mabayoje, Abimbola G. Akintola, Ayisat W. Asaju-Gbolagade, Muhammed K. Jimoh, Rasheed G. Jimoh, and Victor E. Adeyemo. Improving the phishing website detection using empirical analysis of function tree and its variants. *Heliyon*, 7(7):e07437, 2021.

[7] Alessio Benavoli, Giorgio Corani, Janez Demsar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, 18:77:1–77:36, 2017.

[8] Kristin P. Bennett, Nello Cristianini, John Shawe-Taylor, and Donghui Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3):295–313, 2000.

[9] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[10] Marco Better, Fred Glover, and Michele Samorani. Classification by vertical and cutting multi-hyperplane decision tree induction. *Decision Support Systems*, 48(3):430 – 436, 2010.

New concepts, methodologies and algorithms for business education and research in the 21st century.

[11] Rajen Bhatt and Abhinav Dhall. Skin segmentation dataset. *UCI Machine Learning Repository*, 2012.

[12] R Bhatt. Fuzzy-rough approaches for pattern classification: Hybrid measures, mathematical analysis, feature selection algorithms, decision tree algorithms, neural learning, and applications. In *Decision Tree Algorithms, Neural Learning, and Applications*. 2017.

[13] Leon Bobrowski and Marek Kretowski. Induction of multivariate decision trees by using dipolar criteria. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, pages 331–336, 2000.

[14] Jeffrey P Bradford, Clayton Kunz, Ron Kohavi, Cliff Brunk, and Carla E Brodley. Pruning decision trees with misclassification costs. pages 131–136, 1998.

[15] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[16] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[17] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[18] L. Cañete-Sifuentes, R. Monroy, M. A. Medina-Pérez, O. Loyola-González, and F. Vera Voronisky. Classification based on multivariate contrast patterns. *IEEE Access*, 7:55744–55762, 2019.

[19] Laurent Candillier and Vincent Lemaire. Design and analysis of the nomao challenge active learning in the real-world. In *Proceedings of the ALRA: Active Learning in Real-world Applications, Workshop ECML-PKDD*. Citeseer, 2012.

[20] Leonardo Cañete-Sifuentes, Raúl Monroy, and Miguel Angel Medina-Pérez. A review and experimental comparison of multivariate decision trees. *IEEE Access*, 9:110451–110479, 2021.

[21] Leonardo Cañete-Sifuentes, Raúl Monroy, and Miguel Angel Medina-Pérez. Ft4cip: A new functional tree for classification in class imbalance problems. *Knowledge-Based Systems*, 252:109294, 2022.

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794, 2016.

[23] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[24] Jacek Czerniak and Hubert Zarzycki. Application of rough sets in the presumptive diagnosis of urinary system diseases. In *Artificial intelligence and security in computing systems*, pages 41–51. Springer, 2003.

[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[26] M. Elter, R. Schulz-Wendtland, and T. Wittenberg. The prediction of breast cancer biopsy outcomes using two cad approaches that both emphasize an intelligible decision process. *Medical Physics*, 34(11):4164–4172, 2007.

[27] Elaine Fehrman, Awaz K. Muhammad, Evgeny M. Mirkes, Vincent Egan, and Alexander N. Gorban. The five factor model of personality and evaluation of drug consumption risk. In Francesco Palumbo, Angela Montanari, and Maurizio Vichi, editors, *Data Science*, pages 231–242, Cham, 2017. Springer International Publishing.

[28] Kelwin Fernandes, Jaime S. Cardoso, and Jessica Fernandes. Transfer learning with partial observability applied to cervical cancer screening. In Luís A. Alexandre, José Salvador Sánchez, and João M. F. Rodrigues, editors, *Pattern Recognition and Image Analysis - 8th Iberian Conference, IbPRIA 2017, Faro, Portugal, June 20-23, 2017, Proceedings*, volume 10255 of *Lecture Notes in Computer Science*, pages 243–250. Springer, 2017.

[29] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugenics*, 7:179–188, 1936.

[30] Eibe Frank, Mark A. Hall, and Ian H. Witten. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". Morgan Kaufmann, 4th edition, 2016.

[31] Jerome H Friedman. A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, (4):404–408, 1977.

[32] Keinosuke Fukunaga. Chapter 10 - feature extraction and linear mapping for classification. In Keinosuke Fukunaga, editor, *Introduction to Statistical Pattern Recognition (Second Edition)*, pages 441–507. Academic Press, Boston, second edition edition, 1990.

[33] João Gama. Probabilistic linear tree. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997*, pages 134–142, 1997.

[34] João Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.

[35] Heng Guo and Saul B. Gelfand. Classification trees with neural network feature extraction. *IEEE Transactions on Neural Networks*, 3(6):923–933, 1992.

[36] Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 545–552, 2004.

[37] David G. Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 1002–1007, 1993.

[38] Víctor Adrián Sosa Hernández, Raúl Monroy, Miguel Angel Medina-Pérez, Octavio Loyola-González, and Francisco Herrera. A practical tutorial for decision tree induction: Evaluation measures for candidate splits and opportunities. *ACM Computing Surveys*, 54(1), January 2021.

[39] Brian A Johnson and Kotaro Iizuka. Integrating openstreetmap crowdsourced data and landsat time-series imagery for rapid land use/land cover (lulc) mapping: Case study of the laguna de bay area of the philippines. *Applied Geography*, 67:140–149, 2016.

[40] Brian Johnson and Zhixiao Xie. Classifying a high resolution image of an urban area using super-object information. *ISPRS Journal of Photogrammetry and Remote Sensing*, 83:40–49, 2013.

[41] Brian Johnson, Ryutaro Tateishi, and Zhixiao Xie. Using geographically weighted variables for image classification. *Remote Sensing Letters*, 3(6):491–499, 2012.

[42] Brian Alan Johnson, Ryutaro Tateishi, and Nguyen Thanh Hoan. A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International Journal of Remote Sensing*, 34(20):6969–6982, 2013.

[43] Hamdi Tolga Kahraman, Seref Sagiroglu, and Ilhami Colak. The development of intuitive knowledge classifier and the modeling of domain dependent data. *Knowledge-Based Systems*, 37:283–295, 2013.

[44] Hyunjoong Kim and Wei-Yin Loh. Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, 96(454):589–604, 2001.

[45] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Second International Conference on Knoledge Discovery and Data Mining*, pages 202–207, 1996.

[46] Niels Landwehr, Mark A. Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[47] Xiao-Bai Li, James R. Sweigart, James T. C. Teng, Joan M. Donohue, Lori A. Thombs, and S. M. Wang. Multivariate decision trees using linear discriminants and tabu search. *IEEE Transactions on Systems, Man, and Cybernetics, Part A (Systems and Humans)*, 33(2):194–205, 2003.

[48] Xiao-Bai Li. A scalable decision tree system and its application in pattern recognition and intrusion detection. *Decision Support Systems*, 41(1):112 – 130, 2005.

[49] Max Little, Patrick McSharry, Stephen Roberts, Declan Costello, and Irene Moroz. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. *Nature Precedings*, pages 1–1, 2007.

[50] Huan Liu and Rudy Setiono. Feature transformation and multivariate decision tree induction. In *Discovery Science, First International Conference, DS '98, Fukuoka, Japan, December 14-16, 1998, Proceedings*, pages 279–290, 1998.

[51] Weiwei Liu and Ivor W. Tsang. Making decision trees feasible in ultrahigh feature and label dimensions. *Journal of Machine Learning Research*, 18:81:1–81:36, 2017.

[52] Wei-Yin Loh and Yu-Shan Shih. Split selection methods for classification trees. *Statistica sinica*, pages 815–840, 1997.

[53] Wei-Yin Loh. Improving the precision of classification trees. *The Annals of Applied Statistics*, pages 1710–1737, 2009.

[54] Wei-Yin Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

[55] Victoria López, Alberto Fernández, and Francisco Herrera. On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Information Sciences*, 257:1–13, 2014.

[56] Asdrúbal López Chau, Jair Cervantes, Lourdes López-García, and Farid García-Lamont. Fisher's decision tree. *Expert Systems with Applications*, 40(16):6283–6291, 2013.

[57] Octavio Loyola-González. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE Access*, 7:154096–154113, 2019.

[58] Octavio Loyola-González, Miguel Angel Medina-Pérez, José Fco. Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, Raúl Monroy, and Milton García-Borroto. Pbc4cip: A new contrast pattern-based classifier for class imbalance problems. *Knowledge-Based Systems*, 115:100–109, 2017.

[59] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171, 2013.

[60] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, 04 2016.

[61] Arturo Magana-Mora and Vladimir B Bajic. Omniga: Optimized omnivariate decision trees for generalizable classification models. *Scientific reports*, 7(1):3898, 2017.

[62] Olvi L. Mangasarian, W. Nick Street, and William H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Oper. Res.*, 43(4):570–577, 1995.

[63] Kamel Mansouri, Tine Ringsted, Davide Ballabio, Roberto Todeschini, and Viviana Consonni. Quantitative structure-activity relationship models for ready biodegradability of chemicals. *Journal of Chemical Information and Modeling*, 53(4):867–878, 2013.

[64] Naresh Manwani and P. S. Sastry. Geometric decision tree. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1):181–192, 2012.

[65] Bjoern H. Menze, B. Michael Kelm, Daniel Nicolas Splitthoff, Ullrich Köthe, and Fred A. Hamprecht. On oblique random forests. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II*, pages 453–469, 2011.

[66] Jose G. Moreno-Torres, José A. Sáez, and Francisco Herrera. Study on the impact of partition-induced dataset shift on k -fold cross-validation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1304–1312, 2012.

[67] Amirhosein Mosavi, Ataollah Shirzadi, Bahram Choubin, Fereshteh Taromideh, Farzaneh Sajedi Hosseini, Moslem Borji, Himan Shahabi, Aryan Salvati, and Adrienn A Dineva. Towards an ensemble machine learning model of random subspace based functional tree classifier for snow avalanche susceptibility mapping. *IEEE Access*, 8:145968–145983, 2020.

[68] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

[69] Feiping Nie, Wei Zhu, and Xuelong Li. Decision tree SVM: an extension of linear SVM for non-linear classification. *Neurocomputing*, 401:153–159, 2020.

[70] Tao Peng, Yunzhi Chen, and Wei Chen. Landslide susceptibility modeling using remote sensing data and random subspace-based functional tree classifier. *Remote Sensing*, 14(19), 2022.

[71] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[72] Thomas Rusch and Achim Zeileis. Discussion on fifty years of classification and regression trees. *International Statistical Review*, 82(3):361–367, 2014.

[73] Betul Erdogdu Sakar, M. Erdem Isenkul, Cemal Okan Sakar, Ahmet Sertbas, Fikret Gürgen, Sakir Delil, Hulya Apaydin, and Olcay Kursun. Collection and analysis of a parkinson speech dataset with multiple types of sound recordings. *IEEE Journal of Biomedical and Health Informatics*, 17(4):828–834, 2013.

[74] Guzmán Santafé, Iñaki Inza, and José Antonio Lozano. Dealing with the evaluation of supervised classification algorithms. *Artificial Intelligence Review*, 44(4):467–508, 2015.

[75] Miriam Seoane Santos, Pedro Henriques Abreu, Pedro J. García-Laencina, Adélia Simão, and Armando Carvalho. A new cluster-based oversampling method for improving survival prediction of hepatocellular carcinoma patients. *J. Biomed. Informatics*, 58:49–59, 2015.

[76] Michael Scholz and Tristan Wimmer. A comparison of classification methods across different data complexity scenarios and datasets. *Expert Systems with Applications*, 168:114217, 2021.

[77] Semeion, Research Center of Sciences of Communication. Steel plates faults data set, 2010. dataset provided by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy. www.semeion.it.

[78] S. Shah and P. Shanti Sastry. New algorithms for learning and pruning oblique decision trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 29(4):494–505, 1999.

[79] Pedro F. B. Silva, André R. S. Marçal, and Rubim M. Almeida da Silva. Evaluation of features for leaf discrimination. In Mohamed Kamel and Aurélio J. C. Campilho, editors, *Image Analysis and Recognition - 10th International Conference, ICIAR 2013, Póvoa do Varzim, Portugal, June 26-28, 2013. Proceedings*, volume 7950 of *Lecture Notes in Computer Science*, pages 197–204. Springer, 2013.

[80] Rastislav J. R. Struharik, Vuk Vranjkovic, Stanisa Dautovic, and Ladislav A. Novak. Inducing oblique decision trees. In *IEEE 12th International Symposium on Intelligent Systems and Informatics, SISY 2014, Subotica, Serbia, September 11-13, 2014*, pages 257–262, 2014.

[81] Raquel Teixeira, Carina Rodrigues, Carla Moreira, Henrique Barros, and Rui Camacho. Machine learning methods to predict attrition in a population-based cohort of very preterm infants. *Scientific reports*, 12(1):1–10, 2022.

[82] Athanasios Tsanas, Max A. Little, Cynthia Fox, and Lorraine O. Ramig. Objective automatic assessment of rehabilitative speech treatment in parkinson's disease. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(1):181–190, 2014.

[83] Turing Institute. Statlog (vehicle silhouettes) data set. This dataset comes from the Turing Institute, Glasgow, Scotland.

[84] Priscilla Koch Wagner, Sarajane Marques Peres, Renata Cristina Barros Madeo, Clodoaldo Aparecido de Moraes Lima, and Fernando de Almeida Freitas. Gesture unit segmentation using spatial-temporal information and machine learning. In William Eberle and Chutima Boonthum-Denecke, editors, *Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014, Pensacola Beach, Florida, USA, May 21-23, 2014*. AAAI Press, 2014.

[85] Fei Wang, Quan Wang, Feiping Nie, Weizhong Yu, and Rong Wang. Efficient tree classifiers for large scale datasets. *Neurocomputing*, 284:70–79, 2018.

[86] Fei Wang, Quan Wang, Feiping Nie, Zhongheng Li, Weizhong Yu, and Fuji Ren. A linear multivariate binary decision tree classifier based on k-means splitting. *Pattern Recognition*, 107:107521, 2020.

[87] J. Weinstein, E. Collisson, and et al. The cancer genome atlas pan-cancer analysis project. *Nature Genetics*, 45(10):1113–1120, Oct 2013.

[88] D. C. Wickramarachchi, B. L. Robertson, Marco Reale, C. J. Price, and J. Brown. HH-CART: an oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

[89] DC Wickramarachchi, Blair L Robertson, Marco Reale, CJ Price, and JA Brown. A reflected feature space for cart. *Australian & New Zealand Journal of Statistics*, 61(3):380–391, 2019.

[90] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. Knowledge discovery on RFM model using bernoulli sequence. *Expert Syst. Appl.*, 36(3):5866–5871, 2009.

[91] Olcay Taner Yildiz and Ethem Alpaydin. Omnivariate decision trees. *IEEE Transactions on Neural Networks*, 12(6):1539–1546, 2001.

[92] Olcay Taner Yildiz and Ethem Alpaydin. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):323–353, 2005.

[93] Chongsheng Zhang, Changchang Liu, Xiangliang Zhang, and George Almpanidis. An up-to-date comparison of state-of-the-art classification algorithms. *Expert Systems with Applications*, 82:128–150, 2017.

[94] Fang Zhou, Claire Q, and Ross D. King. Predicting the geographical origin of music. In Ravi Kumar, Hannu Toivonen, Jian Pei, Joshua Zhexue Huang, and Xindong Wu, editors, *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 1115–1120. IEEE Computer Society, 2014.

[95] Maciej Zikeba, Sebastian K Tomczak, and Jakub M Tomczak. Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, 2016.

Appendix A

List of databases

Here we show the number of objects, features, classes, and degree of class imbalance of the 110 databases used in our experimental comparison. Table A.1 shows the first 40 databases used in the experimental comparison of all MDTs in our recent survey [20]. In table A.2, we show 17 additional databases used in the experimental comparison of some MDTs of the same survey. The need to divide the databases is explained in Section 3.1.2. In Table A.3, we show 53 databases that were not used in the MDT survey. All databases, except dataset3d, come from the UCI repository [25], and in each table, we show the names exactly as found on the repository. We also include additional citations for each database when requested by the owners.

Table A.1: Details about 40 of the 110 databases used in the experimental comparison. These are the databases that worked with all MDTs. We show each database's number of objects, features (without the class), and classes. The column Imbalance shows the number of objects of the majority class for each object of the minority class. References for the databases with a citation request are included.

Database	#Objects	#Features	#Classes	Imbalance
Balance Scale	625	4	3	5.88
QSAR biodegradation [63]	1055	41	2	1.96
Breast Tissue	106	9	6	1.57
Climate Model Simulation Crashes [59]	540	20	2	10.74
Cloud	108	4	4	1.33
CNAE-9	1080	856	9	1.00
Vertebral Column - two class	310	6	2	2.10
Vertebral Column - three class	310	6	3	2.50
dataset3d [18]	120	3	2	1.00
Pima Indians Diabetes	768	8	2	1.87

Continued on next page.

Database	#Objects	#Features	#Classes	Imbalance
Ecoli	336	7	8	71.50
Glass Identification	214	9	6	8.44
Haberman's Survival	306	3	2	2.78
Ionosphere	351	34	2	1.79
Iris	150	4	3	1.00
User Knowledge Modeling [43]	403	5	4	2.58
Letter Recognition	20000	16	26	1.11
Liver Disorders	345	6	2	1.38
LSVT Voice Rehabilitation [82]	126	310	2	2.00
Madelon [36]	2600	500	2	1.00
Libras Movement	360	90	15	1.00
Optical Recognition of Handwritten Digits	5620	64	10	1.03
Parkinson Speech Dataset with Multiple Types of Sound Recordings [73]	1208	26	2	1.32
Parkinsons [49]	195	22	2	3.06
Pen-Based Recognition of Handwritten Digits	10992	16	10	1.08
seeds	210	7	3	1.00
Image Segmentation	2310	19	7	1.00
Connectionist Bench (Sonar, Mines vs. Rocks)	208	60	2	1.14
Spambase	4601	57	2	1.54
SPECTF Heart	267	44	2	3.85
Low Resolution Spectrometer	531	100	4	3.63
Statlog (Image Segmentation)	2310	19	7	1.00
Statlog (Vehicle Silhouettes) [83]	846	18	4	1.10
Connectionist Bench (Vowel Recognition - Deterding Data)	990	10	11	1.00
Breast Cancer Wisconsin (Diagnostic)	569	30	2	1.68
Wholesale customers [1]	440	7	2	2.10
Wilt [42]	4839	5	2	17.54
Wine	178	13	3	1.48
Wine Quality - red [23]	1599	11	6	68.10
Yeast	1484	8	10	92.60

Table A.2: Details about 17 of the 110 databases used in the experimental comparison. These are the databases used in the experimental comparison that some MDTs could not process. We show each database's number of objects, number of features (without the class), number of classes. The column Imbalance shows the number of objects of the majority class for each object of the minority class. References for the databases with a citation request are included.

Database	#Objects	#Features	#Classes	Imbalance
Quality Assessment of Digital Colposcopies - Green [28]	98	62	2	2.16
Quality Assessment of Digital Colposcopies - Hinselmann [28]	97	62	2	5.47
Quality Assessment of Digital Colposcopies - Schiller [28]	92	62	2	2.68
Cardiotocography	2126	41	3	9.40
Geographical Original of Music [94]	1059	68	33	6.27
Geographical Original of Music - Chromatic [94]	1059	116	33	6.27
Steel Plates Faults [77]	1941	27	7	9.73
Forest type mapping [41]	523	27	4	2.35
Gesture Phase Segmentation - Processed [84]	9873	32	5	2.96
Hill-Valley - without noise	1212	100	2	1.02
HTRU2 [60]	17898	8	2	9.92
MAGIC Gamma Telescope	19020	10	2	1.84
Ozone Level Detection	2534	72	2	14.84
Urban Land Cover [40]	675	147	9	4.21
Waveform Database Generator (Version 1)	5000	21	3	1.03
Waveform Database Generator (Version 2)	5000	40	3	1.02
Wireless Indoor Localization [12]	2000	7	4	1.00

Table A.3: Details about the remaining 53 of the 110 databases used in the experimental comparison. We include this 53 databases that were not used in the experimental comparison of MDTs.We show each database's number of objects, number of features (without the class), number of classes. The column Imbalance shows the number of objects of the majority class for each object of the minority class. References for the databases with a citation request are included.

Database	#Objects	#Features	#Classes	Imbalance
Internet Advertisements	3279	1558	2	6.14

Continued on next page.

Database	#Objects	#Features	#Classes	Imbalance
Thyroid Disease	3772	23	3	257.79
Arcene [36]	200	10000	2	1.27
Arrhythmia	452	279	16	122.50
Statlog (Australian Credit Approval)	690	14	2	1.25
banknote authentication	1372	4	2	1.25
Breast Cancer Wisconsin (Original) [62]	699	9	2	1.90
Car Evaluation	1728	6	4	18.62
Cervical cancer (Risk Factors) - Biopsy [28]	858	32	2	14.60
Cervical cancer (Risk Factors) - Cytology [28]	858	32	2	18.50
Cervical cancer (Risk Factors) - Hinselmann [28]	858	32	2	23.51
Cervical cancer (Risk Factors) - Schiller [28]	858	32	2	10.59
Crowdsourced Mapping [39]	300	28	6	2.17
Credit Approval	690	15	2	1.25
Acute Inflammations - d1 [24]	120	6	2	1.03
Acute Inflammations - d2 [24]	120	6	2	1.40
Drug consumption (quantified) - alcohol [27]	1885	12	7	22.32
Drug consumption (quantified) - amphet [27]	1885	12	7	16.00
Drug consumption (quantified) - benzos [27]	1885	12	7	11.90
Drug consumption (quantified) - caff [27]	1885	12	7	138.50
Drug consumption (quantified) - cannabis [27]	1885	12	7	3.31
Drug consumption (quantified) - choc [27]	1885	12	7	269.00
Drug consumption (quantified) - coke [27]	1885	12	7	54.63
Drug consumption (quantified) - crack [27]	1885	12	7	813.50

Continued on next page.

Database	#Objects	#Features	#Classes	Imbalance
Drug consumption (quantified) - ecstasy [27]	1885	12	7	48.62
Drug consumption (quantified) - heroin [27]	1885	12	7	123.46
Drug consumption (quantified) - ketamine [27]	1885	12	7	372.50
Drug consumption (quantified) - legalh [27]	1885	12	7	37.72
Drug consumption (quantified) - lsd [27]	1885	12	7	82.23
Drug consumption (quantified) - meth [27]	1885	12	7	36.64
Drug consumption (quantified) - mushrooms [27]	1885	12	7	245.50
Drug consumption (quantified) - nicotine [27]	1885	12	7	5.65
Drug consumption (quantified) - semer [27]	1885	12	5	1877.00
Drug consumption (quantified) - vsa [27]	1885	12	7	207.86
gene expression cancer RNA-Seq [87]	801	20531	5	3.85
Gisette [36]	7000	5000	2	1.00
HCC Survival [75]	165	49	2	1.62
Hill-Valley - with noise	1212	100	2	1.00
ILPD (Indian Liver Patient Dataset)	583	10	2	2.49
Leaf [79]	340	14	36	2.00
Mammographic Mass [26]	961	5	2	1.16
Nomao [19]	34465	118	2	2.50
Page Blocks Classification	5473	10	5	175.46
Polish companies bankruptcy data - 1year [95]	7027	64	2	24.93
Polish companies bankruptcy data - 2year [95]	10173	64	2	24.43
Polish companies bankruptcy data - 3year [95]	10503	64	2	20.22

Continued on next page.

Database	#Objects	#Features	#Classes	Imbalance
Polish companies bankruptcy data - 4year [95]	9792	64	2	18.01
Polish companies bankruptcy data - 5year [95]	5910	64	2	13.41
Statlog (Landsat Satellite)	6435	36	7	2.45
SECOM	1567	590	2	14.07
Skin Segmentation [11]	245057	3	2	3.82
Blood Transfusion Service Center [90]	748	4	2	3.20
Wine Quality - white [23]	4898	11	7	439.60