PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Optimization Methods for Interpretable Machine Learning

Tommaso Aldinucci

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Optimization Methods for Interpretable Machine Learning

Tommaso Aldinucci

Advisor:

Prof. Fabio Schoen

Head of the PhD Program:

Prof. Stefano Berretti

Evaluation Committee:
Prof. Marco Locatelli, *Università degli Studi di Parma*
Prof. Rafael Blanquero Bravo, *Universidad de Sevilla*

XXXVI ciclo — January 2024

Alla mia famiglia

Acknowledgments

Here we go again. Another time, the last, in the whole life that I have to find the words to write this short Section. The period of my PhD has been the strangest of my whole life. For the major part of these three years my mind was stacked into solving life problems rather than optimization ones. As all men of science, we are instinctively inclined to approach each problem using reasoning. Unfortunately, or fortunately, life is not an optimization problem. Many times you have no power to change all the variables in play, you can only try to do the best by acting on those you can control. Well, if this PhD period had been more "relaxed", we all know from the theory of relaxation that the results obtained would probably have been better (surely not worse).

Now, some people I feel I should deeply thank. First of all, my advisor, Prof. Fabio Schoen. I have had the pleasure of being under his supervision since my bachelor's degree and these few lines don't do justice to the extraordinary person he is. The second person is undoubtedly Matteo. I still remember the first day I spoke to you for the first time during a linear algebra class, almost twelve years ago now. I knew immediately that you were a kind of "alien" compared to the rest of us. I deeply thank you for the immense help in this very difficult time. Then, all the guys of the optimization squad of the laboratory during these years: Pierluigi, Enrico, Simone, Leonardo, Tomaso, Alessio and Marco. Thank you for all the laughs and for the many scientific discussions addressed. Finally, Davide, Arturo and Saliha the latest additions to the group. I hope to work with you in the future as well.

Florence, January 31, 2024.

Tommaso

Abstract

This dissertation is concerned with optimization problems related to transparent machine learning. In the contemporary landscape of machine learning, the importance of interpretability has become essential, resonating across diverse industries and applications. As complex machine learning models, often characterized as "black boxes," continue to demonstrate remarkable predictive capabilities, there arises a pressing need to explain their decision-making processes. Interpretability is crucial not only for building trust in automated systems but also for meeting ethical, legal, and regulatory requirements. In healthcare, interpretable models can offer insights into diagnostic reasoning, aiding healthcare professionals in decision-making and enhancing patient trust. In finance, transparent models are essential for ensuring fair lending practices and regulatory compliance. Legal systems increasingly demand explainability to justify algorithmic decisions, emphasizing the significance of interpretable machine learning in justice.

Within this setting, we first briefly describe the most used interpretable models then we discuss novel optimization strategies to handle their learning problem. More precisely, we describe state-of-art approaches to learn decision tree models and, to this aim, we propose two new techniques based to genetic algorithms and mixed integer programming.

Afterwards, we present a work on risk score models, enhancing their expressiveness by extending these estimators in a more generalized framework. Finally, in the context of local explanations, we propose a recommendation system for random forest that dinamically maps each point to a single shallow tree in the ensemble.

Contents

Chapter 1

Introduction

In the contemporary era of machine learning, where algorithms wield immense power in shaping decisions across diverse domains, the call for transparency and interpretability has never been more resonant. As artificial intelligence gradually becomes more and more present in our lives, from healthcare diagnostics to financial predictions and legal deliberations, the black-box nature of many advanced models poses inherent challenges. The rise of interpretable models stands as a response to the imperative need for understanding and trust in the decision-making processes of these sophisticated algorithms.

Given the huge increase of complex models, such as deep neural networks, that exhibit state-of-art predictive performance in many complex tasks, interpretability is often sacrified to obtain more expressive power. While these models excel in discerning intricate patterns within very large datasets, the miss of explanations on how they arrive at specific decisions raises ethical, legal, and practical concerns. The ability to comprehend, question, and validate the reasoning behind algorithmic outcomes becomes essential, especially in applications where human lives, financial stability, or legal outcomes hang in the balance. Even though there is no a formal definition for interpretability, it is worth to mention the one by (Miller, 2019):

"Interpretability is the degree to which a human can understand the cause of a decision."

The importance of interpretable models is underscored by the necessity to bridge the gap between accuracy and transparency. It's not merely a quest for simplicity but a strategic imperative to empower end-users, be they healthcare professionals, financial analysts, or legal experts, with the ability to gain insights into the decision-making processes of these algorithms.

A common belief in the machine learning community suggests that enhanced model complexity directly correlates with superior accuracy, implying the necessity of intricate black-box structures for good predictive performance (see Figure 1.1). However, this assumption often proves inaccurate, especially when dealing with

structured data with a robust representation through inherently meaningful features (Rudin, 2019). In these scenarios, there frequently emerges small differences in performance between more intricate classifiers (such as deep neural networks, boosted decision trees, and random forests) and considerably simpler counterparts (like logistic regression).

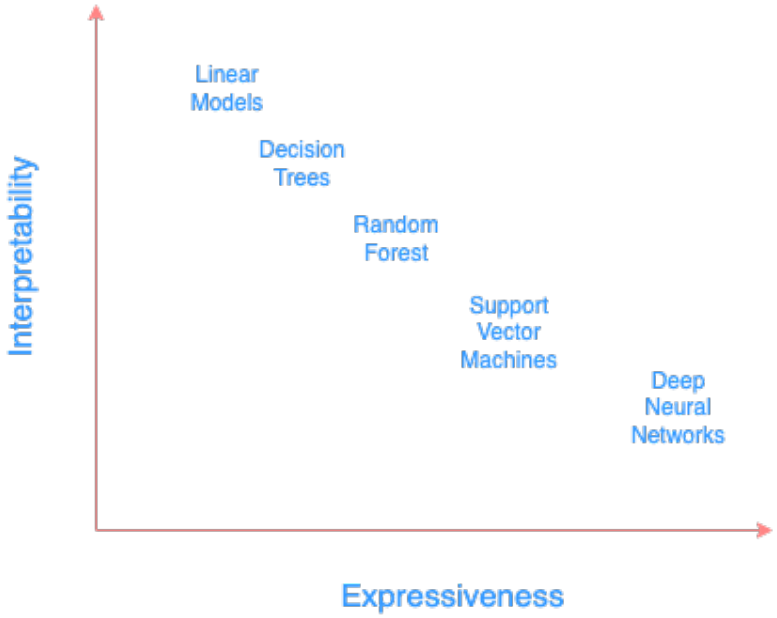


Figure 1.1: A fictional graph of the relationship between the performance of models and their interpretability.

As a compromise, a stream of research is focused on creating methods to explain black-boxes, by fitting a surrogate transparent model to approximate the behaviour of the black-box one. In this context, it is worth to mention the work (Ribeiro et al., 2016) where a linear model is employed to provide explanations of single points (*local explanations*) by building a first order approximator.

Within this background, in this thesis we propose and analize different optimization approaches to handle the learning problem of the most common transparent models. In a supervised learning context, let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the output space. For the rest of this manuscript, unless otherwise stated, we will focus on datasets with N points and where $\mathcal{X} = \mathbb{R}^p$, using a bold notation to indicate feature vectors $\boldsymbol{x} \in \mathbb{R}^p$. The output space $\mathcal{Y}$ will be chosen as $\mathbb{R}$ in the case of regression and $\{-1, 1\}$ for binary classification. The rest of the thesis is organized as follows:

- In Chapter 2, we briefly review the most know trasparent models, focusing on their interpretability.

- In Chapter 3, we review the literature in the context of learning decision tree. Moreover, we propose two novel methods for handling the problem. The first one is a new memetic strategy based on genetic algorithms which is able to induce classification trees with competitive performance with respect to exact formulations on real-world datasets. In the second part we introduce the concept of *Loss Optimal Classification Trees* by encapsulating different losses in a general Mixed Integer Programming framework. Then we propose to use the logistic loss to induce *Optimal Logistic Classification Trees*.

- In Chapter 4, we address the problem of learning *Risk Scores* that are very common trasparent and sparse linear estimators widely used in the healthcare context. Moreover, we extend these models by proposing a generalized framework to induce *Optimal Generalized Risk Scores*.

- In Chapter 5, we propose a recommendation system to dinamically select a single shallow decision tree in a Random Forest to obtain post-hoc local explanations for each point.

- In Chapter 6, we finally give some concluding remarks and suggest possible themes for future research.

Chapter 2

Interpretable Models

A straightforward approach to enhance interpretability is to use algorithms that generate models that are ineherently transparent every time that it is possible. Frequently employed interpretable models include Linear Regression, Logistic Regression, and Decision Trees. In this Chapter, we provide a brief overview of these models without delving into extensive details.

2.1 Linear Regression

In the regression context, let $y_i \in \mathbb{R}\ \forall i = 1, ..., N$. A linear regression model is a particular instance of a Generalized Linear Model (GLM) (Nelder and Wedderburn, 1972) where the response variable y is assumed to be linearly dependent with respect to the feature vector $\boldsymbol{x} \in \mathbb{R}^p$, i.e.:

$$\mathbb{E}[y|\boldsymbol{x}] = g^{-1}(\boldsymbol{w}^T\boldsymbol{x} + \beta + \epsilon) \tag{2.1}$$

where ϵ is a normally distributed random variable to represent a noise term and $g : \mathbb{R} \to \mathbb{R}$ is known as *link function* of the GLM model and, for linear regression, it is equal to the identity function thus, $g(t) = t\ \forall t \in \mathbb{R}$, obtaining:

$$y = \boldsymbol{w}^T\boldsymbol{x} + \beta + \epsilon \tag{2.2}$$

The method commonly employed for determining the weights is Ordinary Least Squares (OLS), which aims to minimize the squared differences between the observed and predicted outcomes by solving:

$$\min_{\boldsymbol{w},\beta} \mathcal{L}(\boldsymbol{w}, \beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i - \beta)^2 \tag{2.3}$$

which is an unconstrained, smooth and convex optimization problem that can be efficiently solved with second order algorithms like the Newton Method and BFGS.

The primary strength of linear regression models lies in their linearity which simplifies the estimation process. The additivity of the components facilitates the straightforward isolation of their effects. However, in cases where there's a suspicion of feature interactions or a non-linear correlation between a feature and the target value, this kind of models tends to be insufficiently expressive. Notably, the straightforward interpretation of these models is related to the fact that each weight of the classifier can be understood. This characteristic contributes to the widespread use of linear models and similar approaches across academic disciplines such as medicine, sociology, psychology, and various quantitative research fields. In the medical domain, for instance, the ability not only to predict a patient's clinical outcome but also to quantify the impact of a drug while incorporating factors like sex, age, and other features in an easily interpretable manner is crucial.

2.2 Logistic Regression

In the same way, in the binary classification context, we assume $y \mid \boldsymbol{x}$ to be a target random variable with Bernoulli distribution $B(p)$. A logistic regression model $m(\boldsymbol{x}; \boldsymbol{w})$ is an instance of the GLM class that estimates the expected value $\mathbb{E}[y|\boldsymbol{x}] = p = \mathbb{P}(y = 1 \mid \boldsymbol{x})$:

$$\mathbb{E}[y|\boldsymbol{x}] = g^{-1}(\boldsymbol{w}^T\boldsymbol{x} + \beta) \tag{2.4}$$

through the logistic link function $g^{-1} = \sigma(l(\boldsymbol{x}; \boldsymbol{w}))$ i.e., m is a linear estimator of the log-odds or *logit* (Hastie et al., 2009):

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = m(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + \beta.$$

The choice of the weights of this model can be made by minimizing the logistic loss function, which can be obtained by the maximum likelihood estimation (see, e.g, (Hastie et al., 2009)):

$$l(\boldsymbol{w}^T\boldsymbol{x} + \beta; y) = log(1 + \exp(-y(\boldsymbol{w}^T\boldsymbol{x} + \beta))) \tag{2.5}$$

Thus, the empirical risk minimization problem becomes the following:

$$\min_{\boldsymbol{w},\beta} \mathcal{L}(\boldsymbol{w}, \beta) = \frac{1}{N}\sum_{i=1}^{N} log(1 + \exp(-y_i(\boldsymbol{w}_i^T\boldsymbol{x}_i + \beta))) \tag{2.6}$$

Also in this case, the learning problem 2.6 is unconstrained and the objective function is smooth and convex. For this reason, even this time, second order algorithms can be efficiently employed.

The interpretation of the weights in logistic regression differs from the one in linear regression. In logistic regression, the outcome represents a probability thus, the weights no longer implies directly a linear influence on the outcome since the weighted sum undergoes a transformation through the logistic function to yield a probability. Indeed we have:

$$logit(p) = log(\frac{p}{1-p}) = \boldsymbol{w}^T\boldsymbol{x} + \beta \tag{2.7}$$

$$odds = \frac{p}{1-p} = \exp(\boldsymbol{w}^T\boldsymbol{x} + \beta) \tag{2.8}$$

We can infer how each feature x_j is related to the odds of the positive outcome by the odds ratio:

$$\frac{odds_{x_j+1}}{odds_{x_j}} = \frac{\exp(\sum_{k\neq j} w_k x_k + w_j(x_j+1) + \beta)}{\exp(\sum_k w_k x_k + \beta)} = \tag{2.9}$$

$$= \exp(w_j(x_j+1) - w_j x_j) = \exp(w_j) \tag{2.10}$$

That is, a variation in a feature x_j by one unit leads to a multiplicative change in the odds ratio by a factor of $\exp(w_j)$. Another way to articulate this is that a one-unit change in x_j increases the log odds ratio by the value of the corresponding weight. Many individuals prefer interpreting the odds ratio directly, since dealing with the *log* of a value is considered more difficult. Understanding the odds ratio itself can be a bit challenging. For instance, if the odds are 2, it implies that the probability of $y = 1$ is twice as high as $y = 0$.

2.3 Generalized Additive Models

Generalized Additive Models (GAMs) represent a flexible and powerful extension of traditional GLMs, allowing for the modeling of complex, non-linear relationships between variables. Introduced in (Hastie and Tibshirani, 1990) , GAMs are particularly useful when dealing with data that exhibits non-linearity, interactions, and other intricate patterns that cannot be adequately captured by linear models. The fundamental idea behind GAMs is to extend the linear model by incorporating single functions of the predictor variables. Unlike linear models, which assume a linear relationship between the predictors and the response, GAMs allow for non-linear relationships by adding a general function $f_j(x_j)$ for each covariate x_j.

The general form of a GAM can be expressed as follows:

$$\mathbb{E}[y|\boldsymbol{x}] = g^{-1}(\sum_j f_j(x_j) + \beta) \tag{2.11}$$

where, as previously discussed for GLM, the link function g depends on the particular task.

The key innovation in GAMs is the inclusion of a single function for each feature. These functions are flexible and can capture complex relationships without assuming a specific functional form. Popular choices for smooth functions include splines, thin-plate splines, and local regression. Due to the potential complexity of the shape functions, GAMs exhibit higher accuracy compared to straightforward linear models but, even this time, they lack interactions between features, resulting in easily interpretable models.

The main approach to fit these models is the *Backfitting* method which is a greedy iterative optimization algorithm. For each predictor variable, its function is updated while keeping the others fixed. This involves fitting a univariate smoother (e.g., a spline) to the residuals from the current model. Then, the overall model is updated by adding the component to the predictor and this process is repeated while a convergence criteria is not satisfied, for further details see (Hastie and Tibshirani, 1990).

Finally, an end user can easily undertand how a component impacts on the target by looking at the plot of the relative function. An example of this is shown in Figure 2.1.

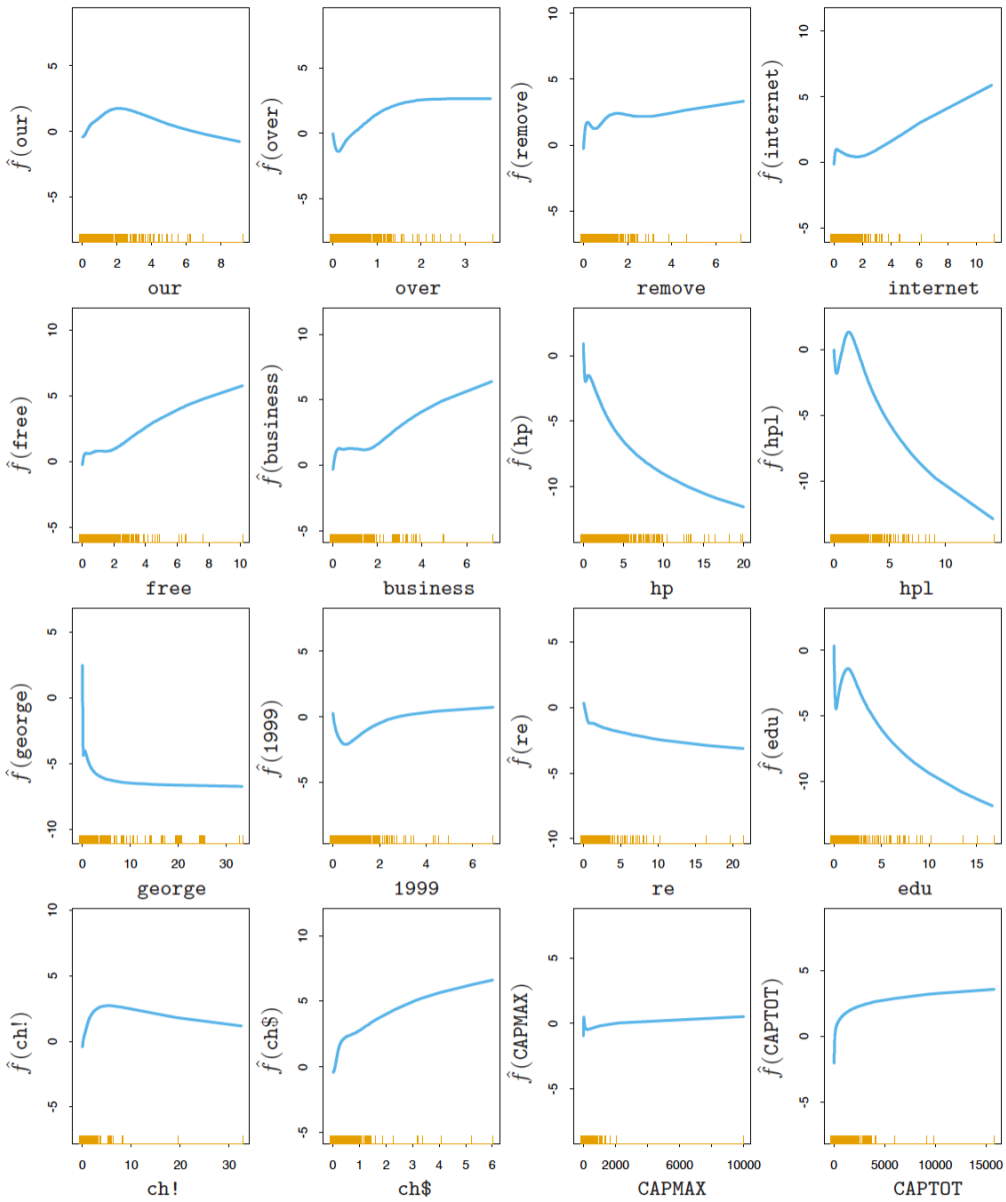


Figure 2.1: A GAM learned on the `spam` dataset from the UCI repository (Dua and Graff, 2017). Figure shows estimated functions for significant predictors. Therug plot along the bottom of each frame indicates the observed values of the corresponding predictor. Figure credit: (Hastie et al., 2009).

2.4 Decision Trees

The main limitation of linear models is related to all the scenarios in which the association between features and outcomes is nonlinear or involves interactions among features. In these cases, in the context of supervised learning, Decision Tree (DT) models are widely used alternatives for trasparent machine learning. Formally, a DT is a connected and acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and arcs respectively. DTs iteratively divide the dataset based on specific threshold values within the features. This process involves creating distinct subsets of the data, where each instance is assigned to a particular subset through successive splits. For this purpose, the set $\mathcal{V}$ is divided in two subsets $\mathcal{B}$ and $\mathcal{L}$ which represent the sets of *branch* nodes and *leaf* nodes. Each branch is responsible for the creation of the split at its level and it splits data in different subsets of points that will be forwarded to different children. Finally, terminal nodes i.e., leaf nodes, encode the value of the prediction for all points that have reached the leaf itself. If the output space $\mathcal{Y}$ is a finite discrete set, these models are called Classification Trees (CTs) otherwise, if $y \in \mathbb{R}$, they are generally called Regression Trees (RTs).

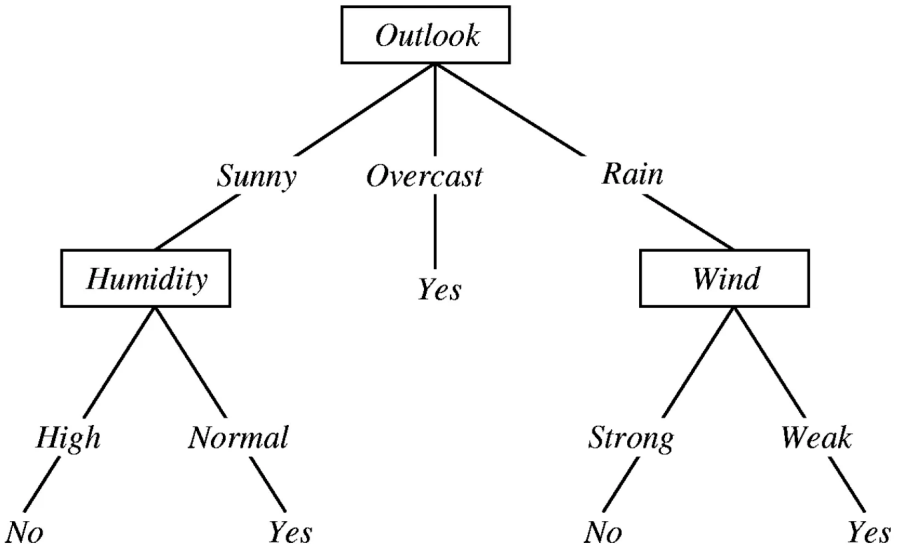


Figure 2.2: A decision tree model for rain forecasting.

The major advantage of DTs is their intrinsic interpretability. Indeed, each prediction is basically represented by the path followed by the point in the tree which is the sequence of simple decision rules taken at each branch node. Moreover, the final user can understand how different feature contribute to the final decision. An example of the ease in the model uderstanding can be seen in Figure 2.2 where a classification tree has been learned for the task of rain forecasting.

In the rest of this thesis we will focus on binary decision trees with the following differences:

- **Axis-Aligned Decision Trees**: also known as decision trees with parallel splits, are a type of decision tree where the decision boundaries are aligned parallel to the axes of the feature space. In other words, the splits along the nodes of the tree are made along one feature at a time, and each split is perpendicular to the axis of that specific feature. This simplicity makes axis-aligned decision trees easy to interpret and computationally efficient. However, it may struggle with capturing complex relationships in the data when the decision boundaries are not aligned with the axes. An example of this kind of structures is visible in Figure 2.3 (B);
- **Oblique Decision Trees**: also known as multivariate decision trees, on the other hand, allow for more flexible decision boundaries that are not restricted to aligning with the axes of the feature space. In oblique decision trees, the splits are made through a general hyperplane $\boldsymbol{w}^T\boldsymbol{x} + \beta$, providing the model with the ability to capture more complex relationships between features. This flexibility allows oblique decision trees to potentially achieve better predictive performance on datasets where the underlying patterns are not well-aligned with the coordinate axes. However, the interpretability of oblique decision trees may be clearly compromised. An example of this kind of structures is visible in Figure 2.3 (A);

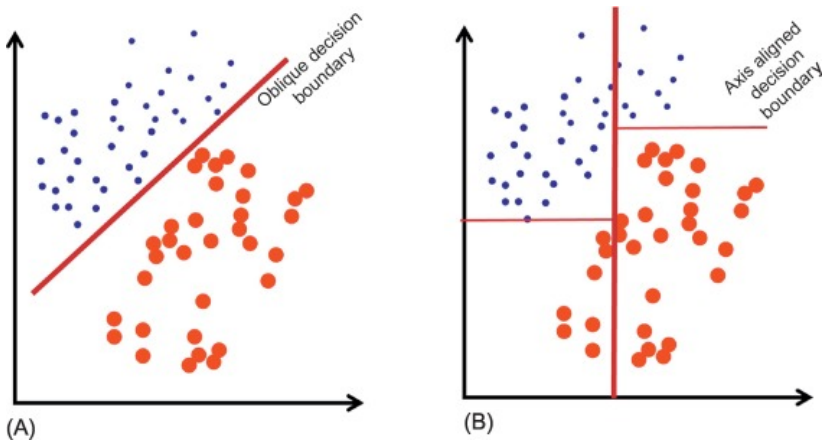


Figure 2.3: Two different classification tree on the same data. In figure (A) a single node multivariate CT is able to linearly separate the two classes. In figure (B) the feature space is divide by a CT of depth two in four regions.

Chapter 3

Optimization for Classification Trees

As previously mentioned in Chapter 2, in the context of supervised learning, Classification Trees (CTs) are some of the most widely used models for classification problems (Kotsiantis, 2013; Song and Ying, 2015). Introduced in the seminal work in Breiman et al. (1984), CTs have widely been employed for decades, especially for tasks with small-sized tabular data.

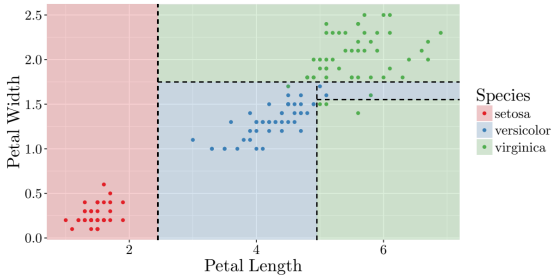
The induction of optimal decision trees is a $\mathcal{NP}$-complete problem (Laurent and Rivest, 1976) and, for this reason, most algorithms employ a top-down greedy approach to fit the model. Most interpretable classification trees are based on *univariate* (axis-aligned) *splits* (Breiman et al., 1984; Quinlan, 1986; De Mántaras, 1991). With univariate splits, a single feature is chosen at each node and the corresponding value of each data point is compared to a given threshold; data points are thus forwarded to one of the children nodes, based on the result of this comparison. Each prediction of the model is therefore finally obtained following a path along the tree, based on the sequence of "decisions" at each encountered node. Thus, it's not surprising that this kind of models are very common and useful especially in the healthcare context (Intrator et al., 1992; Olanow et al., 2001; Bertsimas et al., 2019) and every time the model understanding is a major requirement (Rudin, 2019).

In order to improve the expressive power of CTs, however, more complex splitting rules have been also considered in the literature for defining branching splits. These rules can take into account linear (in this case we talk about *oblique* trees) or even nonlinear relations between features (Friedman et al., 1977; Loh and Vanichsetakul, 1988; John, 1995). However, even the simplest *oblique* trees, i.e., CTs using general linear classifiers at each node to define splits, are not as interpretable as univariate ones. For this reason, recent research has dealt with training algorithms to construct CTs having both good predictive performance and intrinsic interpretability properties. In particular, oblique trees are often considered interpretable as long as branching linear classifiers are actually sparse models (Ross et al., 2017; Ribeiro et al., 2016; Jovanovic et al., 2016).

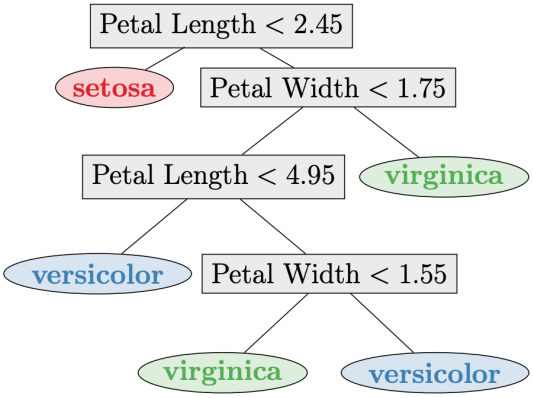
Within this scenario, starting with the best known greedy algorithm CART (Breiman et al., 1984), we present state-of-art methodologies to address the optimization for classification tree learning.

3.1 CART

The goal of the CART algorithm is to recursively split the dataset into subsets based on the values of input features and labels. These splits are determined in a greedy way that maximizes the "homogeneity" of the target variable within each subset. Thus, from the root node, the algorithm iteratively determines a split by solving an optimization problem to identify the most "effective" division. In this framework, the concept of most effective split is related to a proper choice of an impurity measure (discussed later in this section) that acts as a proxy for the quality of the cut. This process is repeated recursively until a stopping condition is met and, finally, to each leaf is associated the label of the class which is the most present in the subset of points arriving on the leaf itself. Figure 3.1 shows two different views of the tree model induced by CART on the Iris dataset (Dua and Graff, 2017). It is easy to see how the feature space is partitioned in an hierarchical manner as the result of the five leaves generated by the algorithm.



(a) The axis-aligned feature space partition generated by the tree in Figure 3.1b.



(b) The resulting tree structure of depth four.

Figure 3.1: Different views of the tree model learned by the CART algorithm on the Fisher's "Iris" dataset from the UCI Repository (Dua and Graff, 2017). Figure credit: Dunn (2018).

For the sake of completeness, we report in algorithm 1 the pseudocode of a standard CART scheme.

The scheme of the algorithm is very similar to a *Depth-First Search* where new nodes are generated by solving the inner purity optimization problem and finally

Algorithm 1 CART scheme

Input: Train data $\mathcal{D}$
Initialize the tree $\mathcal{T}$ with the root node N_0;
$\mathcal{T} = N_0$;
Initialize a list for exploration $\mathcal{L} = N_0$;
while $\mathcal{L} \neq \emptyset$ **do**
    $N = \mathcal{L}.pop()$
    Generate two children nodes N_l and N_r solving the splitting problem for a chosen metric on node N
    if a chosen stopping criteria holds **then**
        $N.leaf = True$
    else
        $\mathcal{L}.push(N_r)$
        $\mathcal{L}.push(N_l)$
    end if
end while

added to the tree only in the case they violate the criteria imposed to stop the growing. Each node, for which a stopping criterion applies, becomes a leaf and it is associated with the label of the most numerous class among the examples that arrive to it.

Below we discuss both the most typical purity measure to determine the split and common choices for the stopping criteria.

Splitting Criteria for Classification

Each iteration of CART involves solving of an optimization problem to define the parameters of the cut. We now discuss the two typical measures which are commonly used as impurity function.

Gini Impurity

The Gini impurity quantifies the likelihood of misclassifying a randomly selected element within a dataset. Specifically, it measures the degree of disorder or impurity in a given set of class labels. In the classification context, let $\mathcal{C}$ be the set of possible classes, the Gini impurity $G(\mathcal{D})$ of a dataset $\mathcal{D}$ is defined as:

$$G(\mathcal{D}) = \sum_{c \in \mathcal{C}} p_c(1 - p_c) = 1 - \sum_{c \in \mathcal{C}} p_c^2 \tag{3.1}$$

Where p_c is the relative frequency of the class c in the dataset $\mathcal{D}$. In the case of axis-aligned trees, the algorithm exploits a weighted version of this measure with

respect to the left and right partiction, i.e., let $\mathcal{D}$ be the set of points reaching the current node, each split s divides the feature space and forwards the subset $\mathcal{D}_l(s)$ of points to the left child and the subset $\mathcal{D}_r(s)$ to the right one. At each iteration, CART selects the best split s by solving the optimization problem:

$$\min_s \frac{1}{N}(|\mathcal{D}_l(s)|G(\mathcal{D}_l(s)) + |\mathcal{D}_r(s)|G(\mathcal{D}_r(s))) \tag{3.2}$$

It is easy to observe that the function in the problem 3.2 has 0 as global minimum which is obtained where both the subsets $\mathcal{D}_r(s)$ and $\mathcal{D}_l(s)$ are pure i.e., they contain only points with the same label. Because of the non linear and non convex nature of this objective function, CART solves the problem by enumerating all the possible spits (couples feature/threshold), with a cost of $O(Np)$, assuming that data has been sorted before the training with respect to each feature ($O(pNlogN)$).

Information Gain

Another typical choice for the quality measure is the Information Gain which measures how much information a split provides to predict the target variable. This metric is based on the entropy E of data $\mathcal{D}$, that is:

$$E(\mathcal{D}) = -\sum_{c \in \mathcal{C}} p_c log p_c \tag{3.3}$$

Then, given a split s, we define the average entropy with respect to each subset of data $\mathcal{D}_l(s)$, $\mathcal{D}_r(s)$ forwarded by the split on the left and right child respectively:

$$\bar{E}(\mathcal{D}(s)) = \frac{1}{N}(|\mathcal{D}_l(s)|E(\mathcal{D}_l(s)) + |\mathcal{D}_r(s)|E(\mathcal{D}_r(s))) \tag{3.4}$$

Finally the information gain $I(\mathcal{D}(s))$ is defined as:

$$I(\mathcal{D}(s))) = E(\mathcal{D}) - \bar{E}(\mathcal{D}(s)) \tag{3.5}$$

The goal is to maximize information gain, which corresponds to minimizing the average entropy on the children nodes.

This metric is commonly used, in addition to CART, in other greedy algorithms such as ID3 (Quinlan, 1986) and C4.5 (Quinlan, 2014). Also in this case, the typical strategy to obtain the global optimum is through enumeration.

Figure 3.2 shows the graphs of the discussed impurity measures. Also, it has been reported the misclassification error. The fact that CART uses impurity measure

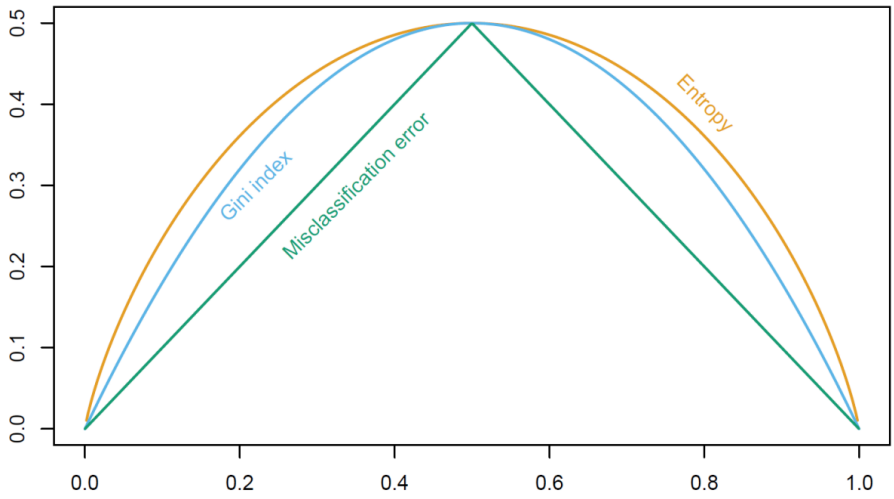


Figure 3.2: Different impurity measures in the case of binary classification tasks (y-axis). The x-axis is the probability that an instance belongs to the positive class.

when selecting splits, rather than using directly the misclassification error, which is, in any case, the final objective to the aim of predictions, may be a little bit strange. To understand the reason for this choice on this latter metric, it is worth to mention the explanation reported in (Breiman et al., 1984):

...the [misclassification] criterion does not seem to appropriately reward splits that are more desirable in the context of the continued growth of the tree. . . . This problem is largely caused by the fact that our tree growing structure is based on a one-step optimization procedure.

Thus, using directly the misclassifcation loss when bulding the tree in a greedy fashion tends to increment the overfitting problem which is the major drawback of this kind of models.

Stopping Conditions

For now, the described algorithm it is able to iterate by generating each time two new children: the left and right node. This process may continue also until leaves containing a single example of the dataset have been created. In this case, the final model would be able to generate a feature space partition which is the most fine grained, i.e., the estimator would reach the perfect accuracy on the train data at the expense of a strong overfitting in out-of-sample predictions. For this reason, it is very important to estabilish robust stopping criteria during the growing phase. We now discuss different stopping criteria that are mostly employed for the implementation of CART.

- **Maximum depth**: the algorithm terminates when the tree reaches a given maximum depth;
- **Minimum number of sample in a split**: the algorithm terminates where all the nodes have a number of samples that is less than this parameter. Each node with a number of samples less than this threshold becomes a leaf;
- **Minimum number of samples in a leaf**: the algorithm terminates when each node which has not yet been split would generate two nodes with fewer points than this parameter. Thus each node of this kind will become a leaf;
- **Minimum value for impurity decrease**: the algorithm terminates when there are not other nodes to split that can bring a reduction in their impurity of at least this parameter.

When a splitting criterion is no longer met, a node becomes a leaf node, and a class label is assigned based on the majority class of the instances in that node.

Pruning

After the tree is fully grown, pruning may be applied to avoid overfitting. Pruning strategies play a crucial role in refining decision trees, ensuring they generalize well to unseen data and avoid overfitting. Pruning involves the removal of specific branches or nodes from a fully grown tree to enhance its simplicity and predictive accuracy. Two primary approaches to pruning exist: pre-pruning and post-pruning. Pre-pruning, also known as early stopping, involves the usage of the aforementioned criteria during the tree-building process to halt further growth.

Post-pruning, on the other hand, takes place after the tree has been fully grown. It entails assessing the impact of each subtree on the model's performance and removing branches that contribute less significantly to predictive accuracy. Common metrics for post-pruning include cross-validation techniques, where the tree's performance is evaluated on validation sets. Pruning strategies strike a delicate balance, preventing the model from becoming overly complex and capturing noise in the training data, thereby fostering a more robust and generalizable decision tree.

Limitation

The primary drawback of the top-down approach utilized by CART (Breiman et al., 1984), C4.5 (Quinlan, 2014), and ID3 (Quinlan, 1986), lies in its inherently greedy nature, as each split in the tree is determined in isolation without considering the potential impact of future splits. This approach may result in trees that inadequately capture the underlying characteristics of the dataset, potentially leading to suboptimal performance in classifying future data points.

In next sections we discuss some techinques that aim to search for the optimal classification tree in a global perspective.

3.2 Exact Formulations

The sub-optimal structure induced by greedy methods such as CART, led, during the years, to the need for new algorithms able to explore the global space of feasible solutions. In particular, exact formulations of the learning problem exploiting linear programming have been considered. Back in the early '90s, Bennet et al. proposed linear methods for constructing separation hyperplanes (Bennett and Mangasarian, 1992, 1994) and for the induction of oblique classification trees solving an LP problem at each branch node (Bennett, 1992).

More recently, thanks to the outstanding improvements in both hardware power and software solvers capabilities (Bixby, 2012), new formulations of the learning problem have been developed. These formulations are related to both univariate and multivariate splits, and exploit various mathematical optimization techniques (Carrizosa et al., 2021) for an effective modelling of the learning problem. One particularly prominent approach involves the use of Mixed Integer Linear Programming (MILP) formulations. The seminal work in this context is the one by Bertsimas and Dunn (Bertsimas and Dunn, 2017), where an exact MILP model is proposed that can be solved up to a certifiably globally optimal CT (OCT) structure (in terms of misclassification error).

Inspired by this work, a plethora of improved MILP-based approaches followed; among them, we can notably cite a formulation of the problem for the case of binary classification with categorical features (Günlük et al., 2021), or a flow-based formulation exploiting linear relaxation and Bender's decomposition to efficiently handle the specific case of binary classification with binary features (Aghaei et al., 2021).

Focusing on the interpretability aspect of CTs, some works take advantage of regularization terms to limit the number of branch nodes (Bertsimas and Dunn, 2017) and total number of leaves in the model (Hu et al., 2019; Lin et al., 2020). The above formulations can in principle generate certified globally optimal structures. In practice, however, there is a combinatorial explosion in the number of binary variables as the depth of the tree or the size of the dataset increases. Thus, optimality gap can be closed by branch-and-bound type procedures only for really shallow trees and on tasks with a few hundred examples at most.

Recently, different losses have also been investigated instead of the standard misclassification error. Using a mixed-integer quadratic programming (MIQP) formulation, D'Onofrio et al. (D'Onofrio et al., 2022) proposed a formulation of the CT learning problem including the concept of maximum margin. This approach allows to define OCTs where each branch node is a maximum-margin linear classifier. The

maximum margin property, obtained minimizing the hinge loss at all branch nodes, allows for significant improvements on generalization performance.

In this section, we introduce state-of-art MIP strategies for learning oblique classification trees with particular regard to the sparsity of coefficients as a proxy for interpretability. We first show that the most relevant formulations in the literature can be seen as being part of a more general framework of *loss optimal* classification trees. Next we will show how to deal with logistic loss in a MILP framework with the aim of induction of *Optimal Logistic* Classification Trees (OLCT) (Aldinucci and Lapucci, 2024) i.e., oblique trees with a logistic classifier at each branch node.

Preliminaries

In the binary classification context, let $\mathcal{I} = \{1, ..., N\}$ and $\mathcal{D} = \{(\boldsymbol{x}_i, y_i),\ \boldsymbol{x}_i \in \mathbb{R}^p,\ y_i \in \{-1, 1\},\ i \in \mathcal{I}\}$ be a finite dataset of N observations with p features and binary labels. To describe each element of the MIP problem we introduce the notation, largely based on (Bertsimas and Dunn, 2017; D'Onofrio et al., 2022), which will be used for the formulation of the learning problem. Formally, let $\mathcal{T}$ be the set of branch nodes of a generic (oblique) CT. We denote by d the number of layers of branching nodes of the tree, i.e., its depth, by $\mathcal{H}$ the set $\{1, \dots, d\}$ and by $\mathcal{T}_h$ the set of nodes at depth h, for $h \in \mathcal{H}$.

Each branch node $t \in \mathcal{T}$ is characterized by a linear function $\boldsymbol{w}_t^T \boldsymbol{x} + b_t,\ \boldsymbol{w}_t \in \mathbb{R}^p,\ b_t \in \mathbb{R}$, which induces the hyperplane $\boldsymbol{w}_t^T \boldsymbol{x} + b_t = 0$ as decision boundary. This has the effect of forwarding an examined data point $\boldsymbol{x}_i$ to the left child of node t if $\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \leq 0$ and to the right child otherwise. Note that, using this notation, a glass-box axis-aligned CT can be obtained as a special case, imposing $\|\boldsymbol{w}_t\|_0 = 1,\ \forall\, t \in \mathcal{T}$, where $\|\cdot\|_0$ denotes the ℓ_0 pseudo-norm.

Since we are interested in CTs where each node is a binary classifier, we can let the final prediction of the point depend solely on the decision of the last branch node. Indeed, each splitting hyperplane is a linear classifier with decision function $\text{sgn}(\boldsymbol{w}_t^T \boldsymbol{x} + b)$. In view of this, there is no need of tracking the points up until the *leaves*, so that the number of nodes to be modelled can be reduced of 2^d units, or, in other words, cut to a half. An example of an oblique CT with branching classifiers of depth 3 is shown in Figure 3.3.

We now introduce the essential building blocks of the MIP models proposed in the literature for OCT training. Recalling that $\mathcal{T}_d \subset \mathcal{T}$ is the set of nodes of the last branching layer, the routing of each point to the proper branch of the last layer can be modelled introducing the binary variables:

$$z_{i,t} = \begin{cases} 1 & \text{if example } \boldsymbol{x}_i \text{ reaches node } t \in \mathcal{T}_d, \\ 0 & \text{otherwise.} \end{cases}$$

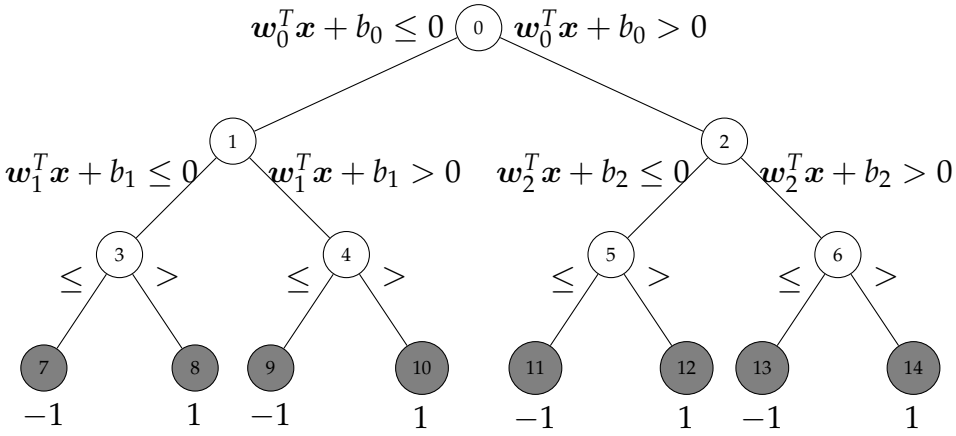


Figure 3.3: An oblique tree of depth 3. Branches are on white background, leaves on gray.

Using these variables, we can force each point to be routed to exactly one of the last branching nodes, complying with the structure of the model. In particular, this can be obtained imposing the following constraints:

$$\sum_{t \in \mathcal{T}_d} z_{i,t} = 1 \qquad \forall\, i \in \mathcal{I}, \tag{3.6a}$$

$$z_{i,t} \in \{0,1\} \qquad \forall i \in \mathcal{I},\ t \in \mathcal{T}_d. \tag{3.6b}$$

We then introduce the constraints to take into account the path of each data point, which is fully determined by the sequence of decisions taken at the branch nodes. For this purpose, we introduce the set $\mathcal{T}_d(t) \subseteq \mathcal{T}_d$ as the set of branches at the last layer that are successors of the node t; in other words, nodes in $\mathcal{T}_d(t)$ belong to the last layer of the sub-tree rooted at t; moreover, we define the subsets $\mathcal{T}_d^r(t)$, $\mathcal{T}_d^l(t)$ as a partition of $\mathcal{T}_d(t)$ such that:

- $\mathcal{T}_d^r(t)$ is the set of nodes of the last branching level that belong to the right sub-tree rooted at node t;
- $\mathcal{T}_d^l(t)$ is the set of nodes of the last branching level that belong to the left sub-tree rooted at node t.

Each branch t forwards points to its left child if $\boldsymbol{w}_t^T\boldsymbol{x} + b_t \leq 0$ and to the right one otherwise. This logic can be enforced by means of the following forwarding

constraints:

$$\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \leq M\Big(1 - \sum_{s \in \mathcal{T}_d^l(t)} z_{i,s}\Big) \qquad \forall\, i \in \mathcal{I},\ \forall\, t \in \mathcal{T} \setminus \mathcal{T}_d, \tag{3.7a}$$

$$\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \geq \epsilon - M\Big(1 - \sum_{s \in \mathcal{T}_d^r(t)} z_{i,s}\Big) \qquad \forall\, i \in \mathcal{I},\ \forall\, t \in \mathcal{T} \setminus \mathcal{T}_d, \tag{3.7b}$$

where M is a suitable constant for a big-M type constraint and ϵ is a small constant preventing numerically degenerate cases.

For example, given $t \in \mathcal{T}$ and a data point $\boldsymbol{x}_i$, if $z_{i,s} = 1$ and $s \in \mathcal{T}_d(t)$, then $\boldsymbol{x}_i$ has to be routed to the last branch node s, which either belongs to $\mathcal{T}_d^l(t)$ or $\mathcal{T}_d^r(t)$. Hence, one and only one of the following pair of equations hold:

$$\sum_{s \in \mathcal{T}_d^l(t)} z_{i,s} = 1, \qquad \sum_{s \in \mathcal{T}_d^r(t)} z_{i,s} = 1.$$

If the former holds, then (3.7a) guarantees that $\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \leq 0$, otherwise (3.7b) forces $\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t > 0$. The big-$M$ strategy makes the constraint for the "wrong case" irrelevant. Also, no constraint applies for branch t in case $\boldsymbol{x}_i$ is forwarded to a node $s \in \mathcal{T}_d \setminus \mathcal{T}_d(t)$. Moreover, we shall observe that these constraints are not required if $t \in \mathcal{T}_d$ since the last branching level is only responsible to make the final prediction.

The Generalized Framework

In this section we show how different losses can be encapsulated in a general framework through a proper definition of the slacks variables. For a CT to be meaningful, a suitable *loss function* necessarily has to be used to push the last branching layers and the overall model to correctly classify the data points. This can be done by estimating the errors, or slacks ξ_i, $i = 1, \ldots, n$, committed on each data point:

$$L = \sum_{i \in I} \xi_i, \qquad \boldsymbol{\xi} \in \mathbb{R}^n. \tag{3.8}$$

The definition of quantities $\boldsymbol{\xi}$ clearly depends on the particular loss function to be used. For example, most MIP models for OCTs (Bertsimas and Dunn, 2017; Aghaei et al., 2021; Hu et al., 2019; Carreira-Perpinán and Tavallali, 2018) directly employ the *misclassification loss*, which is defined using additional binary variables and big-M constraints as

$$\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \leq M(2 + \hat{y}_i - z_{i,t}) \qquad \forall i \in \mathcal{I},\ \forall\, t \in \mathcal{T}_d, \tag{3.9a}$$

$$\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \geq -M(2 - \hat{y}_i - z_{i,t}) \qquad \forall i \in \mathcal{I},\ \forall\, t \in \mathcal{T}_d, \tag{3.9b}$$

$$\xi_i = \frac{1}{2}(1 - y_i \hat{y}_i) \qquad \forall i \in \mathcal{I}, \tag{3.9c}$$

$$\hat{\boldsymbol{y}} \in \{-1, 1\}^n. \tag{3.9d}$$

The additional variables $\hat{y}$ model the predicted class for each data point; the slack variable corresponding to each data point will be set to 0 if prediction is correct $y_i\hat{y}_i = 1$, otherwise it will be equal to 1. Constraints (3.9a)-(3.9b) guarantee that, for a point x_i arriving at the leaf t, $\hat{y}_i = 1$ if and only if $\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t \geq 0$.

However, this is not necessarily the only option for a loss function. In fact, different choices not only might be statistically more robust, but may also avoid the introduction of the additional binary variables and logical constraints that increase problem complexity. An example of a continuous loss with these features, easily embeddable in the MIP framework, is the *hinge loss*, $\max\{0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)\}$, that can be modeled as:

$$\xi_i \geq 1 - y_i(\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t) - M\Big(1 - z_{i,t}\Big) \qquad \forall\, i \in \mathcal{I},\ \forall\, t \in \mathcal{T}_d, \tag{3.10a}$$

$$\xi_i \geq 0 \qquad \forall\, i \in \mathcal{I},\ \forall\, t \in \mathcal{T}_d. \tag{3.10b}$$

The objective function can then be defined as the sum of the total loss function L and a regularization term for weights $\boldsymbol{w}_t$, $t \in \mathcal{T}_d$, so that we basically have an empirical risk minimization problem:

$$L + \lambda \sum_{t \in \mathcal{T}_d} \Omega(\boldsymbol{w}_t). \tag{3.11}$$

We note that:

- By setting $\lambda = 0$ and using (3.9), we actually retrieve the standard OCTs problem from (Bertsimas and Dunn, 2017);
- setting $\lambda > 0$, $\Omega(\cdot) = \|\cdot\|_2^2$ and using (3.10) we get "SVM leaves".

In fact, loss terms can also be associated with the upper branching nodes; in this case, we have d vectors of slack variables, $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_d$: along its path to the leaves, each data point encounters only one node at each layer, and thus only the slack associated with the corresponding classifier has to be taken into account. The overall objective function for a general *loss-optimal* CT model is therefore given by

$$\sum_{h \in \mathcal{H}} \left(\sum_{i \in \mathcal{I}} \xi_{i,h} + \lambda_h \sum_{t \in \mathcal{T}_h} \Omega(\boldsymbol{w}_t) \right). \tag{3.12}$$

It is worth noting at this point that the Margin Optimal CT models (MARGOT) from (D'Onofrio et al., 2022) can then be retrieved as a special case of the general framework, setting

$$\xi_{i,h} \geq 1 - y_i(\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t) - M\Big(1 - \sum_{s \in \mathcal{T}_d(t)} z_{i,s}\Big) \quad \forall\, i \in \mathcal{I},\ \forall\, h \in \mathcal{H},\ \forall\, t \in \mathcal{T}_h, \tag{3.13a}$$

$$\xi_{i,h} \geq 0 \quad \forall\, i \in \mathcal{I},\ \forall\, h \in \mathcal{H}. \tag{3.13b}$$

and $\Omega(\cdot) = \|\cdot\|_2^2$.

Exact Modelling of ℓ_0 Terms

In order to enhance the interpretability of the models, sparsity can be compelled within the weights of branching classifiers, thus inducing features selection. The ℓ_0 norm of vectors $\boldsymbol{w}_t$ can easily be modeled, in a MINLP program, by introducing binary indicator variables, big-M constraints and linear expressions:

$$\boldsymbol{\delta}_t \in \{0,1\}^p, \qquad -M\boldsymbol{\delta}_t \leq \boldsymbol{w}_t \leq M\boldsymbol{\delta}_t, \qquad \|\boldsymbol{w}_t\|_0 = \mathbf{1}^T\boldsymbol{\delta}_t. \tag{3.14}$$

Then, the value of $\|\boldsymbol{w}_t\|_0$ can either be upper bounded or penalized. Following the terminology in (D'Onofrio et al., 2022) we talk about Hard Feature Selection (HFS) in the former case and Soft Feature Selection (SFS) in the latter one.

Optimal Logistic Classification Trees

In this section, we formalize a novel, particular instance of loss-optimal CT model, the Optimal Logistic CT (OLCT). In particular, we first show how to introduce the logistic loss function within the considered MIP; then, we point out the benefits of using ℓ_1-regularization terms; we then show the resulting overall optimization model.

Logistic Loss in Mixed Integer Linear Optimization

The logistic loss function for binary linear classifiers is defined as

$$\ell(\boldsymbol{w}^T\boldsymbol{x}_i + b; y_i) = f(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)) = \log(1 + \exp(-y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b))).$$

This loss function appears in the individual terms of the summation in the negative log likelihood function of logistic regression models (see, e.g, (Hastie et al., 2009)), i.e., the linear model for binary classification that is obtained by maximum likelihood estimation under the assumption that data follows a Bernoulli distribution.

In addition to often having strong performance in terms of out-of-sample prediction accuracy, other advantages of logistic regression compared to other linear classifiers, such as SVMs, include

(a) the possibility of obtaining estimates of features importance by simple manipulation of the model weights (Hastie et al., 2009);

(b) the opportunity of getting calibrated probability estimates associated with predictions.

The above properties offer nice insights that are valuable from the perspective of model interpretability. These considerations motivate us to consider the employment of the logistic loss within the general framework for loss-optimal CTs, discussed in the previous section.

The straightforward objection, at this point, concerns the nonlinearity of the logistic loss function; indeed, contrarily to the hinge loss defining SVMs, the log loss cannot exactly be modeled by linear constraints in MILP. However, this issue can be addressed following the strategy, proposed in (Sato et al., 2016), where the best subset selection problem in logistic regression is solved by means of a MILP approach. In particular, Sato et al. proposed to approximate the logistic loss function by a piece-wise linear underestimator (see figure 3.4).

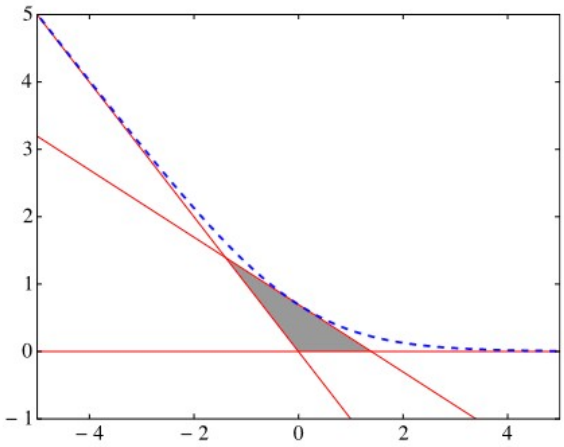


Figure 3.4: Piece-wise linear approximation of the log-loss using V_0 as set of tangent points. Figure credit: Sato et al. (2016).

The function

$$f(v) = \log(1 + \exp(-v))$$

is a convex function, thus its tangent line at a point v_0 constitutes a global underestimator; more explicitly, for all $v, v_0 \in \mathbb{R}$, it holds

$$f(v) \geq f(v_0) + f'(v_0)(v - v_0).$$

To obtain an accurate approximation of the logistic loss, we can thus construct a piece-wise linear underestimator obtained as the point-wise maximum of a family of tangent lines:

$$f(v) \approx \max\{f(v_k) + f'(v_k)(v - v_k) \mid k = 0, \ldots, K\}.$$

Sato et al. also propose a greedy strategy to select points v_k where computing the tangent lines so as to minimize the approximation error: at each iteration, a tangent line is added to the piece-wise linear approximation so that the area between the exact and approximated loss function is minimized. The resulting sets of tangent points to be used for increasingly accurate approximations of the logistic loss are

$$V_0 = \{0, \pm\infty\}, \qquad V_1 = V_0 \cup \{\pm 1.9\}, \qquad V_2 = V_1 \cup \{\pm 0.89, \pm 3.55\},$$
$$V_3 = V_2 \cup \{\pm 0.44, \pm 1.37, \pm 2.63, \pm 5.16\}.$$

Obviously, a larger number of points leads to a larger number of constraints in the MILP model and thus makes it more difficult to solve.

Following this methodology, we can redefine the slack variables to finally introduce the logistic loss in the general framework previosly discussed:

$$\xi_{i,h} \geq f(v_k) + f'(v_k)(y_i(\boldsymbol{w}_t^T \boldsymbol{x}_i + b_t) - v_k) - M\Big(1 - \sum_{s \in \mathcal{T}_d(t)} z_{i,s}\Big), \quad \forall\, i \in \mathcal{I},\ \forall\, h \in \mathcal{H},\ \forall\, t \in \mathcal{T}_h,\ \forall v \in V. \tag{3.15}$$

Lasso Regularization

Together with the loss function defining the slack values, the second element to be chosen in our generalized OCT framework is the regularizer. Using an ℓ_2-regularization term, as for example in MARGOT, has the effect of making the entire problem an MIQP instance. Here, we instead propose to consider a Lasso regularizer (Tibshirani, 1996), i.e., an ℓ_1 penalty term; there are two main reasons for doing so:

- the ℓ_1-norm can be easily handled by linear constraints within a linear programming model; thus, using the ℓ_1-norm instead of the squared ℓ_2-norm we can derive a fully linear model which should be easier to solve;
- exploiting the well-known properties of the ℓ_1-norm (Bach et al., 2012), we can implicitly induce sparsity within branch nodes classifiers, without the need of recurring to explicit (and expensive) ℓ_0-norm penalization as in SFS models.

The ℓ_1-norm can be efficiently handled in a linear program, similarly as what is done in (Figueiredo et al., 2007), setting

$$\boldsymbol{w}_t = \boldsymbol{w}_t^+ - \boldsymbol{w}_t^- \qquad \forall t \in \mathcal{T}, \tag{3.16a}$$

$$\boldsymbol{w}_t, \boldsymbol{w}_t^+, \boldsymbol{w}_t^- \in \mathbb{R}^p \qquad \forall t \in \mathcal{T}, \tag{3.16b}$$

$$w_{j,t}^+, w_{j,t}^- \geq 0 \qquad \forall t \in \mathcal{T},\ \forall\, j = 1, \ldots, p. \tag{3.16c}$$

Basically, $\boldsymbol{w}_t$ is split into its positive and negative parts $\boldsymbol{w}_t^+$ and $\boldsymbol{w}_t^-$. Indeed, constraints (3.16) are satisfied by $\hat{w}_{j,t}^+ = \max\{0, w_{j,t}\}$ and $\hat{w}_{j,t}^- = \max\{0, -w_{j,t}\}$; this solution is such that $\hat{w}_{j,t}^+ + \hat{w}_{j,t}^- = |w_{j,t}|$. If the term $\mathbf{1}^T(\boldsymbol{w}^+ + \boldsymbol{w}^-)$ is minimized in the objective, then it is always guaranteed that $\mathbf{1}^T(\hat{\boldsymbol{w}}^+ + \hat{\boldsymbol{w}}^-) = \|\boldsymbol{w}_t\|_1$.

Lasso regularization is well known not only to induce sparsity and guarantee that the optimization process is well-behaved, but also to be beneficial at tackling overfitting (Hastie et al., 2009; Tibshirani, 1996). Thus, if the sparsity requirement within each node is not set by a specific budget (as in HFS), but it is imprecisely imposed by means of a penalty, then the expedient of setting $\Omega(\cdot) = \|\cdot\|_1$ allows to preserve, at a much lower cost, both the predictive performance of the obtained model and its interpretability.

The alternative SFS strategy from (D'Onofrio et al., 2022) exploits the objective function

$$\sum_{t\in\mathcal{T}}\Big(\Omega(\boldsymbol{w}_t)+C_t\sum_{i\in I}\xi_{i,t}+\alpha_t\max(\|\boldsymbol{w}_t\|_0,B_t)\Big).$$

This kind of penalty on the ℓ_0-norm of weights is modeled with binary variables, see eq. (3.14), highly increasing the complexity of the optimization model. Moreover, here there is an additional hyperparameter to be tuned for each layer, making the cross-validation procedure harder to manage.

The Overall Model

The overall *Optimal Logistic Classification Tree* (OLCT) model proposed in this thesis is obtained putting together the pieces described thus far:

$$\min_{\substack{\boldsymbol{w},\boldsymbol{w}^+,\boldsymbol{w}^-\\ \boldsymbol{b},\boldsymbol{\xi},\boldsymbol{z}}} \sum_{h\in\mathcal{H}}\left(\sum_{i\in\mathcal{I}}\xi_{i,h}+\lambda_h\sum_{t\in\mathcal{T}_h}\mathbf{1}^T(\boldsymbol{w}_t^+ + \boldsymbol{w}_t^-)\right) \tag{3.17a}$$

$$\text{s.t. } (3.6), (3.7), (3.15), (3.16) \tag{3.17b}$$

$$\boldsymbol{\xi_h}\in\mathbb{R}^n \quad \forall\, h\in\mathcal{H}, \tag{3.17c}$$

$$b_t\in\mathbb{R} \qquad \forall\, t\in\mathcal{T}. \tag{3.17d}$$

The constraints referenced at (3.17b) contain almost all the defining elements of the logistic tree: constraint (3.6) defines the indicator binary variables of data points routing, constraints (3.7) enforce the routing logic along the tree, constraints (3.15) define the slacks corresponding to the logistic loss and (3.16) define the branching classifiers weights and model their ℓ_1-norm.

Constraints (3.17c) and (3.17d) simply define the domain of the remaining variables. The objective function is nothing but the general loss (3.12) where the ℓ_1 regularizer is employed, formulated as shown in Section 3.2.

Remark 1. *Note that, in our model, we use a piece-wise linear underestimator, i.e., a surrogate function, to approximate the log-loss. Thus, at the end of the training process, we can actually perform a refinement operation affecting the last layer of branching nodes. Specifically, we can exactly fit an ℓ_1-regularized logistic regression model at each node of the last layer, using as training data the samples that actually reach that node. By this procedure, we are guaranteed that the overall exact objective function associated with the entire tree model decreases.*

The same reasoning cannot be applied with higher level nodes: changes to the branching classifiers possibly change how training data are forwarded to lower nodes, thus affecting the corresponding loss terms. The overall loss associated with the tree might increase, because of the lack of global perspective.

Remark 2. *One of the most appealing features of classification trees is their glass-box nature; however, the highest level of interpretability is only reached in the case of parallel splits. For this reason, we believe it to be important to explicitly point out how to retrieve a univariate optimal logistic classification trees. The model is basically equivalent to* (3.17)*, except for HFS type constraints, with an upper bound on the ℓ_0-norm of each branching classifier set to 1. This can be modeled in MILP terms by constraints* (3.14) *and setting $\mathbf{1}^T \boldsymbol{\delta}_t \leq 1$.*

Of course, the addition of a number of new binary variables and big-M type constraints directly proportional to $p \times |\mathcal{T}|$ is significant in terms of the computational resources needed to solve the problem and especially to certify optimality, closing the optimality gap.

Interpreting OLCTs

Inheriting, at least partially, the nice interpretability properties of logistic regression models is one of the main advantages of using the logistic loss within the OCTs framework.

Specifically, there are some aspects that can be taken into account to retrieve additional information about the model prediction mechanisms.

Evaluation of feature influence

At each branching node of oblique OLCTs, the influence of each individual parameter in the splitting decision can be estimated looking at the magnitude of the corresponding coefficient and multiplying it by the standard deviation of the feature among the training data points reaching that node; formally, given the weights $\boldsymbol{w}_t$ at a node $\bar{t} \in \mathcal{T}$, the influence $r_{j,\bar{t}}$ of feature j can be estimated by:

$$r_{j,\bar{t}} = w_{j,\bar{t}} \sigma_{j,\bar{t}}$$

Where:

$$N_{\bar{t}} = \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}_d(t)} z_{i,t}, \qquad \mu_{j,\bar{t}} = \frac{1}{N_{\bar{t}}} \sum_{i \in \mathcal{I}} \left(x_{i,j} \sum_{t \in \mathcal{T}_d(\bar{t})} z_{i,t} \right),$$

$$\sigma_{j,\bar{t}} = \sqrt{\frac{1}{N_{\bar{t}}} \sum_{i \in \mathcal{I}} \left((x_{i,j} - \mu_{j,\bar{t}})^2 \sum_{t \in \mathcal{T}_d(t)} z_{i,t} \right)}$$

The larger the coefficient $r_{j,t}$ is, the higher is the correlation of feature j with positive outputs; on the other hand, the largest negative values of importance are associated with features pushing the point towards the left child of the branching node. The features influence for an OLCT trained on the `heart` dataset is reported, as an example, in Figure 3.5.

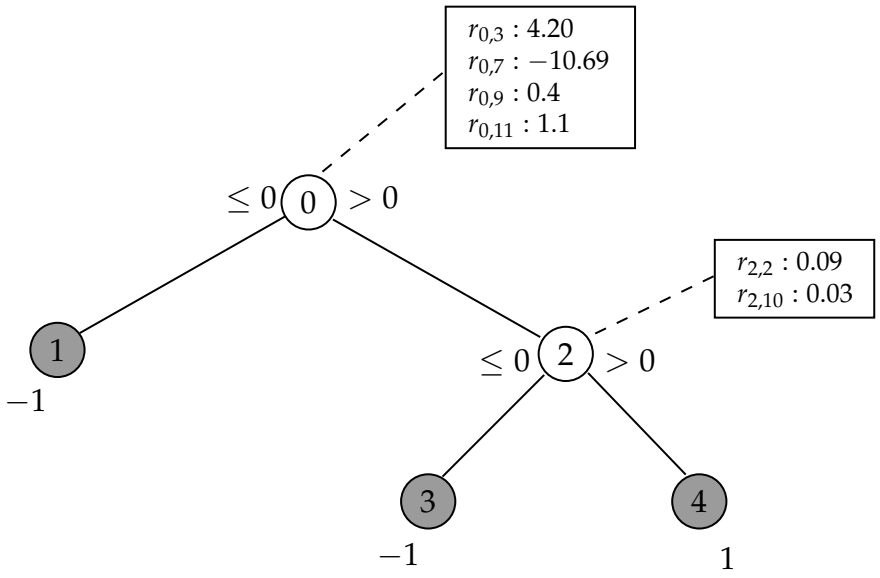


Figure 3.5: A learned logistic oblique tree of depth 2 for the `heart` dataset. Branches and leaves are on white and gray background respectively. Each branch node is a sparse linear regressor; the importance coefficient for each feature involved at each decision is reported in the boxes.

Probabilistic interpretation of outputs

The sigmoid function $s(z) = \frac{1}{1+\exp(-z)}$ is used in logistic regression to map the output of the linear function defined by $\boldsymbol{w}, b$ to $(0, 1)$, so that a data point $\boldsymbol{x}$ can finally be classified as positive if $s(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 0.5$.

In a standalone logistic regression model, these "probability values" are related to the odds of the positive outcome over the negative one; in particular, the linear regressor is designed to model, by maximum likelihood, the log-odds (logits) of the output:

$$\boldsymbol{w}^T\boldsymbol{x} + b = \text{logit}(\mathbb{P}(y = 1 \mid \boldsymbol{x})) = \log\left(\frac{\mathbb{P}(y = 1 \mid \boldsymbol{x})}{\mathbb{P}(y = -1 \mid \boldsymbol{x})}\right).$$

Exponentiating we get

$$\exp(\boldsymbol{w}^T\boldsymbol{x} + b) = \frac{\mathbb{P}(y = 1 \mid \boldsymbol{x}))}{1 - \mathbb{P}(y = 1 \mid \boldsymbol{x}))},$$

and by simple algebraic manipulations we retrieve

$$\mathbb{P}(y = 1 \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}^T\boldsymbol{x} + b)}{1 + \exp(\boldsymbol{w}^T\boldsymbol{x} + b)} = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x} - b)} = s(\boldsymbol{w}^T\boldsymbol{x} + b).$$

In other words, the logistic output is a probability estimate for the positive output, i.e., the logistic model is well *calibrated* by construction.

Within the OLCTs framework, we can exploit this property both at the splitting and the classifying nodes of the tree. If $t \in \mathcal{T}_d$, then the odds associated with the outputs of $\boldsymbol{w}_t$, b_t are actually interpretable as the odds of the positive class over the negative one, given that the data point belongs to the subspace deterministically defined by the splits at the higher nodes. Since each node classifier is calibrated within the corresponding space region, the overall output probability of the OLCT models should also be implicitly well-calibrated, with no need of any extra post-train calibration.

If, on the other hand, $t \in \mathcal{T}_h$, $h < d$, then the classifier has been trained looking not only at its corresponding slacks, but also at those of all its descendant nodes. Thus, in this case we can (somewhat improperly) interpret the probability estimates as the confidence about forwarding the data point to the right child rather than to the left one. This concepts can be visualized through the example in Figure 3.6.

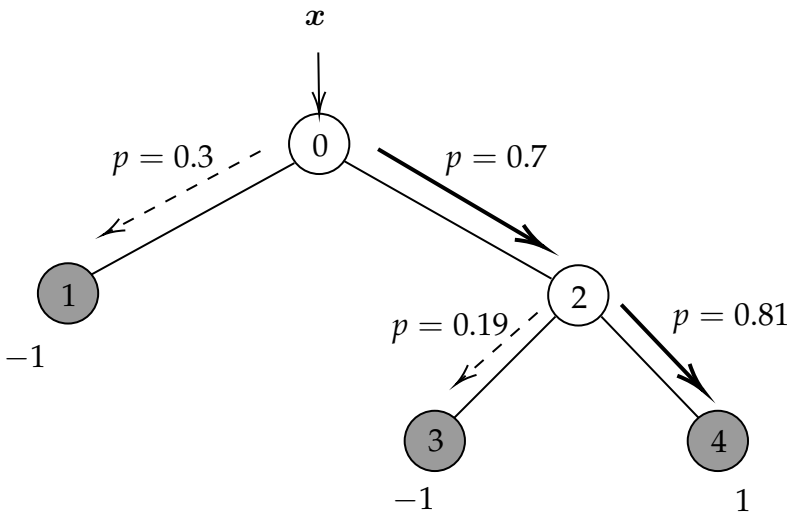


Figure 3.6: Confidence at each branch node of the logistic classification tree from Figure 3.5 for the sample $\boldsymbol{x} = [0.69, 0.87, -1.23, -0.8, -0.4, 1.01, -1.87, 1.4, -0.94, 0.63, 0.35, -0.89, -0.07]$ of the `heart` dataset. The true label y is equal to 1. In this case, the model predicts the correct class of the point and, given the sigmoid activation, we are able to get the confidence (0.7×0.81) of the forwarding decision at each branch node.

Numerical Experiments

In this section, we present the results of computational experiments carried out to evaluate the performance of the proposed approach. All the experiments described in this section have been carried out on a server with an Intel ®Xeon ®Gold 6330N CPU with 28 cores and 56 threads @ 2.20GHz, but we set a limit of only 40 of the

56 available threads and the total available memory is 128GB. The code has been implemented in Python (v. 3.9) and the commercial solver Gurobi (Gurobi Optimization, LLC, 2022) has been used to solve all the mixed-integer programming models considered in this work. All the code is available at `https://github.com/tom1092/Optimal-Logistic-Classification-Trees`.

For each instance of classification tasks, we performed an 80/20 train/test split of the data and we also standardized each feature before the training to zero mean and unit variance. Experiments are always repeated for different random seeds, resulting in different train/test splits. Hyperparameters have been tuned by cross-validation over a grid of values, where the test balanced accuracy (see below) is used as quality metric; more details will be provided for each group of experiments in the following. The parameters M and ϵ in MIP formulations have been set to 100 and 10^{-5} respectively; the value of $M = 100$ is a reasonable value, large enough not to introduce constraints and tight enough not to make Gurobi branch-and-bound too inefficient; this same value was also used, for instance, in (D'Onofrio et al., 2022); as for ϵ, the value 10^{-5} is much larger than the one employed for the IntFeasTol parameter of Gurobi (10^{-9}), so that the issues highlighted in (Liu et al., 2023) did never occur, and at the same time is small enough to let the underlying logic of the constraints work properly - no feasible solution is erroneously cut off. Without loss of generality, we also decided to move the regularization parameter λ_h to the slack component of the loss, to be aligned with the work in (D'Onofrio et al., 2022). The objective function now has the form:

$$\sum_{h \in \mathcal{H}} \left(C_h \sum_{i \in \mathcal{I}} \xi_{i,h} + \sum_{t \in \mathcal{T}_h} \|\boldsymbol{w}_t\|_1 \right)$$

Note that, at the end of the optimization process of OLCTs, we applied the refinement strategy discussed in Remark 1. For each node in the last layer, we retrained the ℓ_1-regularized logistic regression model using `scikit-learn` (Pedregosa et al., 2011) implementation.

To compare the performance of each model, along with the running times, we used the balanced accuracy metric defined as:

$$\mathrm{B_{Acc}} = \frac{\text{Sensitivity} + \text{Specificity}}{2} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right),$$

where TP, TN, FP, FN are the number of true positive, true negative, false positive and false negative outputs, respectively. The $\mathrm{B_{Acc}}$ value is always computed on test set data.

To provide a condensed view of the results, in the following we are making use of performance profiles (Dolan and Moré, 2002). Performance profiles provide a unified view of the relative performance of the solvers on a suite of test problems.

Formally, consider a benchmark of $\mathcal{P}$ problem instances and a set of solvers $\mathcal{S}$. For each solver $\sigma \in \mathcal{S}$ and problem $\pi \in \mathcal{P}$, we define

$$c_{\pi,\sigma} = \text{the cost for solver } \sigma \text{ to solve problem } \pi,$$

where cost is the performance metric we are interested in. In particular, we will be interested in CPU time. We then consider the ratio

$$\eta_{\pi,\sigma} = \frac{c_{\pi,\sigma}}{\min_{\sigma \in \mathcal{S}}\{c_{\pi,\sigma}\}},$$

which expresses a relative measure of the performance on problem π of solver σ against the performance of the best solver for this problem. If a solver fails to solve a problem, we shall put $\eta_{\pi,\sigma} = \eta_M$, with $\eta_M \geq \max\{\eta_{\pi,\sigma} \mid \pi \in \mathcal{P},\, \sigma \in \mathcal{S}\}$.

Finally, the performance profile for a solver σ is given by the function

$$\rho_\sigma(\tau) = \frac{1}{|\mathcal{P}|} \cdot |\{\pi \in \mathcal{P} \mid \eta_{\pi,\sigma} \leq \tau\}|,$$

which represents the estimated probability for solver σ that the performance ratio $\eta_{\pi,\sigma}$ on an arbitrary instance π is at most $\tau \in \mathbb{R}$. The function $\rho_\sigma(\tau) : [1, +\infty] \to [0, 1]$ is, in fact, the cumulative distribution of the performance ratio.

Note that the value of $\rho_\sigma(1)$ is the fraction of problems where solver σ attained the best performance; on the other hand, $\lim_{\tau \to \eta_M^-} \rho_\sigma(\tau)$ denotes the fraction of problems solved from the given benchmark.

In addition to performance profiles, we will also make use of the cumulative distribution of absolute gaps for a given metric μ; in particular, this tool has a similar concept as performance profiles and is obtainable setting

$$\eta_{\pi,\sigma} = |\mu_{\pi,\sigma} - \text{opt}_{\sigma \in \mathcal{S}}\{\mu_{\pi,\sigma}\}|,$$

where the opt operator denotes the minimum or the maximum according to the metric μ selected. Of course, in this case, we have $\rho_\sigma(\tau) : [0, +\infty] \to [0, 1]$. The distribution of absolute gaps is particularly useful when evaluating results in terms of accuracy or objective values.

For example, let us consider a test suite to evaluate the accuracy performance of a set of models on a benchmark of problem instances. From this kind of plot we can infer, for each model, estimates of the probability to obtain an accuracy value distant at most t points from the best one attained by any model. In other words, for any t, we can observe the fraction of times a model reaches an accuracy level within t points from the best one. For $t = 0$, we obtain the fraction of times each model is the best one among all those considered.

Preliminary experiments

The first experiments we carried out concern the assessment of the performance of our model as we vary the set V of the tangent points that are used to obtain the piecewise linear underestimator of the logistic loss. In particular, we are interested in finding a good trade-off between the quality of the approximation and the running time of the model.

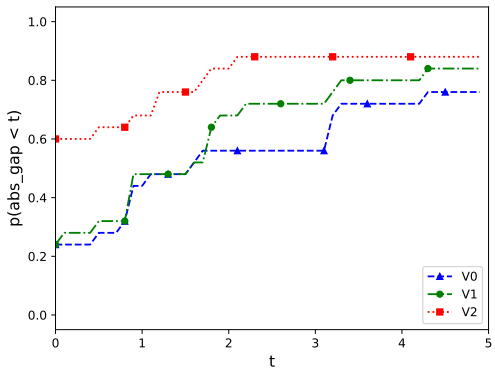
We considered a small benchmark of 5 datasets (`parkinsons`, `wholesale`, `haberman`, `tik-tak-toe`, `sonar`, see Table 3.1) from the UCI repositories (Dua and Graff, 2017), testing V_0, V_1, V_2 with refinment as configurations for the MILP problem of training an OLCT of depth 2. The experiment has been repeated for five different random seeds for each dataset for a total of 25 different problems. The Gurobi time limit has been set to 300s. In these experiments, we did not employ the ℓ_1 regularization terms, as we are mainly interested in assessing the effectiveness of log loss approximation. Note that the MIP solution process is warm-started, initializing the weights of each branch node following a strategy similar to the one adopted in (D'Onofrio et al., 2022): starting from the root in a greedy fashion, we assign to each node weights the values obtained training a logistic regression classifier using the data that are forwarded to that node by the above branches. This strategy allows to obtain significant speedups in computation.

As shown by the performance profiles in Figure 3.7, choosing $V_0 = \{0, \pm\infty\}$ to build the linear piece-wise approximation seems to provide a nice trade-off between running time and solution quality. Indeed, the use of V_1 does not seem to significantly improve the out-of-sample accuracy of the model; on the other hand, the more accurate approximation obtained with V_2 does result in better predictive models, but a much higher training cost has to be paid. For this reason, we decided to use the V_0 setting to assess the performance of our model in the following sections. Nonetheless, we do not rule out that, in certain settings, it might be worth exploiting the increased effectiveness provided by V_2.

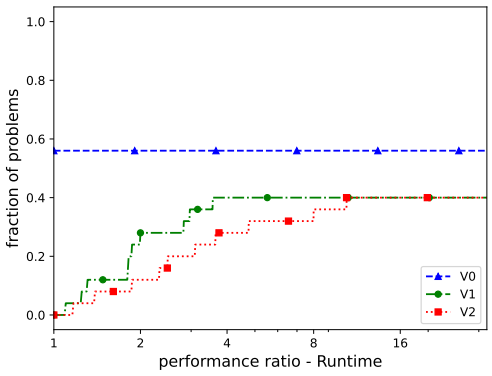
Impact Assessment for the Global Optimization Approach

In this Section, we investigate the actual beneficial effects of conducting a global optimization phase during logistic trees training. Indeed, good performance of OLCTs might be mainly due, in principle, to the warm-start or the refinement steps. In particular, our greedy warm-start procedure constructs a logistic tree model with an iterative top-down approach, which is somewhat similar to the one proposed in (Chan and Loh, 2004) and therefore might actually already lead to an effective classifier. However, the results reported in Figure 3.8 suggest that solving model (3.17) does provide a substantial boost to the performance of the resulting logistic tree.

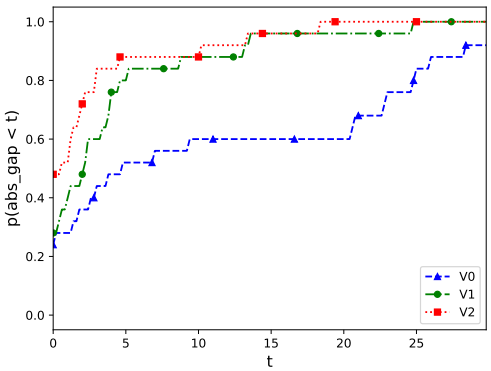
In this experiment, we examined the value of the overall (in-sample) logistic loss



(a) Cumulative distribution of the absolute gap from the best test balanced accuracy value attained by any model.



(b) Performance profiles of the running times for solving the MIP models.



(c) Cumulative distribution of the absolute gap from the best (exact) loss attained on the training set by any model.

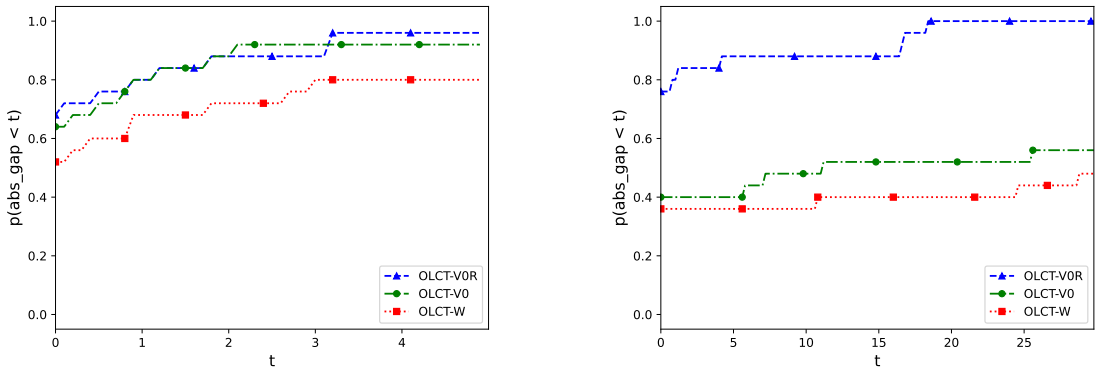
Figure 3.7: Comparison of the performance of OLCT models when different tangent point sets (V_0, V_1 or V_2) are employed. At the end of the optimization process, the refinement of the last layer was carried out for each model.

associated with the model after the warm-start, global optimization and refinement steps, and then we also took into account the (out-of-sample) values of balanced accuracy. Here, we solved the same 25 instances of classification problems considered in Section 3.2. Again, in order to focus on the consequences of approximating the loss in the global step, we did not employ the ℓ_1 regularization terms. We also set Gurobi time limit to 300s. Moreover, based on the results in the previous section, we chose to employ the set V_0 of tangent points in log loss approximation. We did not focus on the running times of the three phases, as the global optimization step obviously represents the main computational burden for our approach.

From Figure 3.8b, we observe that solving the training problem with a global

structure perspective generally allows to slightly improve the overall loss attained by the greedy model. This result has to be underlined, taking into account that the loss function is roughly approximated during the global optimization phase. Then, the refinement step on the last layer allows to really polish the model on training data, often leading to substantially lower values of loss.

The results in Figure 3.8a are even more appealing. Apparently, when it comes to test performance, learning branching rules with a global point of view is crucial to improve the effectiveness of the resulting model. In this perspective, although visible, the positive effect of refinement is much more limited. Thus, we can arguably state that the solving (3.17) as a global optimization has a significant effect in improving the effectiveness of logisitic tree models.



(a) Cumulative distribution of the absolute gap from the best test balanced accuracy value attained by any model.

(b) Cumulative distribution of the absolute gap from the best (exact) loss attained on the training set by any model.

Figure 3.8: Performance of logistic CTs (test balanced accuracy and train logistic loss) obtained carrying out: only the warm start procedure (OLCT-W); warm-start and global optimization with V_0 approximation (OLCT-V0); warm start, global optimization with V_0 and last branch exact refinement (OLCT-V0R).

OLCTs Performance Evaluation

We now present the results of a larger computational experiment where we compare the OLCT model to the MARGOT, SFS-MARGOT (D'Onofrio et al., 2022) and OCT-H (Bertsimas and Dunn, 2017) approaches. To assess the performance of our method, we considered 10 standard binary classification datasets from the UCI repositories (Dua and Graff, 2017) that are reported in Table 3.1. For each dataset, we consider 5 classification problem instances obtained considering different random train/test splits, so that the overall benchmark is made up of 50 test instances. For each problem instance, we set the depth of each tree model to 2. For all models, we

set the Gurobi time limit to 300s both for validation runs and the final fit over the entire training set.

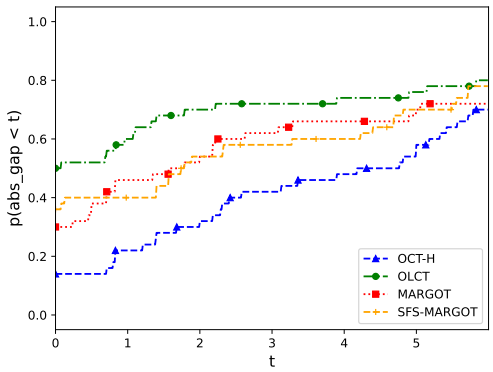
Dataset	N	p
breast	568	30
climate	539	18
haberman	305	13
heart	296	13
ionosphere	350	33
parkinsons	194	22
sonar	207	60
spectf	266	44
tik-tak-toe	957	27
wholesale	439	7

Table 3.1: Description of the datasets used in the computational experiments. All datasets are from the UCI collection (Dua and Graff, 2017).

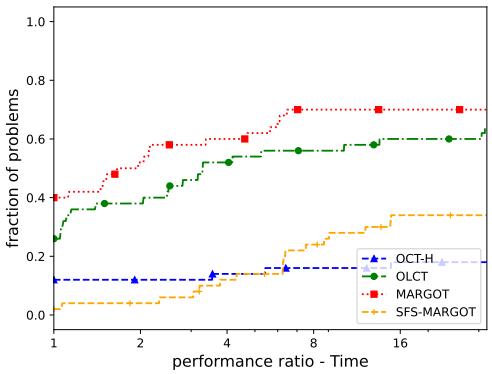
For hyperparameters tuning, this time we used a 4-fold cross-validation. For both OLCT and MARGOT, we considered the slack parameters $C_h \in \{10^i,\ i = -2, -1, ..., 2\}\ \forall\ h \in \mathcal{H}$, obtaining 25 possible model configurations. On the other hand, the SFS-MARGOT model has two hyperparameters for the classifiers at each level: the slack parameter C_h and the ℓ_0 penalty parameter α_h. In order to consider a comparable grid of configurations in size as that of OLCTs and MARGOTs, we used the same α for each branch layer h letting C_h vary in $\{10^{-2}, 1, 10^2\}$ and α in $\{10^{-1}, 1, 10\}$, so that a total number of 27 models is considered for the SFS variant. Finally for OCT-H we used the grid $\alpha \in \{2^i,\ i = -8, -1, ..., 2\} \cup \{0\}$ to tune the α parameter that penalizes the number of features used at each branch node to make the decision.

Again, we initialize the training phase of each MIP model by injecting a *warm start* solution, obtained training logistic or SVM classifiers, depending on the particular tree classifier. For OCT models we used an analogous greedy strategy: we solve the MILP problem at each individual single node, i.e., setting the depth of the tree equal to 1 and only using the set of points reaching the considered branch node, with a time limit of 30s. As also mentioned in (Bertsimas and Dunn, 2017), this strategy can significantly speedup the optimization, providing a good initial upper bound of the loss that may help the branch and bound method.

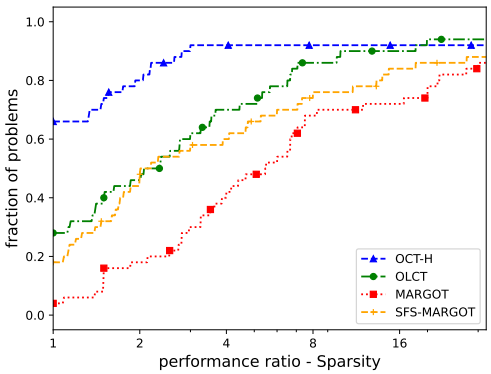
The results of the experiment are shown in Figure 3.9 in the form of cumulative distribution of absolute gap from the best balanced accuracy on the test set and performance profiles (Dolan and Moré, 2002) of runtime and sparsity. Sparsity is measured by the average number of features used at each node of the tree, and constitutes a proxy measure for interpretability of oblique trees.



(a) Cumulative distribution of the absolute gap from the best test balanced accuracy value attained by any model.



(b) Performance profiles of the running times for solving the MIP models.



(c) Performance profiles of the average number of features considered at each split of the classification tree.

Figure 3.9: Comparison of the performance of OLCT, MARGOT and MARGOT-SFS models, on a benchmark of 50 problem instances (10 datasets, 5 random seeds for train/test split).

Observing Figure 3.9a we can infer, for each model, estimates of the probability to obtain a test balanced accuracy value distant at most t points from the best one attained by any model. In other words, for any t, we can observe the fraction of times a model reaches an accuracy level within t points from the best one. This kind of graph is very informative. For example, for $t = 0$ we obtain the fraction of times each model is the best one among all those considered. From this point of view, we observe that our proposed model attains the top accuracy in about 50% of cases; moreover, it also appears to be the most robust, consistently being the most likely to obtain an accuracy value close to the best one, as the gap parameter t increases. Our model is thus not only most frequently the best one, but when it is not, it is still the

one with the lowest probability of falling shorter than any given threshold from the best result.

Another interesting observation is that both SFS-MARGOT and MARGOT exhibit similar performances. That is, the SFS variant is able to produce sparse trees without drastically reduce the out-of-sample prediction performance. In this regard, from Figure 3.9c we can observe that OLCT outperforms both MARGOT and SFS-MARGOT in terms of sparsity, i.e., interpretability.

The high average levels of sparsity in OLCTs branches support the effectiveness of the ℓ_1-regularization approach to this aim. This is of course not particularly surprising, as the effects of LASSO regularization are well known. Yet, the ℓ_1-regularization approach appears to also be able to somewhat outperform the SFS strategy based on ℓ_0 penalization: this result was not granted and is certainly worth to be remarked, even more so taking into account that it is coupled with the efficiency advantage due to the avoided use of binary variables.

On the other hand, the OCT-H approach proves to be the best one in terms of sparsity. This result is driven by two elements within the model as described in (Bertsimas and Dunn, 2017): the penalty term in the objective aimed to encourage splits considering a low number of features, and a constraint in the formulation which forces the weights of each split to have a unitary ℓ_1 norm. However, as a consequence of the combined use of these strategies, the final model tends to be "over-regularized", resulting the clearly worst one in terms of balanced accuracy as shown in Figure 3.9a.

Finally, in Figure 3.9b, performance profiles of the running times highlight that the vanilla MARGOT is, in general, the most likely model to close the optimality gap in less than 300s. However, OLCT appears to have a comparable cost, whereas the SFS-MARGOT and the OCT-H approaches are much more computationally demanding since these latter models make use of many more binary variables. By the way, it is worth to notice that no model was able to close the optimality gap in more than 70% of the instances with a time limit of 300s.

Summarizing, results highlight that our method is able to outperform other approaches in terms of balanced accuracy, to increase the interpretability, inducing sparser structures and exploiting the well known logistic properties discussed in Section 3.2, and finally that these improvements can be achieved in competitive running times with respect to the other approaches.

Performance Analysis with Larger Scale Problems

The mixed-integer optimization models associated with loss-optimal classification trees grow very fast in size as the number of data points or nodes layers increases. In particular, the number of integer variables (and of constraints) grows linearly with the number of data points and exponentially with the trees depth.

This increment is problematic from the perspective of solving the training optimization problem, as even the most efficient solvers from the state-of-the-art struggle when the number of integer variables becomes large.

We are thus interested in conducting, at least, a preliminary assessment of the scalability of the OLCT approach compared to the behavior of MARGOT and OCT-H models. In this larger scale setting, we extended the time limit to 3600 seconds for Gurobi during the final fitting of the models, after the cross-validation phase.

First, we considered 9 additional datasets[1], whose size is reported in Table 3.2. Since the computational burden to carry out the present experiment is significant, we only considered a single train-test split for each dataset. The comparison concerns the OLCT model with the V_0 tangent points set, the MARGOT model and the OCT-H model, setting trees depth to 2. The results of the experiment are reported in Table 3.2. We do not report Gurobi running times as for no instance optimality of the solution was certified. In fact, for all models, the optimality gap was consistently very close to 100% when the time limit of 3600s was reached.

We can observe that OCT-H heavily struggled on these problems, producing meaningful models only with the least large instances of the benchmark. In the most difficult problems, neither the warm start and the global steps were able to find, within their time limits, anything better than a trivial tree forwarding all the points up to the same leaf, leading to a value of B_{Acc} of 0.5. On the other hand, even if unable to certify optimality, both OLCT and MARGOT were able to handle the datasets. The out-of-sample performance of the two approaches is close, with OLCT apparently having a slight advantage; yet, the size of the benchmark does not allow us to state that one model is better than the other.

	B_{Acc}		
Dataset (N, p)	OLCT - V_0	MARGOT	OCT-H
a6a (11220, 122)	**74.49**	73.57	50.00
german (1000, 24)	**69.76**	**69.76**	65.60
a5a (6414, 122)	76.72	**77.50**	50.00
w5a (9888, 300)	**79.91**	78.97	50.00
phishing (11055, 68)	**94.44**	93.93	50.00
w4a (7366, 300)	83.02	**86.62**	50.00
splice (1000, 60)	**88.14**	83.16	80.26
svmguide1 (3089, 4)	**94.98**	94.62	94.52
svmguide3 (1243, 22)	66.86	**67.91**	63.62

Table 3.2: Performance of depth-2 trees fit based on different OCT models on larger scale datasets (available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`). Balanced accuracy (B_{Acc}) is reported for an 80/20 train/test split. We set 300s and 3600s as time limits for the validation and the training phase respectively.

[1] datasets are available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

We then proceeded to evaluate the models as the depth of the trees increased. Specifically, we conducted experiments for trees with a depth of 3. It is important to note that, in this scenario, the cost of the experiments dramatically rises. The addition of an extra layer of nodes necessitates tuning an additional hyperparameter. Consequently, the number of hyperparameters configurations to be considered in the cross-validation phase increases accordingly, rendering the entire process time-consuming. As a result, we only considered 5 datasets from Table 3.1 with a single train/test split.

The results of the experiment are reported in Table 3.3. We can observe that the loss-optimal classification tree framework remains manageable when used to train more complex tree classifiers with a depth of 3. In fact, the produced models demonstrate generalization capabilities, underlining the effectiveness of the procedure. OLCT models appear to have a slight advantage over MARGOT and OCT-H, both in terms of accuracy and ease of training. However, it is important to note that the size of the benchmark considered is too small to draw definitive conclusions in this regard.

	OLCT - V_0		MARGOT		OCT-H	
Dataset	$\mathbf{B_{Acc}}$	**Time**	$\mathbf{B_{Acc}}$	**Time**	$\mathbf{B_{Acc}}$	**Time**
parkinsons	**89.83**	223.66	88.28	3600	83.28	3600
wholesale	**88.69**	3600	**88.69**	3600	87.86	3600
haberman	58.06	3600	54.03	3600	**61.18**	3600
ionosphere	81.78	3600	**83.78**	3600	80.67	3600
spectf	**63.53**	19.12	59.09	3600	53.28	14.9

Table 3.3: Performance of depth-3 trees fit based on different OCT models. Balanced accuracy (B_{Acc}) on test set and running times are reported on an 80/20 train/test split. We set 300s and 3600s as time limits for the validation and the training phase respectively.

3.3 The Evolutionary Approach

Given the computational complexity of MIP formulations and their observed difficulties in handling large datasets and deep trees, another stream of research that attempt to induce the global structure of the tree, without guarantee of optimality, involve the use of Evolutionary Algorithms (EAs) and Genetic Algorithms (GAs) (Eiben et al., 2003).

The general idea of these methods is based on principles of natural selection and genetics (Holland, 1992). Starting from a population (set) of individuals (solutions), at each iteration a new solution is defined through selection and cross-over operations. The quality of the obtained candidate is then assessed by means of a fitness function, thus deciding whether it can replace another solution in the current

set (Jones, 1998). Due to the absence of assumptions about the objective function, these strategies have proven to be very effective in solving global optimization problems and, over the years, many algorithms of this type have been developed (Storn and Price, 1997; Kennedy and Eberhart, 1995; Dorigo et al., 2006). Given the popularity of these techniques, evolutionary approaches have also been proposed both for data mining problems and for knowledge discovery (Freitas, 2003).

Also in the context of decision tree learning, genetic algorithms have been explored. Instead of building the model by exploiting the standard greedy strategy, this family of algorithms performs a more robust search in the global space of solutions (Barros et al., 2011). In particular, in (Cantú-Paz and Kamath, 2003) a top-down method for inducing oblique decision trees is presented which uses an evolutionary approach to solve the hyperplane selection problem at each node, showing that their strategy can build oblique trees whose accuracy is competitive with respect to other methods. In (Papagelis and Kalles, 2001) is proposed a population-based evolutionary method that dynamically evolves a set of minimal decision trees using a fitness function which is balanced between the accuracy on the known instances and the complexity of the new solution.

An extension of evolutionary algorithms is the memetic variant (Moscato et al., 1989). This technique acts by performing local searches or refinement methods to improve the quality of new solutions. Memetic strategies are quite common in the field of global optimization (Jia et al., 2011; Wu and Che, 2019) and, recently, they have been also proposed for solving clustering problems (Mansueto and Schoen, 2021).

Given the empirical effectiveness of this approach, the memetic variant has also been investigated for decision trees in (Kretowski, 2008; Czajkowski and Kretowski, 2012). However, local searches employed in these early works operate at an inner nodes level, greedily exploiting node information during the mutation step. The complete tree structure is never considered as a whole during the optimization process. We deem that the local optimization phase has to be run on the whole tree once that a new solution has been defined and, for this reason, in this section we describe a novel genetic algorithm, *Tree Memetic Optimization* (TMO) (Aldinucci, 2023), that evolves a set of feasible classification trees and exploits local searches to further optimize the whole structure of each new defined solution. Our method employs a specialized local optimizer, *Tree Alternating Optimization* (TAO) (Carreira-Perpinán and Tavallali, 2018), that is able to refine the whole structure encouraging the exploitation of local optimization. This methodology allows a better quality set of solutions to be maintained during the iterations of the algorithm.

Preliminaries

Before introducing our method, it is worth recalling the standard scheme of any genetic algorithm in the case of optimization problems for scalar function. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a generic objective function (we do not require any kind of regularity on this objective). Without any guarantee of global optimality, genetic algorithms aim to produce near-optimal solutions of the optimization problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \tag{3.18}$$

Genetic algorithms operate on a population (set) of feasible points, evolving them over generations to find near-optimal solutions. These class of algorithms is based on three fundamental operators: selection, crossover, and mutation.

- **Selection**: the selection process determines which individuals (or solutions) from the current population will contribute to the next generation. Common selection methods include roulette wheel selection, tournament selection, and rank-based selection;
- **Crossover**: this operator combines different solutions to define new feasible points;
- **Mutation**: introduces small perturbations to new solutions to maintain diversity and explore the feasible space more thoroughly, avoiding premature convergence to the same point;

Because in literature many different strategy have been explored for the definition of the aformentioned phases, there are no limitations on the choices of selection, crossover and mutation mechanisms. For this reason, in Algorithm 2 we give a truly general outline of the scheme of the genetic strategy.

Algorithm 2 Standard Genetic Algorithm Scheme

1: Define an initial random set of N feasible solutions $\mathcal{S} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$
2: **while** A termination criterion is not satisfied **do**
3: $\quad f_{best} = \min\{f(\boldsymbol{x}), x \in \mathcal{S}\}$;
4: $\quad \boldsymbol{x}_{best} = \arg\min\{f(\boldsymbol{x}), x \in \mathcal{S}\}$;
5: $\quad$ Select $\mathcal{K} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_k\}$ $k \leq N$ solutions from $\mathcal{S}$
6: $\quad$ Generate $\tilde{\mathcal{K}} = \{\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_k\}$ using crossover and mutation operators from $\mathcal{K}$;
7: $\quad$ Update $\mathcal{S} = \mathcal{S} \setminus \mathcal{K} \cup \tilde{\mathcal{K}}$
8: **end while**
9: Return $f_{best}, \boldsymbol{x}_{best}$

Tree Memetic Optimization

Given the introduced framework, we now describe our method in detail. In the context of binary classification problems, for any $N \in \mathbb{N}$, let $\mathcal{D} := \{(\boldsymbol{x}_i, y_i),\ \boldsymbol{x}_i \in \mathbb{R}^p,\ y_i \in \{0,1\},\ i = 1,...,N\}$ be a finite set of data points with p features. To alleviate the combinatorial explosion of the feasible space, we search for good classification trees of a given maximum depth d, avoiding to introduce further complexity in the exploration of the space of feasible tree structures.

In the starting phase of any evolutionary algorithm, the population is initialized through the definition of an initial set of solutions. While random sampling in the feasible space is the prevailing method for this task, we argue that a completely random initialization might not be ideal for a population of decision trees due to their inherent structural dependency. Consequently, a random sampling initialization could lead the algorithm to explore "dead zones" within the search space, slowing the convergence to near-optimal solutions.

Given the well known trade-off between exploration and exploitation (Črepinšek et al., 2013), our methodology aims to balance both these effects by adopting a Random Forest (Breiman, 2001) as initial set of feasible classification trees. This latter choice is motivated by the principles of bagging (Breiman, 1996) and the random subspace method (Ho, 1998), integral components of the Random Forest training process that ensure sufficient heterogeneity within the ensemble while maintaining individual solution quality. Consequently, our algorithm adopts a Random Forest $RF = \{t_1, ..., t_k\}$ as a set of k decision trees for the initial population. Subsequent to this, each model is locally optimized with respect to the misclassification loss using TAO. In this way, the initial population becomes a set of local optima in the feasible space of classification trees of depth d for the given problem.

After this first step, a standard genetic algorithm requires to carry out the evolution by implementing the two main phases, selection and crossover. One of the common strategy for the selection is to use a fitness function which measures the "quality" of each solution (e.g., for minimization of scalar functions a trivial choice for the fitness is $-f(\boldsymbol{x})$). Since in this context we are interested in the induction of classification trees with low misclassification error, we employ the classification accuracy on the train set as fitness function. For the purposes of this discussion, from here on, we will denote by $f(t)$ the accuracy of the tree t on the train set.

The selection process can be carried out by favouring individuals with high fitness in the belief that they may contain good components for generating better solutions (Goldberg and Deb, 1991). The choice of the selection strategy greatly affects the convergence rate and, in general, an higher selection pressure results in higher convergence rates but it increases the chances that the population may converge prematurely to a local optimum. For this reason, we decided to not take into account the fitness of the individuals at this stage and to use a selection method which is

quite similar to the *Differential Evolution* algorithm (Storn and Price (1997)).

Our method iterates on each model and it samples another one from the population to obtain the couple of parents that will define a new solution. Let $i \in \{1, ..., k\}$ the index of the current tree, a second index j is sampled from a uniform distribution $\mathcal{U}_{\{j=1,...,k,\ j\neq i\}}$ over all the possible indexes except i. The solutions t_i and t_j are then encoded through a one-to-one mapping in a fixed-length list. These new representations will be used by the stochastic crossover operation, detailed in the next section, to define a new candidate $\hat{t}$.

Algorithm 3 TMO scheme

Input: Train data $\mathcal{D}$, Maximum depth d, Number of generation n_{gen}, Cross rate parameter $CR \in [0, 1]$
Get initial population $\mathcal{S} = \{t_1, ..., t_k\}$ from a fitted Random Forest on $\mathcal{D}$
$f_{best} = \max\{f(t),\ t \in \mathcal{S}\}$;
$t_{best} = \arg\max\{f(t),\ t \in \mathcal{S}\}$;
for $g = 1, .., n_{gen}$ **do**
    Get a bootstrap sample $B_{\mathcal{D}}^{(g)}$
    for $i = 1, ..., k$ **do**
        Sample $j \sim \mathcal{U}_{\{j=1,...,k,\ j\neq i\}}$
        for node $z = 1, ..., 2^d - 1$ **do**
            if $\mathcal{U}_{[0,1]} < CR$ **then**
                $\hat{t}^z = t_j^z$
            else
                $\hat{t}^z = t_i^z$
            end if
        end for
        Optimize $\hat{t}$ using $\mathcal{L}(\hat{t}\,; B_{\mathcal{D}}^{(g)})$
        if $f(\hat{t}) > f(t_i)$ **then**
            $t_i = \hat{t}$
        end if
        if $f(\hat{t}) > f_{best}$ **then**
            $f_{best} = f(\hat{t})$
            $t_{best} = \hat{t}$
        end if
    end for
end for

Unlike common genetic strategies, before asserting the quality of the new solution through the fitness function, our method exploits the memetic strategy to locally optimize the new candidate with respect to the misclassification loss using TAO. This latter local search algorithm, starting with a learned CT, employs a bottom-up alternating optimization technique on each level of the tree by defining new sub-

problems which aim to change the parameters of each node, to minimize the misclassification error of the points forwarded by the node itself.

This local optimization phase, indicated as $\mathcal{L}(\hat{t}\,; B_{\mathcal{D}})$, is made with respect to a bootstrap sample $B_{\mathcal{D}}$ of the training data. This allows to maintain a sufficient diversity, avoiding an over-intensive exploitation that would compromise the ability to explore the feasible space and to generate structures with good generalization performance. Finally, the quality of the improved candidate $\hat{t}$ is assessed by means of the accuracy on the train set which acts as a fitness function. The pseudo-code of our method is reported in Algorithm 3.

Tree encoding scheme

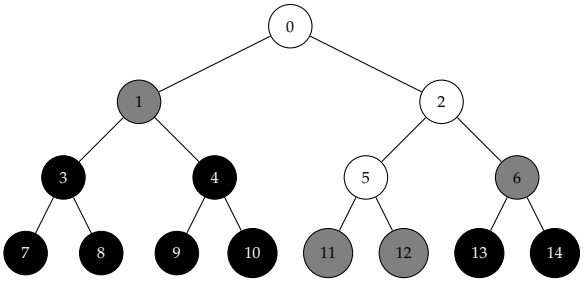


Figure 3.10: An unbalanced tree of depth 3

One of the key problems in the context of GAs applied to decision trees is the encoding mechanism used to get useful representations of feasible solutions. Different strategies can be found in the literature to address this problem but, in general, most common approaches used a tree-based encoding (Barros et al., 2011).

Our method maps each tree to a fixed-length list of tuples containing the parameters of every node in the structure. Since the class associated with each leaf is always chosen by majority from the labels of the points arriving on the leaf itself, these nodes do not directly contribute to the definitions of new solutions, thus the last level of the structure can be omitted from the representation without loss of information.

Similarly to the method in (Aitkenhead, 2008), given a tree t of depth d, the encoding mechanism considers a balanced structure of depth $d-1$ and it uses a tuple of two elements to represent each node. Since the information of a branch z is completely defined by the feature index f_z and the threshold value τ_z that the node uses to perform the split, we use the tuple (f_z, τ_z) to encode the parameters of this kind of nodes and the conventions $(-1,-1)$, (nil, nil) for leaves and missing nodes respectively.

The $2^d - 1$-length encoding list is finally built using all the representation tuples and it is sorted following the order that would be used by a *Breadth-First Search* (BFS) to visit the tree, giving priority to the left child.

Figure 3.10 shows an example of an unbalanced tree with respect to a complete structure of depth 3. Branches, leaves and missing nodes are on white, gray and black backgrounds respectively. The encoding of this structure, using our method, is the 7-length list defined as:

$$[(f_0, \tau_0), (-1, -1), (f_2, \tau_2), (nil, nil), (nil, nil), (f_5, \tau_5), (-1, -1)]$$

Generating new solutions

The core of every evolutionary algorithm is the method by which the definition of new solutions occurs (lines 9-15 of Algorithm 3). This phase is usually defined through the crossover operation. Our method employs a stochastic variant of this procedure, encouraging the exploration of the feasible space. After the selection step, the parents are encoded using the strategy described in the previous paragraph and the crossover operation is performed on each node till depth $d - 1$. More precisely, the nodes of the new solution are defined with respect to a cross-rate parameter CR. Given i, j as the indexes of the parents, each node z of the new solution $\hat{t}$ is defined as:

$$\hat{t}^z = \begin{cases} t_j^z & \text{if } \mathcal{U}_{[0,1]} \leq CR \\ t_i^z & \text{otherwise} \end{cases}$$

Thus, we set each node of the new solution to have the same parameters as the second parent node with probability CR and to have those of the first parent with probability $1 - CR$. However, it is able to produce radical modifications in the structure of the new solution with respect to the parents.

Figures 3.11, 3.12, 3.13, 3.14 show some examples of this. Let t_i be the reference tree in figure 3.10, a branch to leaf modification on node 5 is reported in figure 3.11. The effect of this operation is to prune all the sub-tree having the node 5 as root, setting this node as leaf with class obtained by majority with respect to the labels of the points arriving at that node.

On the other hand, the inverse operation is reported in figure 3.12. In this case the leaf node 1 becomes a branch with the same parameters of the other parent and its two leaves are created.

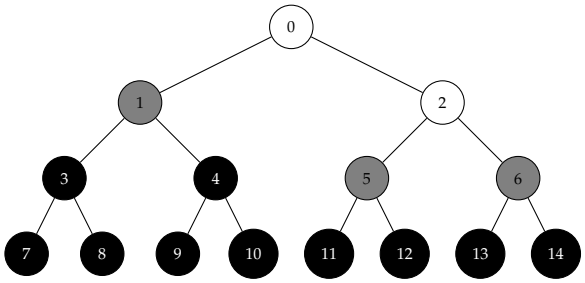


Figure 3.11: Modification *Branch/Leaf* on node 5 with respect to the structure in figure 3.10

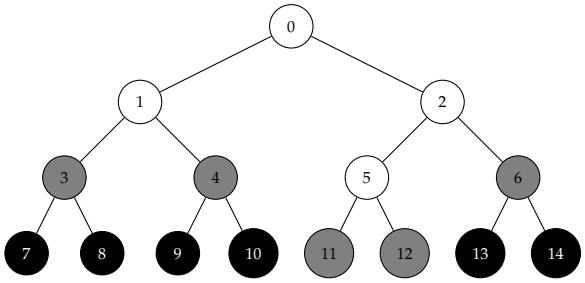


Figure 3.12: Modification *Leaf/Branch* on node 1 with respect to the structure in figure 3.10

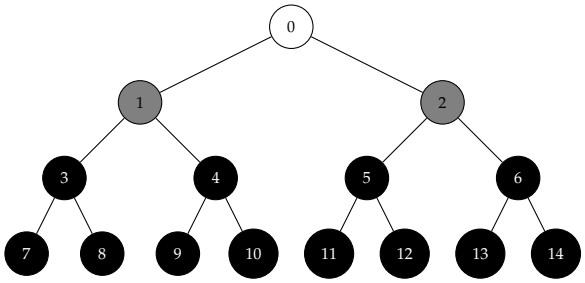


Figure 3.13: Modification *Branch/NIL* on node 5 with respect to the structure in figure 3.10

Because in the population there may exist very unbalanced structures, modifications like nil/branch and branch/nil are also possible. In those cases the structure of the new solution undergoes the most massive transformations.

In figure 3.13 a branch/nil alteration on node 5 is shown. Because the node has to be deleted from the structure, this implies that its father has to become a leaf. Thus, this kind of transformation acts as a total pruning of the sub-trees rooted at that node and at the brother (node 6), setting their father as leaf. Note that the same

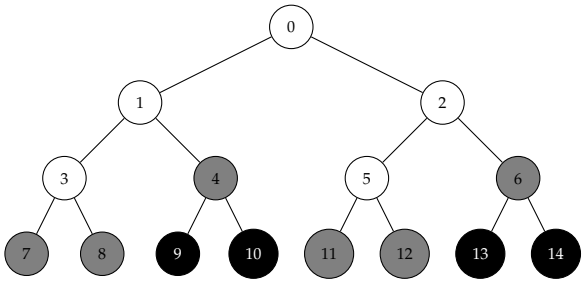


Figure 3.14: Modification *NIL*/*Branch* on node 3 with respect to the structure in figure 3.10

result can also be achieved with a branch/leaf change applied to the parent node.

Figure 3.14 shows a nil/branch conversion on node 3. This operation enlarges the structure of the model. Since a new branch has to be created, also the parent node (node 1), being a leaf, has to become a branch to maintain a correct hierarchy in the model. The parameters of the parent node are initialized as random while those of the current node are taken from the other tree participating in the crossover. Finally, the brother node (node 4) has also to be created and it is set to be a leaf with the class chosen by majority respect to the labels of the points arriving to the leaf itself.

Numerical experiments

To assess the performance of our method, we used a total of 14 benchmark datasets from LIBSVM [2] repositories .

The aim of the experiments is to compare our method with respect to three different baselines. The first one is the previously described CART algorithm for decision trees which is, even today, the most common baseline for classification trees induction. The second one is the alternating optimization TAO algorithm which takes as input the given CART structure and performs a local minimization with respect to the misclassification loss. Finally, we compare our method to the state-of-art MILP approach, OCT, proposed in (Bertsimas and Dunn, 2017).

Each dataset is split up in three parts (64 % training set, 16 % validation set, 20 % test set). We performed experiments considering trees with depth till 4 following the work in (Bertsimas and Dunn, 2017) that, given the exponential increase of the number of binary variables, limits the depth to this value.

First, we fitted the CART baseline with the standard implementation of the scikit-learn framework (Pedregosa et al., 2011), setting only the maximum depth parameter and leaving the rest as default. Then, we run a TAO implementation (python3) on the model obtained by scikit-learn to get the second baseline. Finally, we coded

[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Dataset	Dimension	CART	TAO	TMO	OCT
A2a	2265×119	81.02 ± 1.57	81.02 ± 1.57	80.93 ± 1.25	78.81 ± 3.56
A4a	4781×122	82.28 ± 0.77	82.28 ± 0.77	$\mathbf{82.34 \pm 0.74}$	82.28 ± 0.77
Biodeg	1055×41	74.41 ± 2.45	76.30 ± 4.52	$\mathbf{77.73 \pm 3.68}$	74.79 ± 2.29
Breast	194×33	70.77 ± 6.61	71.28 ± 3.77	71.28 ± 4.41	$\mathbf{74.36 \pm 3.24}$
Diabetes	768×8	72.08 ± 1.69	72.08 ± 1.69	$\mathbf{72.21 \pm 2.92}$	69.61 ± 4.07
Digits	3823×63	94.72 ± 0.39	94.72 ± 0.39	$\mathbf{94.93 \pm 0.46}$	93.59 ± 1.99
Heart	270×25	70.74 ± 3.78	70.74 ± 4.29	$\mathbf{71.48 \pm 5.19}$	67.41 ± 6.48
Ionosphere	351×34	89.86 ± 1.64	$\mathbf{90.7 \pm 1.13}$	$\mathbf{90.7 \pm 1.13}$	87.32 ± 5.19
Libras	360×90	95.0 ± 1.88	$\mathbf{95.28 \pm 1.88}$	95.0 ± 2.26	94.44 ± 1.76
Parkinsons	195×22	83.59 ± 3.48	$\mathbf{84.62 \pm 3.63}$	83.59 ± 4.47	82.05 ± 6.07
Phishing	11055×68	90.75 ± 0.61	90.75 ± 0.61	$\mathbf{91.43 \pm 0.78}$	90.15 ± 0.88
Spam	4601×57	85.41 ± 1.72	85.99 ± 1.39	$\mathbf{86.47 \pm 1.13}$	83.71 ± 2.39
Spectf	267×44	75.19 ± 5.69	$\mathbf{78.52 \pm 3.43}$	78.15 ± 3.59	77.41 ± 3.78
Sonar	208×60	69.52 ± 6.97	70.0 ± 5.75	$\mathbf{72.38 \pm 6.14}$	66.67 ± 2.13

Table 3.4: Accuracy and Standard Deviation for maximum depth $d = 2$, Gurobi time = 600s

the OCT formulation of the problem and we used gurobipy (Gurobi Optimization, LLC, 2022) to handle the optimization. The original formulation of OCT for axis-aligned splits includes a regularization term to take into account the total number of branch nodes, thus encouraging sparsity in the structure. For this reason, we tuned the relative hyperparameter α, sampling 3 values from a log-uniform distribution in $[10^{-3}, 1]$ and choosing the one with the best accuracy on the validation set. Moreover, to provide a strong initial upper bound on the optimal solution we followed the idea of the original OCT paper by performing a warm start procedure, initializing the optimization phase with the model obtained by the CART algorithm.

Our method, as described in the previous sections, employs a random forest to initialize the population for the evolutionary procedure. Also in this case we used the standard implementation in scikit-learn to fit the ensemble, setting only the maximum depth and a number of 100 classification trees as the size of the population.

We decided not to validate the cross-rate hyperparameter as we empirically observed that setting $CR = 0.75$ is a good trade-off for the exploration and it allows to obtain competitive solutions. Moreover, with reference to the pseudo-code reported in Algorithm 3 we set the number of generations of our algorithm n_{gen} equal to 5. Finally, for each method we set the maximum running time limit to 600s.

All the experiments are carried out on a server with an Intel ®Xeon ®Gold 6330N

Dataset	Dimension	CART	TAO	TMO	OCT
A2A	2265×119	80.79 ± 1.3	80.75 ± 1.32	$\mathbf{80.97 \pm 1.54}$	78.90 ± 3.60
A4A	4781×122	81.9 ± 1.18	82.22 ± 0.6	$\mathbf{82.68 \pm 0.95}$	80.08 ± 3.44
Biodeg	1055×41	78.1 ± 1.06	78.1 ± 2.77	$\mathbf{79.91 \pm 1.81}$	74.41 ± 5.59
Breast	194×33	69.74 ± 5.94	$\mathbf{74.36 \pm 3.63}$	67.69 ± 3.48	72.31 ± 5.71
Diabetes	768×8	69.87 ± 2.45	71.43 ± 2.63	$\mathbf{74.42 \pm 1.2}$	69.35 ± 3.97
Digits	3823×63	96.44 ± 0.71	96.81 ± 0.87	$\mathbf{96.89 \pm 0.62}$	92.55 ± 2.92
Heart	270×25	78.15 ± 0.74	78.52 ± 0.91	$\mathbf{81.85 \pm 2.16}$	79.45 ± 1.23
Ionosphere	351×34	$\mathbf{90.99 \pm 1.13}$	$\mathbf{90.99 \pm 0.69}$	87.32 ± 1.99	88.45 ± 4.22
Libras	360×90	93.89 ± 1.11	94.72 ± 1.36	$\mathbf{95.28 \pm 2.72}$	94.44 ± 1.76
Parkinsons	195×22	81.54 ± 1.92	86.15 ± 2.61	$\mathbf{90.77 \pm 3.08}$	86.67 ± 6.36
Phishing	11055×68	90.59 ± 0.56	91.23 ± 0.83	$\mathbf{91.92 \pm 0.45}$	89.43 ± 0.81
Spam	4601×57	87.84 ± 1.75	88.47 ± 1.5	$\mathbf{89.73 \pm 0.9}$	88.56 ± 1.23
Spectf	267×44	72.59 ± 9.18	72.22 ± 8.61	71.85 ± 8.23	$\mathbf{78.15 \pm 4.6}$
Sonar	208×60	72.86 ± 4.9	$\mathbf{75.71 \pm 5.91}$	69.52 ± 3.81	70.95 ± 3.50

Table 3.5: Accuracy and Standard Deviation for maximum depth $d = 3$, Gurobi time = 600s

CPU with 28 cores and 56 threads @ 2.20GHz. The total available memory is 128GB. For each dataset we report the size, the mean out-of-sample accuracy and the standard deviation of each method on 5 different seeds.

Discussion

In Tables 3.4, 3.5, 3.6 we compare the performance of our approach in case of trees with maximum depth up to 4. The comparison aims to prove the effectiveness of our method with respect to different depths, to capture also information about the performance trend with respect to the transparency of the model. Moreover, the main contribution is related to the applicability of our method to real world datasets with thousand of samples in opposition to the MILP approach that often fails to obtain a better solution than the one provided by the CART warm start.

Both for depth 2 and 3, results highlights the effectiveness of our memetic evolutionary method which outperforms other approaches in 9 out of 14 datasets (64%). More generally, it is evident that TMO is often able to induce structures with better generalization capabilities even for datasets with thousands of samples. This result is particularly evident for larger datasets (Phishing, A4A, Spam, Digits) where MILP models suffers the fact that the number of binary variables scales exponen-

Dataset	Dimension	CART	TAO	TMO	OCT
A2a	2265 × 119	80.66 ± 1.22	81.1 ± 1.44	**81.20 ± 1.85**	80.57 ± 2.84
A4a	4781 × 122	82.22 ± 1.41	82.11 ± 1.27	**82.13 ± 0.87**	80.44 ± 2.89
Biodeg	1055 × 41	**81.23 ± 2.26**	80.85 ± 2.57	80.85 ± 2.55	80.09 ± 1.44
Breast	194 × 33	67.18 ± 4.41	68.72 ± 5.23	72.31 ± 5.23	**77.44 ± 1.03**
Diabetes	768 × 8	70.78 ± 2.36	70.65 ± 3.49	**72.21 ± 2.07**	69.61 ± 4.01
Digits	3823 × 63	97.15 ± 0.58	97.23 ± 0.48	**97.39 ± 0.45**	97.31 ± 0.28
Heart	270 × 25	75.56 ± 4.44	75.93 ± 4.22	**78.89 ± 3.63**	73.70 ± 3.78
Ionosphere	351 × 34	88.45 ± 3.26	**89.01 ± 2.25**	86.76 ± 2.61	84.51 ± 3.88
Libras	360 × 90	**95.0 ± 1.88**	**95.0 ± 1.11**	93.61 ± 3.36	94.72 ± 2.04
Parkinsons	195 × 22	83.59 ± 3.48	84.1 ± 2.99	85.64 ± 4.76	**86.67 ± 5.94**
Phishing	11055 × 68	91.67 ± 0.32	91.67 ± 0.32	**92.15 ± 0.76**	-
Spam	4601 × 57	89.75 ± 1.17	90.16 ± 1.24	**90.47 ± 0.68**	89.88 ± 1.57
Spectf	267 × 44	72.59 ± 6.87	73.33 ± 7.46	73.33 ± 6.04	**78.52 ± 5.44**
Sonar	208 × 60	**70.0 ± 4.9**	67.62 ± 3.56	69.52 ± 3.81	68.10 ± 8.19

Table 3.6: Accuracy and Standard Deviation for maximum depth $d = 4$, Gurobi time = 600s

tially with respect to the depth of the tree and linearly with respect to the number of samples in the datasets. Moreover, OCT, although it exploits the warm start procedure, can rarely outperform CART. This drawback is mainly related to the model becoming intractable, making hard for the MILP solver to find any better feasible solution than the one initially provided by CART.

An example of this fact is visible in the results for depth 4 (Table 3.6) for the phishing dataset. The OCT approach exceeds the availbale RAM (128GB) as the branch and bound tree allocated by the Gurobi solver requires too much memory during the validation, causing the operating system to kill the process. This is clearly a consequence of the $\mathcal{NP}$-completeness of the problem that makes exact models unusable for real world datasets even with small trees.

These latest results show also that OCT can sometimes provide structures with better out-of-sample accuracy for deeper trees, exploiting the regularization term that may discover pruned models, encouraging generalization. However, also at this depth, our algorithm is the most competitive among the four proposed, achieving the best performance on half of the datasets.

Finally, although theoretically able to provide the certified global optimum, in practice results show that this optimal solution is almost never reached (or, at least, certified) because of the difficulties in closing the optimality gap. For this reason,

our method, with no claim to provide the global optimum, has proven to be a good trade-off between the applicability of over-exploited greedy approaches and the need of exploration of the feasible space to induce near-optimal solutions.

Computational complexity

Unlike MILP models, the complexity of our method does not depend exponentially on the depth of the tree. In fact, at each iteration of the algorithm, the method performs the selection/crossover phase which can be made in $O(2^d)$, then an instance of the local optimizer TAO is performed and its cost is comparable to running CART to grow a tree of the same depth (Carreira-Perpinán and Tavallali, 2018), thus getting a complexity of $O(npd)$. These two routines are repeated for each tree in the population getting a total cost of $O(k2^d + knpd)$. Note that in case of shallow trees and large datasets, complexity mainly depends on the term $O(knpd)$ saying that a generation of TMO is comparable to running CART a number of times equal to the size of the population.

3.4 Concluding Remarks

In this chapter we reviewed the most relevant methods for the induction of CTs. Starting with the most common greedy algorithm, CART, we discussed the recent state-of-art exact MIP formulations for the modelling of the learning problem. We then introduce a general framework for loss-optimal classification trees, showing how different losses can be handled through a proper definition of the slack variables. Following further this path, we provided a new formulation which employs a piece-wise linear surrogate of the log-loss for the induction of logistic multivariate classification trees. We showed how logistic splits with a standard LASSO regularization can be used to construct sparser and more interpretable trees with better generalization performances, with a reasonable computational cost. The trade-off between out-of-sample accuracy, interpretability and running time attained by our proposed approach thus appears to be optimal among the models considered in our numerical experiments.

On the other hand, given the computational complexity of MIP formulations and their observed difficulties in handling large datasets and deep trees, we reviewed genetic strategies which, without any claim to certify optimality, are empirically able to learn classification trees with competitive performance. In this regard, we extend the standard evolutionary scheme by exploiting the memetic approach to perform local searches during the evolution, increasing the effectiveness of the algorithm in inducing near optimal solutions.

Chapter 4

Optimization for Risk Scores

In this Chapter we discuss the optimization problem of learning a particular class of linear models called *Risk Scores* (RSs). RSs (Ustun and Rudin, 2019) are widely used linear classification models that enable users to assess the risk (or equivalently, in statistical terms, the log-odds) of a positive outcome by means of a simple summation of few small integer terms. More precisely, RSs can be described as very (Super) Sparse Linear Integer Models (SLIMs) (Ustun and Rudin, 2016) that are designed for risk assessment (RiskSLIM) rather than for decision. For this reason, such models are optimized using the logistic loss function as opposed to the 0-1 loss used for SLIM models.

Thanks to the extreme glass-box nature of risk scores, users can estimate the impact, on the predicted result, of changes to one or even several input variables; informed decisions can then be made, even rejecting the output of the model if it is found to be unreasonable by experts. In the health care context especially, the use of this kind of models enables clinicians and researchers to understand the factors influencing risk assessment and enhances the reliability of the predictions. Additionally, risk scores provide valuable insights into the decision-making process, allowing for improved clinical outcomes and patient care. Famous risk scores models in healthcare include Wells' criteria for Deep Vein Trombosis (DVT) (Wells et al., 1995, 1997, 2003, 2006) and for pulmonary embolism (Wells et al., 2001; Wolf et al., 2004), the Glasgow Coma Scale (GCS) (Teasdale and Jennett, 1974), the Sepsis-related Organ Failure Assessment (SOFA) score (Vincent et al., 1996) and the $CHADS_2$ risk score (Figure 4.1) to assess stroke risk (Gage et al., 2001); more examples can be seen at `www.mdcalc.com`.

The broad acceptance and utilization of risk scores in these delicate domains can be attributed to their ease of employment and comprehension from the user side; as opposed to many other classification models, risk scores are specifically designed to be simple and transparent, being accessible to individuals with diverse levels of technical expertise.

Variable	Score
Congestive Heart Failure (CHF) history	+ 1 point
Hypertension history	+ 1 point
Age ≥ 75 years	+ 1 point
Diabetes mellitus history	+ 1 point
Stroke or TIA symptoms previously	+ 2 points

Total Score:	0	1	2	3	4	5	6
Risk:	1.9%	2.8%	4.0%	5.9%	8.5%	12.5%	18.2%

Figure 4.1: $CHADS_2$ risk score to assess the stroke risk for patients with atrial fibrillation.

Although early studies on risk factor models can be found as early as the first half of the 1900s (Burgess, 1928), in practice even today the learning process in most cases involves the use of ad-hoc strategies. In fact, such models are often hand-crafted by experts of the specific field of use, deploying heuristics to capture relations between the features and the outcome (McGinley and Pearse, 2012). Approaches based on logistic regression can also be found in the literature; however, these methods usually consist of a pipeline that involves an initial stage of feature selection followed by the fitting of the logistic model and finally a rounding procedure to obtain integer scores (Antman et al., 2000).

All in all, the aforementioned procedures are blind greedy strategies, that do not consider an exact formulation of the problem. The natural way to solve this issue would be to train the entire model in a single step: the result could be considered an *Optimal* Risk Score (ORS) for the given dataset. Constructing a risk score model intrinsically requires a series of discrete decisions, such as the selection of relevant features to be considered and the scores to be assigned to each one; thus, the definition of the learning problem as an optimization one naturally leads to a Mixed Integer Optimization (MIO) program to handle.

Indeed, following this path, the seminal work for learning ORSs (Ustun and Rudin, 2019) exploits a Lattice Cutting Plane Algorithm (LCPA) to handle the non-linear log-loss in a MILP context but is limited to the case of binary features.

Within this scenario, in this chapter we give a new formulation for learning ORSs on binary features, which leverages on the same piece-wise linear underestimator, previously discussed for *Optimal Logistic Classification Trees* (chapter 3), to efficiently handle the logistic loss in MILP. Moreover we extend this kind of models by introducing Generalized Risk Scores (GRSs). The new class of models extends standard risk scores by generating boolean decisions from each feature to assign the score. By leveraging this mechanism, the GRS model enhances its predictive performance and flexibility, with no loss of interpretability and it is able to handle each type of features.

4.1 Learning Optimal Risk Scores

Let us consider a training dataset of N observations $\mathcal{D} = \{(\boldsymbol{x}^i, y^i),\ \boldsymbol{x}^i \in \mathbb{R}^p,\ y^i \in \{-1, 1\}\ \forall i \in \mathcal{I}\}$, being $\mathcal{I} = \{1, 2, ..., N\}$ the index set. In the binary classification context, we assume $y \mid \boldsymbol{x}$ to be a target random variable with Bernoulli distribution $B(p)$. A logistic regression model $R(\boldsymbol{x}; \boldsymbol{s})$ is a linear classifier that estimates the expected value $p = \mathbb{P}(y = 1 \mid \boldsymbol{x})$ through the logistic activation $\sigma(R(\boldsymbol{x}; \boldsymbol{s}))$ i.e., R is a linear estimator of the log-odds or *logit* (Hastie et al., 2009):

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = R(\boldsymbol{x}; \boldsymbol{s}) = \boldsymbol{s}^T\boldsymbol{x} + \beta.$$

We say that $R(\boldsymbol{x}; \boldsymbol{s})$ is a risk score model (cfr. Ustun and Rudin (2019)) for $\mathcal{D}$ if the following holds:

$$s_j \in A \subset \mathbb{Z} \qquad \forall j = 1, ..., p, \tag{4.1a}$$

$$\|\boldsymbol{s}\|_0 = T, \tag{4.1b}$$

where $\|\cdot\|_0$ denotes the ℓ_0 pseudo-norm of a vector, i.e., the number of nonzero components and, for interpretability reasons, we constraint this number to be not greater than a small threshold which frequently is set to seven (see (Miller, 1956) for psychological details on this choice).

A risk score model is thus a very sparse logistic regression classifier with (small) integer weights chosen from a set A which is user defined.

Constraints (4.1) greatly enhance interpretability. Weights sparsity simplifies the understanding and interpretation of the model, as it makes it easier to discern the features that have a significant impact on predictions (Tibshirani, 1996; Fonti and Belitser, 2017). On the other hand, integer weights are in general much easier to manage for humans even without the help of a calculator.

Given this background, we are now able to introduce each block of our formulation to induce ORSs which is largely based on the work in (Ustun and Rudin, 2019).

We are given the set $A \subset \mathbb{Z}$ of feasible scores and a number of rules T which have to be created. To satisfy the constraint (4.1b) the model must be able to choose only T features among p and we handle this combinatorial selection by introducing the indicator variables:

$$a_j = \begin{cases} 1 & \text{if feature } j \text{ is selected} \\ 0 & \text{otherwise.} \end{cases} \qquad \forall j = 1, ..., p$$

And imposing the following equation to handle the ℓ_0 pseudo-norm constraint:

$$\sum_{j=1}^{p} a_j = T \tag{4.2}$$

Each coefficient s_j is then linked to the variables a_j through the common big-M strategy:

$$s_j \leq M a_j \tag{4.3a}$$
$$s_j \geq -M a_j \tag{4.3b}$$

where the value for the big-M constant can be chosen as $M = \max\{|s|,\ \forall s \in A\}$ which is in general very small (≤ 5). Constraints (4.2) and (4.3) are sufficient to model all the required logic of a RS, leaving to define only the form of the loss function used.

In a binary classification context the log-loss, which is related to the negative log likelihood for logistic regression (see, e.g, (Hastie et al., 2009)), is defined as

$$\ell(\boldsymbol{w}^T \boldsymbol{x}^i + \beta; y^i) = f(y^i(\boldsymbol{w}^T \boldsymbol{x}^i + \beta)) = \log(1 + \exp(-y^i(\boldsymbol{w}^T \boldsymbol{x}^i + \beta))).$$

In addition to having excellent properties in terms of out-of-sample accuracy, this function provides insights on features importance by manipulation of the model weights (Hastie et al., 2009) and it is also known for inducing well calibrated models.

However, the main issue with this loss function is the fact that it cannot exactly be modeled in a MILP context because of its nonlinear nature. To address this issue, instead of using cutting plane strategies as in (Ustun and Rudin, 2019), we follow the method proposed by Sato et al. in (Sato et al., 2016) already employed in Chapter 3 for the induction of *Optimal Logistic Classification Trees*. Thus, we approximate the log-loss through the maximum of a set of its tangent lines built at some points v_k, i.e.,

$$f(v) \approx \max\{f(v_k) + f'(v_k)(v - v_k) \mid k = 0, \ldots, K\}.$$

We recall that the sets of tangent points proposed by Sato et al. are:

$$V_0 = \{0, \pm\infty\}, \qquad V_1 = V_0 \cup \{\pm 1.9\}, \qquad V_2 = V_1 \cup \{\pm 0.89, \pm 3.55\},$$
$$V_3 = V_2 \cup \{\pm 0.44, \pm 1.37, \pm 2.63, \pm 5.16\}.$$

Now, using suitable slack variables $\xi_i \in \mathbb{R}^+$, we are able to approximate the log-loss to be managed in a MILP context as follows

$$\xi_i \geq f(v_k) + f'(v_k)(y^i(\boldsymbol{s}^T \boldsymbol{x}^i + \beta) - v_k) \qquad \forall i,\ \forall k \tag{4.4}$$

Clearly, the larger the set V, the more accurate is the approximation of the logistic loss but also the number of linear constraints to handle the definition of the slack variables and, thus, the complexity of the MILP model.

In the numerical experiments we will show how different sets lead to different results both for the quality of the approximation and for the predictive accuracy of the estimator. Finally, the resulting MILP model for learning ORS is the following:

$$\min_{\boldsymbol{s},\boldsymbol{\xi},\boldsymbol{a},\beta} \sum_{i\in\mathcal{I}} \xi_i \tag{4.5a}$$

$$\text{s.t. } (4.2), (4.3), (4.4) \tag{4.5b}$$

$$\xi_i \in \mathbb{R}^+ \qquad \forall i \in \mathcal{I}, \tag{4.5c}$$

$$\beta \in \mathbb{R} \tag{4.5d}$$

$$a_j \in \{0,1\} \qquad \forall j = 1,...,p \tag{4.5e}$$

$$s_j \in A \qquad \forall j = 1,...,p \tag{4.5f}$$

This model allows to potentially learn the optimal risk score given a number T of features to be considered. However, there is no assumption on the type of these variables and the highest levels of interpretability for a risk score model is clearly obtained with binary features (Ustun and Rudin, 2019). Indeed, in this special case, the final score for a given sample is established by very simple and clear rules as shown in the model in Figure 4.1.

4.2 Generalized Risk Scores

The assumption that data contains only binary features is very limiting and, when attained in practice, it is often the result of a heuristic preprocessing phase of discretization (as done in Ustun and Rudin (2019)) that may lead to poor out-of-sample performance.

Arguably, the main advantage of binary features is that end users can readily assess whether the score has to be assigned or not. However, the ease of determining the binary decision to assign a score is what really matters, rather than data being of binary nature.

With the above observation in mind, we are motivated to define risk scores in a more general way. In particular, we want numerical features to be handled directly during the learning phase, preserving the useful binary conditions structure.

The extension of risk score classifiers is carried out by means of possibly nonlinear, general functions $f_j(x_j)$, $j = 1,\dots,p$, such that the *logit* of the positive outcome is now modeled as:

$$\text{logit}(p) = \sum_{j=1}^{p} f_j(x_j) + \beta = R(\boldsymbol{x}; \boldsymbol{f}, \beta) \tag{4.6}$$

In this way, the contribution of each feature to the final score is easily understood observing the map f_j.

Equation (4.6) actually represents the standard form of a Generalized Additive Model (GAM) (Hastie and Tibshirani, 1990).

GAMs allow for the incorporation of non-linear functions of individual predictors, enabling them to capture intricate patterns in the data. One of the key strengths of these models lies in their interpretability. By decomposing the relationship between the response variable and predictors into a series of non linear functions, GAMs provide a clear visual representation of the impact of each predictor, allowing researchers and practitioners to gain insights into the data without sacrificing model transparency. Consequently, GAMs strike a delicate balance between complexity and comprehensibility and for this reason are very common choices in the health-care context (Hastie and Tibshirani, 1995; Caruana et al., 2015).

We are now able to introduce the Generalized Risk Score (GRS) model as a particular instance of a GAM, such that each function f_j has the form

$$f_j(x_j) = s_j r_j(x_j) \tag{4.7}$$

with $r_j : \mathbb{R} \to \{0, 1\}$ and constraints (4.1a) and (4.1b) hold. We can thus denote GRS models as

$$R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta) = \sum_{j=1}^{p} s_j r_j(x_j) + \beta$$

Functions r_j can in principle be any arbitrary mapping to $\{0, 1\}$; the only underlying requirement is that they need to be easily interpretable. In other words, a GRS is a very sparse GAM where each component is a transparent step function with a small integer response. For this class of models, we are able to prove the following equivalence result.

Theorem 4.1 (Editability of Generalized Risk Scores)**.** *Let $R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta)$ be any GRS model. Let $k \in \{1, \dots, p\}$, and consider vectors $\hat{\boldsymbol{s}}$, $\hat{\boldsymbol{r}}$ and the value $\hat{\beta}$ defined as follows:*

$$\hat{s}_j = \begin{cases} -s_j & \text{if } j = k \\ s_j & \text{otherwise,} \end{cases} \quad \hat{r}_j(x_j) = \begin{cases} 1 - r_j(x_j) & \text{if } j = k \\ r_j(x_j) & \text{otherwise,} \end{cases} \quad \hat{\beta} = \beta + s_k.$$

Then, $R_G(\boldsymbol{x}; \hat{\boldsymbol{s}}, \hat{\boldsymbol{r}}, \hat{\beta}) = R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta)$ for all $\boldsymbol{x} \in \mathbb{R}^p$, i.e., model $R_G(\boldsymbol{x}; \hat{\boldsymbol{s}}, \hat{\boldsymbol{r}}, \hat{\beta})$ is equivalent to $R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta)$.

Proof. For any arbitrary point $\boldsymbol{x} \in \mathbb{R}^p$ we have

$$R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta) = \boldsymbol{s}^T \boldsymbol{r}(\boldsymbol{x}) + \beta$$

Now, let us define the functions $\bar{r}_j(x_j) = \mathbb{1}\{r_j(x_j) = 0\} = 1 - r_j(x_j)$. Then, for all $k = 1, ..., p$ we have

$$\begin{aligned}\sum_j s_j r_j(x_j) + \beta &= \sum_{j \neq k} s_j r_j(x_j) + s_k(1 - \bar{r}_k(x_k)) + \beta \\ &= \sum_{j \neq k} s_j r_j(x_j) - s_k \bar{r}_k(x_k) + \beta + s_k \\ &= \sum_j \hat{s}_j \hat{r}_j(x_j) + \hat{\beta},\end{aligned}$$

defining $\hat{\beta} = \beta + s_k$,

$$\hat{s}_j = \begin{cases} -s_j & \text{if } j = k, \\ s_j & \text{otherwise} \end{cases}$$

and

$$\hat{r}_j(x_j) = \begin{cases} \bar{r}_j(x_j) & \text{if } j = k, \\ r_j(x_j) & \text{otherwise,} \end{cases}$$

we got the thesis. □

The above proposition states that an end user can edit any decision of a generalized risk score by inverting the logic of that binary function, the sign of the associated score and updating the bias to obtain an equivalent model. This property can be particularly useful for an expert that can be able to modify the final model increasing the versality and the ease of use.

Moreover, from this latter Theorem we are able to deduce this nice Corollary.

Corollary 4.1. *Let $R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta)$ be any GRS model. Then, there exist an equivalent model $R_G(\boldsymbol{x}; \hat{\boldsymbol{s}}, \hat{\boldsymbol{r}}, \hat{\beta})$ such that $\hat{s}_j \geq 0$ for all $j = 1, \ldots, p$.*

Proof. Following the same technique of the proof of Theorem 4.1 we have:

$$\begin{aligned}R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta) &= \boldsymbol{s}^T \boldsymbol{r}(\boldsymbol{x}) + \beta \\ &= \sum_{j=1}^{p} s_j r_j(x_j) + \beta \\ &= \sum_{j: s_j \geq 0} s_j r_j(x_j) + \sum_{j: s_j < 0} s_j r_j(x_j) + \beta\end{aligned}$$

Now, let us define the functions $\bar{r}_j(x_j) = \mathbb{1}\{r_j(x_j) = 0\} = 1 - r_j(x_j)$. Setting $\hat{\beta} = \beta + \sum_{j: s_j < 0} s_j$ and

$$\hat{s}_j = \begin{cases} s_j & \text{if } s_j \geq 0, \\ -s_j & \text{otherwise} \end{cases}$$

$$\hat{r}_j(x_j) = \begin{cases} r_j(x_j) & \text{if } s_j \geq 0, \\ \bar{r}_j(x_j) & \text{otherwise,} \end{cases}$$

we get

$$\begin{aligned} R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta) &= \sum_{j:s_j \geq 0} s_j r_j(x_j) + \sum_{j:s_j < 0} s_j(1 - \bar{r}_j(x_j)) + \beta \\ &= \sum_{j:s_j \geq 0} s_j r_j(x_j) + \sum_{j:s_j < 0} -s_j \bar{r}_j(x_j) + \beta + \sum_{j:s_j < 0} s_j \\ &= \sum_{j=1}^{p} \hat{s}_j \hat{r}_j(x_j) + \hat{\beta} \\ &= R_G(\boldsymbol{x}; \hat{\boldsymbol{s}}, \hat{\boldsymbol{r}}, \hat{\beta}). \end{aligned}$$

Taking into account that $\hat{s}_j \geq 0$ for all j, we got the thesis. □

In addition to providing a nice insight into this type of models, the result tells us that we could consider only positive scores when modelling the learning problem, reducing the complexity of the formulation without loss of expressive power.

4.3 The Optimal Generalized Risk Score Model

In this Section we formalize our novel formulation for the OGRS model. In particular, we first introduce the form of r_j that we use in our approach then we describe how to model OGRS through Mixed Integer Linear Programming.

Choosing the family of the r_j functions

In Section 4.2 we gave the standard form of a GRS, specifing how the functions r_j should be chosen. From here on, we will use the term "rule" to indicate each decision function r_j. Among the many possible choices for such rules, we specialized on r_j that are defined through a single clause that tests if the value x_j is greater or equal than a threshold b_j, that is, r_j can be written as:

$$r_j(x_j) = \mathbb{1}\{x_j \geq b_j\} = \max\{0, sign(x_j - b_j)\}$$

Using rules of a single clause provides several advantages. Since single clauses involve the usage of one condition, they are easy to comprehend, to test their value and thus to obtain the final decision. Moreover, they often enhance efficiency by demanding fewer computational resources for the learning. Additionally, the utilization of single clauses offers flexibility in modifying or extending the logic, allowing a simplified editing.

Moreover, from theorem 4.1 we are able to derive another corollary on the verse of each rule.

Corollary 4.2. *Let $R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta)$ be any GRS model. Then, there exist an equivalent model $R_G(\boldsymbol{x}; \hat{\boldsymbol{s}}, \hat{\boldsymbol{r}}, \hat{\beta})$ such that $\hat{r}_j(x_j) = \mathbb{1}\{x_j \geq b_j\}$ or $\hat{r}_j(x_j) = \mathbb{1}\{x_j < b_j\}$ for all $j = 1, \ldots, p$, i.e, all the rules in the model have the same logic.*

Proof. Without loss of generality we prove the thesis for rules $r_j(x_j) = \mathbb{1}\{x_j \geq b_j\}$.

Following the same technique of the proof of theorem 4.1 we have:

$$R_G(\boldsymbol{x}; \boldsymbol{s}, \boldsymbol{r}, \beta) = \boldsymbol{s}^T \boldsymbol{r}(\boldsymbol{x}) + \beta$$

Now, let us define the functions $\bar{r}_j(x_j) = 1 - r_j(x_j)$ and the sets $\bar{\mathcal{J}} = \{j = 1, \ldots, p \mid r_j(x_j) = \mathbb{1}\{x_j < b_j\}\}$ and $\mathcal{J} = \{j = 1, \ldots, p \mid r_j(x_j) = \mathbb{1}\{x_j \geq b_j\}\}$. Then we have

$$\begin{aligned}
\sum_j s_j r_j(x_j) + \beta &= \sum_{j \in \mathcal{J}} s_j r_j(x_j) + \sum_{j \in \bar{\mathcal{J}}} s_j r_j(x_j) + \beta \\
&= \sum_{j \in \mathcal{J}} s_j r_j(x_j) + \sum_{j \in \bar{\mathcal{J}}} s_j (1 - \bar{r}_j(x_j)) + \beta \\
&= \sum_j \hat{s}_j \hat{r}_j(x_j) + \hat{\beta},
\end{aligned}$$

setting $\hat{\beta} = \beta + \sum_{j \in \bar{\mathcal{J}}} s_j$ and

$$\hat{s}_j = \begin{cases} s_j & \text{if } j \in \mathcal{J}, \\ -s_j & \text{otherwise} \end{cases}$$

$$\hat{r}_j(x_j) = \begin{cases} r_j(x_j) & \text{if } j \in \mathcal{J}, \\ \bar{r}_j(x_j) & \text{otherwise,} \end{cases}$$

we got the thesis. □

Formulating OGRS as a MIO problem

We are given the set $A \subset \mathbb{Z}$ of feasible scores and a number of rules T which have to be created. Without loss of generality, we assume that each feature has been normalized in the range $[0, 1]$. We want the model to select, among all the p features, only T of these to create clauses in the form $x_j \geq b_j$ (note that corollary 4.2 guarantees that this assumption is not limiting) associated to a score $s_t \in A$. Each point receives a total score $\bar{s}_i \in \mathbb{Z}$ which is the summation of the scores obtained at each satisfied rule.

To handle the decision of each rule we introduce the binary variables:

$$a_{j,t} = \begin{cases} 1 & \text{if feature } j \text{ is selected by rule } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

Using these variables we can force the formulation to comply with the structure of the model.

The first constraint we add is related to the selection of the T features that will be considered as rules, thus, $\forall t \in \mathcal{T}$ we impose that each rule has to select exactly one feature through the following:

$$\sum_{j=1}^{p} a_{j,t} = 1 \qquad \forall t \tag{4.8a}$$

Moreover we want to avoid feature repetitions in the rules, that is, the model can select each feature at most once. This can be enforced imposing that for each feature j, at most one variable $a_{j,t}$ is active i.e.:

$$\sum_{t=1}^{T} a_{j,t} \leq 1 \qquad \forall j \tag{4.9a}$$

The number of possible ways for selecting T features from p is equal to $\binom{p}{T}$. Since, for now, a permutation in the order of the rules leads to different configurations associated with the same risk score, we prune the space of feasible solutions for the variables a_{jt} by a factor of $T!$, imposing that rules have to be ordered with respect to the features considered. Thus, if rule t splits on feature j, then each rule $s < t$ cannot split on a feature $k > j$. This fact can be imposed through the following inequality.

$$\sum_{k=j+1}^{p} \sum_{s=1}^{t-1} a_{k,s} \leq (1 - a_{j,t})\, T \qquad \forall j,\ \forall t \tag{4.10a}$$

We also need to know whether a point $\boldsymbol{x}^i$ satisfies a rule $t \in \mathcal{T}$ in order to assign the related score. For this purpose we introduce the binary variables:

$$z_{i,t} = \begin{cases} 1 & \text{if } \boldsymbol{x}^i \text{ satisfies rule } \ t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

The GRS model must also be able to select, for each rule t, the threshold b_t that imposes the satisfaction of the clause.

Hence, if rule t splits on feature j then, $\forall\ i \ \mid \ z_{i,t} = 1$ i.e, for each point $\boldsymbol{x_i}$ that satisfies rule t, it must hold $x_i^j \geq b_t$. Otherwise the point $\boldsymbol{x}^i$ does not satisfy the rule and $z_{it} = 0$.

This logic can be inferred through the following constraints:

$$\boldsymbol{a}_t^T \boldsymbol{x}^i \geq b_t - 1 + z_{i,t} \qquad \forall t,\ \forall i \tag{4.11a}$$
$$\boldsymbol{a}_t^T \boldsymbol{x}^i \leq b_t - \epsilon + z_{i,t} \qquad \forall t,\ \forall i \tag{4.11b}$$

Constraint (4.11a) force each point $\boldsymbol{x}^i$ that fulfil the rule t to impose $z_{i,t} = 1$ and to satisfy the clause $\boldsymbol{a}_t^T \boldsymbol{x}^i \geq b_t$. On the other hand, constraint (4.11b) force the point to respect the opposite logic, $\boldsymbol{a}_t^T \boldsymbol{x}^i < b_t$, in case $z_{i,t} = 0$. Please note that the small ϵ constant in constraint (4.11b) is the common method to model strict inequalities in commercial MIP solvers that are not able to handle this type of constraints directly.

Another requirement that we want to be satisfied by the model is related to the definition of discriminative rules, i.e., to derive clauses that are satisfied at least by one sample (constraint (4.12a)) but not by all (constraint(4.12b)). Therefore we impose:

$$\sum_i z_{i,t} \geq 1 \qquad \forall t \tag{4.12a}$$
$$\sum_i z_{i,t} \leq N - 1 \qquad \forall t \tag{4.12b}$$

Then, we introduce the score obtained by each point $\boldsymbol{x}^i$ on the rule $t \in \mathcal{T}$ as the integer variable $\hat{s}_{i,t}$ which is defined as the score s_t of the rule t if the point satisfies that rule or 0 otherwise.

$$\hat{s}_{i,t} = \begin{cases} s_t & \text{if } z_{i,t} = 1 \\ 0 & \text{if } z_{i,t} = 0 \end{cases}$$

That is, $\hat{s}_{i,t} = s_t z_{i,t}$. We can model this bilinear constraint with a common big-M strategy using the following four linear constraints:

$$\hat{s}_{i,t} \geq s_t - (1 - z_{i,t})M \qquad \forall t,\ \forall i \tag{4.13a}$$
$$\hat{s}_{i,t} \leq s_t + (1 - z_{i,t})M \qquad \forall t,\ \forall i \tag{4.13b}$$
$$\hat{s}_{i,t} \geq -z_{i,t}M \qquad \forall t,\ \forall i \tag{4.13c}$$
$$\hat{s}_{i,t} \leq z_{i,t}M \qquad \forall t,\ \forall i \tag{4.13d}$$

Where $M = \max\{|s| \in A\}$ is sufficient to fullfil the logic and to obtain the tiniest relaxation of the big-M inequalities.

The total score of each sample $\boldsymbol{x}^i$ obtained by the model is tracked through the integer variables $\bar{s}_i$:

$$\bar{s}_i = \sum_t \hat{s}_{i,t} \qquad \forall i \tag{4.14a}$$

Finally, we help the model to select the threshold b_t for the subset $B_{\mathcal{D}} \subseteq \{1, ..., p\}$ of binary features. In fact, if the rule t selects a binary feature j, the threshold is trivially imposed to be 0.5 by the following constraints:

$$a_{j,t} - 1 \leq b_t - 0.5 \qquad \forall t, \forall j \in B_{\mathcal{D}} \tag{4.15a}$$

$$1 - a_{j,t} \geq b_t - 0.5 \qquad \forall t, \forall j \in B_{\mathcal{D}} \tag{4.15b}$$

To handle the optimization phase, we used the same piece-wise linear approximation as proposed for standard Risk Scores. Indeed, in this case, the LCPA approach is not applicable due to the presence of the decision variables used to assess wether a sample has to obtain the score for each rule, making it impossible to use gradient-based approximation for the objective function.

Putting all of this together gives the MILP formulation for learning OGRS reported in the next page.

$$\min \sum_{i \in \mathcal{I}} \xi_i \tag{4.16a}$$

$$\text{s.t.} \tag{4.16b}$$

$$\sum_{j=1}^{p} a_{j,t} = 1 \qquad \forall t \tag{4.16c}$$

$$\sum_{t=1}^{T} a_{j,t} \leq 1 \qquad \forall j \tag{4.16d}$$

$$\sum_{k=j+1}^{p} \sum_{s=1}^{t-1} a_{k,s} \leq (1 - a_{j,t})\, T \qquad \forall j, \forall t \tag{4.16e}$$

$$\boldsymbol{a}_t^T \boldsymbol{x}^i \geq b_t - 1 + z_{i,t} \qquad \forall t, \forall i \tag{4.16f}$$

$$\boldsymbol{a}_t^T \boldsymbol{x}^i \leq b_t - \epsilon + z_{i,t} \qquad \forall t, \forall i \tag{4.16g}$$

$$\sum_{i} z_{i,t} \geq 1 \qquad \forall t \tag{4.16h}$$

$$\sum_{i} z_{i,t} \leq N - 1 \qquad \forall t \tag{4.16i}$$

$$\hat{s}_{i,t} \geq s_t - (1 - z_{i,t})M \qquad \forall t, \forall i \tag{4.16j}$$

$$\hat{s}_{i,t} \leq s_t + (1 - z_{i,t})M \qquad \forall t, \forall i \tag{4.16k}$$

$$\hat{s}_{i,t} \geq -z_{i,t}M \qquad \forall t, \forall i \tag{4.16l}$$

$$\hat{s}_{i,t} \leq z_{i,t}M \qquad \forall t, \forall i \tag{4.16m}$$

$$\bar{s}_i = \sum_{t} \hat{s}_{i,t} \qquad \forall i \tag{4.16n}$$

$$\xi_i \geq f(v_k) + f'(v_k)(y^i(\bar{s}_i + \beta) - v_k) \qquad \forall i, \forall k \tag{4.16o}$$

$$a_{j,t} - 1 \leq b_t - 0.5 \qquad \forall t, \forall j \in B_{\mathcal{D}} \tag{4.16p}$$

$$1 - a_{j,t} \geq b_t - 0.5 \qquad \forall t, \forall j \in B_{\mathcal{D}} \tag{4.16q}$$

$$\xi_i \in \mathbb{R}^+ \; \forall i, \qquad \beta \in \mathbb{R} \tag{4.16r}$$

$$a_{j,t} \in \{0,1\} \; \forall j, \forall t, \qquad z_{i,t} \in \{0,1\} \; \forall i, \forall t \tag{4.16s}$$

$$s_t \in A,\; b_t \in \mathbb{R} \; \forall t \tag{4.16t}$$

$$\hat{s}_{i,t} \in \mathbb{R} \; \forall i, \forall t, \qquad \bar{s}_i \in \mathbb{R} \; \forall i \tag{4.16u}$$

4.4 Numerical Experiments With Binary Features

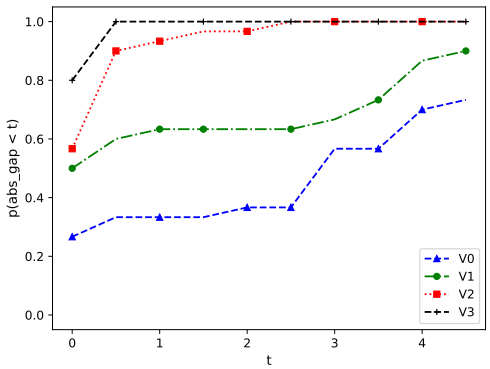
In this section we show the results obtained by the standard Risk Score model, proposed in section 4.1, in the case of binary features. We considered a benchmark of 6 datasets (`tik-tak-toe`, `mushrooms`, `income`, `mammo`, `spambase`, `balance`), from the UCI repositories (see Table 4.1), performing an 80/20 train/test split. For all the experiments we used $A = \{-5, ..., 5\}$ thus setting the big-M constant to 5. Moreover, to avoid numerical issues, we set ϵ to 10^{-4} that is much larger than the one employed for the IntFeasTol parameter of Gurobi (10^{-6}).

Dataset	N	p
balance	576	20
income	32561	36
mushrooms	8124	113
mammo	961	14
tik-tak-toe	957	27
spambase	4601	57

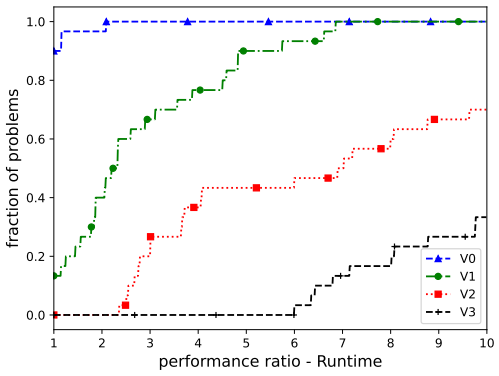
Table 4.1: Description of the datasets used in the computational experiments. All datasets are from the UCI collection (Dua and Graff, 2017).

Preliminary Experiments

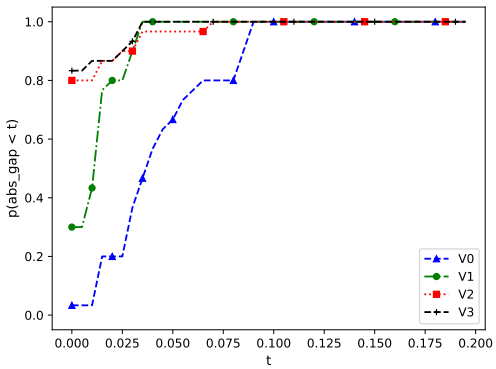
The first experiments we carried out concern the assessment of the performance of the RS model for binary features as we vary the set V of the tangent points that are used to obtain the piece-wise linear underestimator of the logistic loss. In particular, we are interested in finding the set of tangent points that is able to obtain the best AUC on the test set. For this reason, we run the experiments for Risk Scores with 3 rules and for five different random seeds for each dataset, obtaining a total of 30 different problems. The Gurobi time limit has been set to 3600s. As shown by the performance profiles in Figure 4.2, choosing V_3 to build the linear piece-wise underestimator seems to provide the best overall AUC and the best approximation with the respect to the true logistic loss computed. This is clearly at cost the computational time due to the presence of more constraints in in the MILP formulation. For this reason, we decided to use the V_3 setting to compare the performance of our model with the state-of-art LCPA approach of Ustun and Rudin (2019) in the next section.



(a) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by any model.



(b) Performance profiles of the running times for solving the MIP models.



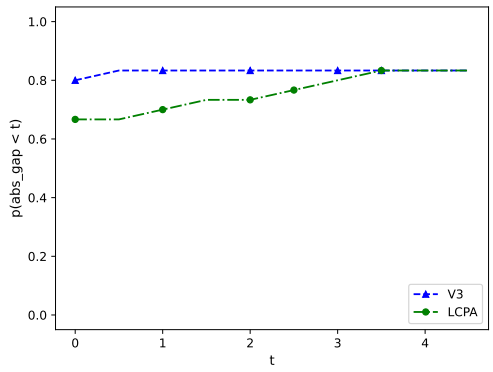
(c) Cumulative distribution of the absolute gap from the best (exact) loss attained on the training set by any model.

Figure 4.2: Comparison of the performance of Risk Scores models for binary features when different tangent point sets (V_0, V_1, V_2 V_3) are employed.

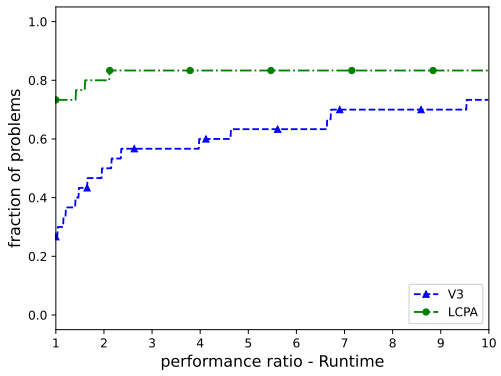
ORS Performance Evaluation

In this section, we describe numerical results of the comparison between our method and the LCPA approach of Ustun and Rudin (2019) on datasets with binary feature with the aim to show that our method can be a valid alternative to handle the optimization. Also in this case, we used the datasets reported in Table 4.1 and we performed 80/20 train/test splits testing 5 seeds, obtaining 30 different problems. Since we are interested in the comparison by varying the number of the rules, we avoid to validate this parameter and we study the performance of the two approaches on each problem with 3, 5 and 7 rules. The results of the experiment are shown in Figure 4.3 in the form of cumulative distribution of absolute gap from the best AUC on the test set and performance profiles (Dolan and Moré, 2002) of runtime. The

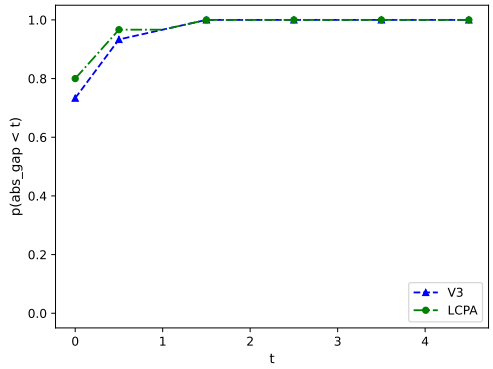
general trend which can be observed is that both our method with V_3 and LCPA are able to obtain very similar results in terms of AUC for each number of rules considered. The major differences are related to the running times. Indeed, LCPA tends to be faster than our method but it seems to scale worse when increasing the number of the rules.



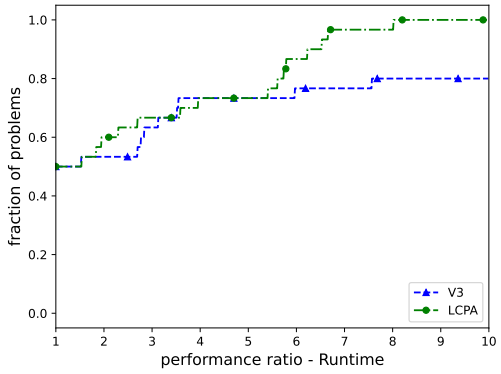
(a) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach. The number of rules is equal to 3.



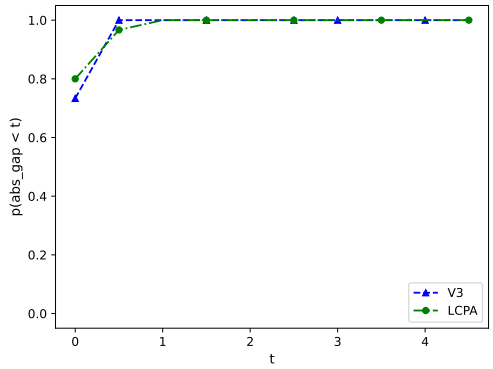
(b) Performance profiles of the running times for solving the MIP models. The number of rules is equal to 3.



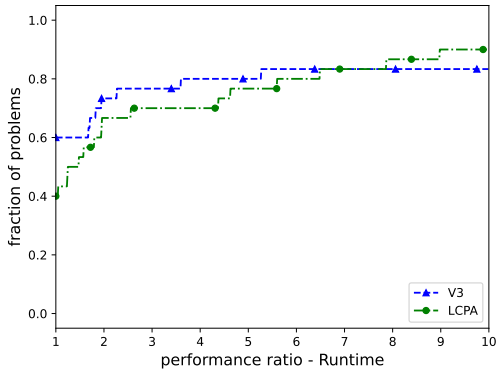
(c) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach. The number of rules is equal to 5.



(d) Performance profiles of the running times for solving the MIP models. The number of rules is equal to 5.



(e) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach. The number of rules is equal to 7.



(f) Performance profiles of the running times for solving the MIP models. The number of rules is equal to 7.

Figure 4.3: Comparison of the performance of Risk Scores on binary feature varying the number of rules. We report both running time and the out-of-sample AUC for our method with V3 and the state-of-art LCPA approach from Ustun and Rudin (2019).

4.5 Numerical Experiments With Continuous Feature

In this section we show the results obtained by the Generalized Risk Score model, proposed in Section 4.3 on general datasets with both continuous and binary features. We considered a benchmark of 10 datasets from the UCI repositories (see Table 4.2), performing an 80/20 train/test split. For all the experiments we used the same parameters as for the standard Risk Score model previously described.

Dataset	N	p
breast	568	30
diabetes	768	8
heart disease	296	13
ionosphere	350	33
parkinsons	194	22
sonar	207	60
spectf	266	44
heart failure	299	12
wholesale	439	7
haberman	305	3

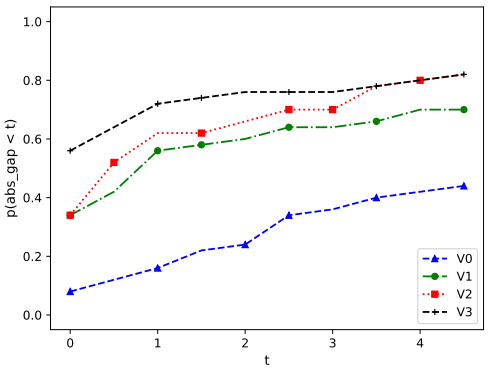
Table 4.2: Description of the datasets used in the computational experiments. All datasets are from the UCI collection (Dua and Graff, 2017).

Preliminary Experiments

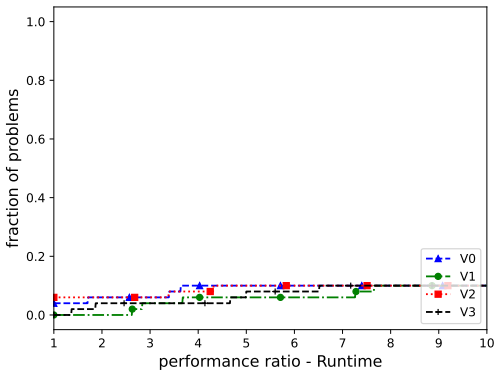
As previously made for Risk Score models, in this section we show how different choices for the set of tangent points lead to different performance in Generalized Risk Scores. For this purpose, we used the datasets reported in Table 4.2. As made for the case of binary features, in Figure 4.4, we report numerical experiments in the form of performance profiles with respect to 3 rules models. Also in this case, choosing V_3 as set of tangent points lead to the best AUC and to the best approximation with respect to the true logistic loss. However, due to the combinatorial complexity of the MILP model, the solver is able to certify the optimum in 3600s only in 10% of the problems even in the case of V_0.

OGRS Performance Evaluation

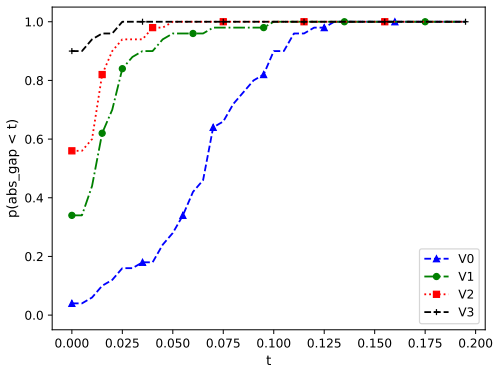
In this section we show the last part of the experiments. In this case we compare the OGRS method with the LCPA approach after having binarized each feature. In the seminal work of Ustun and Rudin (2019) authors performs an a-priori binarization of each continuous feature through discretization in a predefined number of intervals. This strategy, however, lead to a huge increase in the final number of features



(a) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by any model.



(b) Performance profiles of the running times for solving the MIP models.



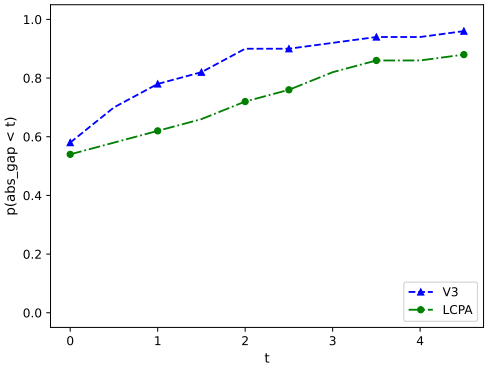
(c) Cumulative distribution of the absolute gap from the best (exact) loss attained on the training set by any model.

Figure 4.4: Comparison of the performance of Risk Scores models for binary features when different tangent point sets (V_0, V_1, V_2 V_3) are employed.

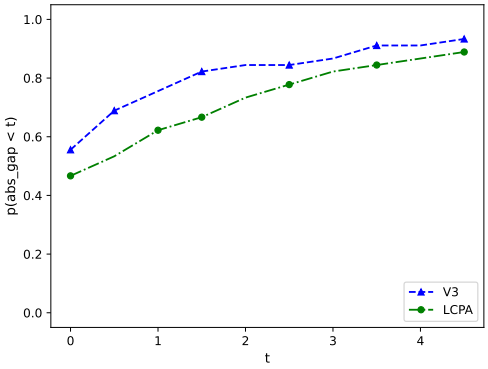
thus compromising the ability of the LCPA method to obtain good cuts for the approximation of this new high dimensional objective. For this reason, we decided to perform a more robust binarization by fitting a single-node decision tree on each feature, using the log-loss as impurity measure. More precisely, we defined a new dataset where each feature $j \in \{1, ..., p\}$ has been binarized according to the threshold t_j obtained by fitting a single decision stump on that feature. In this way, we used the transformation $x_j \to \mathbb{1}\{x_j \geq t_j\}$ to obtain a new binarized dataset with the same number of feature, thus avoiding to compromise the quality of the LCPA approach. Note that this method acts as a greedy baseline for our approach. We are interested in showing that our formulation of OGRS is able to induce a final model by defining each rule in a global perspective. Also in this case, we show numerical experiments

of the 10 datasets reported in Table 4.2 and 5 seeds for a total of 50 different problem instances. We set the Gurobi time limit to 3600 and we report the performance profiles of the out-of-sample AUC for 3, 5, and 7 rules in Figure 4.5. We do not report the running times since the LCPA approach is applied to the binarized dataset and it is clearly always the fastest one. Moreover, we employed a warm start strategy to initialize the optimization phase with a good feasible solution, i.e., at each iteration we solve the problem on just one rule maintaining the other variables fixed to the values obtained at the previous iterations.

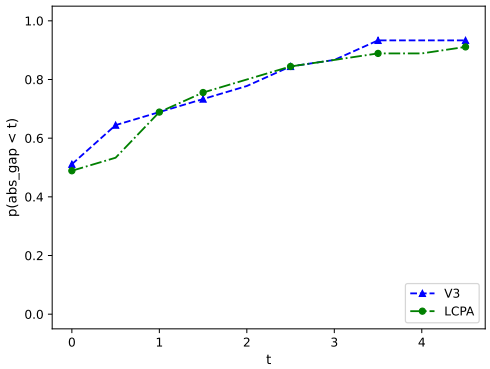
We observe that our proposed model appears to obtain the best performance, consistently being the most likely to obtain an AUC value close to the best one, as the gap parameter t increases. Our model is thus not only most frequently the best one, but when it is not, it is still the one with the lowest probability of falling shorter than any given threshold from the best result. Finally, we can observe that in case of 7 rules the two methods are quite comparable and this is mainly bacause the OGRS model becomes clearly more difficult to optimize.



(a) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach on the binarized dataset. The number of rules is equal to 3.



(b) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach on the binarized dataset. The number of rules is equal to 5.



(c) Cumulative distribution of the absolute gap from the best test AUC score accuracy value attained by our method and the LCPA approach on the binarized dataset. The number of rules is equal to 7.

Figure 4.5: Comparison of the performance of Generalized Risk Scores varying the number of rules. We report the out-of-sample AUC for our method with V3 and the LCPA approach from Ustun and Rudin (2019) with binarization.

4.6 Concluding Remarks

In this Chapter we reviewed the literature of Risk Score which are essentially very sparse linear models with small integer weights. Then, we proposed to use the piece-wise linear approximation of Sato et al. (2016) rather than the LCPA approach of Ustun and Rudin (2019) to handle the non linear logistic loss in a MILP context. We shown how these two approaches perform quite similar on datasets with binary feature. Moreover, we extended the concept of Risk Scores by proposing Generalized Risk Scores as a special form of Generalized Additive Models and we gave a MILP formulation which aims at the induction of Optimal GRS in the case of rules made with a single clause. Numerical experiments highlighted how this new generalized model is able to handle each kind of feature and to derive rules in a global perspective, outperforming standard approaches based to a preprocessing phase of binarization, maintaining its semplicity and increasing the ease of editability from an end user.

Chapter 5

Decision Trees for Local Explanations

As previously discussed in Chapter 2, the concept of explainability in machine learning is taking on an increasingly important role (Rudin, 2019; Burkart and Huber, 2021). In particular, in the context of deep learning, several methods have been proposed to obtain explanations for a single point prediction (*local explanations*), using interpretable surrogate models (Ribeiro et al., 2016; Kenny and Keane, 2021; Blanco-Justicia et al., 2020).

In this chapter we describe an application of Contextual Bandits to a feature-based selection of a single Decision Tree (DT) in a Random Forest, aiming at a final prediction that is explainable (Aldinucci et al., 2022). The proposed system learns a mapping which links the input feature space to the space of grown decision trees in order to identify an interpretable and effective predictor for that input sample. We clarify that the proposed recommendation system is itself a black-box model but its recommendation leads to an interpretable model. More specifically, although our method leads to post-hoc local interpretability, the recommended tree does not act as a surrogate explainer of the whole black-box Random Forest. Our intuition is that the presence of such a black-box layer, which guarantees a dynamic and data-driven selection of decision trees, can approximate the predictive performances of the Random Forest. Due to this context-based nature, we will refer to our approach as Contextual Classification Tree (CCT) or Contextual Regression Tree (CRT) for, respectively, classification and regression tasks.

More in detail, the problem is addressed within a Contextual Multi-Armed Bandit framework, which represents a subclass of Markov Decision Processes (MDP) with an unitary length of each episode (Gampa and Fujita, 2019). The usage of Multi-Armed Bandit models for recommendations is well known in the literature and it is employed in fields such as advertisements, personalized news articles, healthcare and finance (Zhang et al., 2021; Li et al., 2010; Durand et al., 2018; Shen et al., 2015).

In our case, the action taken by the agent consists in the choice of one tree of

the Random Forest on the basis of the input features which act like the observed context. This choice is determined by a parametrized policy which encodes a probability distribution over the trees of the Random Forest. After each choice, the agent receives a reward that depends both on the context and the selected action. Considering that we focus on prediction, the adopted reward is related to the predictive ability of the chosen tree: high reward if the selected one obtains a low prediction error and vice-versa.

Parameters of policy are learnt, using policy gradient Reinforcement Learning (RL) methods (Gampa and Fujita, 2019), in order to maximize the cumulative rewards over a certain period. Gradient methods for bandits are well motivated in (Sutton and Barto, 1998) and can be applied also in the contextual case (Pan et al., 2019).

5.1 Preliminaries

Reinforcement Learning (RL) aims to train autonomous agents in order to learn behavior through trial-error interactions with a dynamic environment. The agent chooses an action at each time step, which changes the state of the environment in an unknown way, and receives feedback based on the consequence of the action.

RL can be formally defined as a Markov Decision Process (MDP) (Arulkumaran et al., 2017) consisting of:

- a set of states $\mathcal{S}$;
- a set of actions $\mathcal{A}$;
- transition dynamics $d(s_{t+1}|s_t, a_t)$ mapping a state, action couple at time t into a distribution of possible states at time $t+1$;
- a Reward Function

$$r_{t+1} = r(s_t, a_t, s_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R};$$

- a discount factor $\gamma \in [0, 1)$, where lower values place more emphasis on immediate rewards.

In addition to this, RL models are based on the use of a policy π:

$$\pi : \mathcal{S} \to p(\mathcal{A} = a|\mathcal{S})$$

i.e., a mapping from states to a probability distribution over the set $\mathcal{A}$ of actions, which determines the behavior of the agent in the choice of adequate actions. Our

method considers episodic MDP, where the agent interacts with the environment repeatedly in episodes of fixed length T. Each episode is characterized by a sequence of states and actions $\tau = (s_t, a_t)_{t=0}^{T-1}$, usually called *trajectory*. The accumulated reward $R(\tau)$ for a trajectory is given as

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}.$$

We stress the fact that the policy affects the probability of a given trajectory, i.e., $\tau = \tau_\pi$.

Policy search methods, which represent one of the main approaches to solve RL problems (Arulkumaran et al., 2017), are conceived to find the best policy π^* that maximizes the expected return over all possible *trajectories*:

$$\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)].$$

Policy search methods use parametrized policies π_θ and, in deep RL (Arulkumaran et al., 2017), deep neural networks are employed to approximate these policies. The network outputs a probability distribution over the set of actions A and the action with the highest probability will define the move that the policy should do in order to get the highest final reward. In this setting, policy parameters θ are updated using a gradient-based approach in order to maximize the expected return:

$$J_\theta = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)].$$

The gradient is computed by means of the REINFORCE algorithm (Williams, 1992), which belongs to the class of Likelihood-ratio methods (Deisenroth et al., 2013). These methods make use of the so called "likelihood-ratio" trick i.e., given a random variable $X \sim p(x|\theta)$ and a function $f(x)$ we have:

$$\nabla_\theta \mathbb{E}_X\big[f(x)\big] = \nabla_\theta \int_X f(x)p(x|\theta)dx = \tag{5.1}$$

$$= \int_X \nabla_\theta p(x|\theta)f(x)dx = \int_X p(x|\theta)\frac{\nabla_\theta p(x|\theta)}{p(x|\theta)}f(x)dx = \tag{5.2}$$

$$= \mathbb{E}_X\big[f(x)\nabla_\theta \log p(x|\theta)\big] \tag{5.3}$$

Thus, we can use the loss function $\mathcal{L}$:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\cdot R(\tau^{(i)})$$

where N is the number of episodes that define a training batch.

5.2 Proposed Method

Our method, as summarized in Fig. 5.1, exploits the structure of Random Forests to provide a general locally-interpretable and feature-based recommendation system. Given a Random Forest $RF = \{h_1, h_2, \ldots, h_B\}$ with B trees and an input space $\mathcal{X}$, the aim of our approach consists of training an autonomous agent to learn a policy of tree recommendation which, given an input point $\boldsymbol{x} \in \mathcal{X}$ as context and a state $s \in \mathcal{S}$ returns a probability distribution over the indexes of the trained trees of the forest $\mathcal{T} = \{1, 2, \ldots, B\}$:

$$\pi(\cdot) : \mathcal{X} \times \mathcal{S} \rightarrow p(\mathcal{A} = a|\mathcal{S}, \mathcal{X})$$

Therefore, our action space $\mathcal{A}$ is the discrete index set $\mathcal{T}$ of all the possible B trees in the Random Forest.

As mentioned above, our scenario can be viewed as a contextual multi-armed bandit problem. Contextual bandit is a variant of the bandit problem, where at each episode i the agent conditions its action a_i on the context $\boldsymbol{x}_i$ of the environment and observes the reward $r(a_i)$ for the chosen action. Moreover, it is important to note that the action affects only the immediate reward and for this reason our formulation is one-state and one-step episodic, meaning that our episode is made by only one step and a_i does not condition the next state which will be always the same. Hence, for this particular set up, the following holds:

$$\begin{aligned} &T = 1, \quad s_t = s, \quad a_t = a_{x_i} = a_i, \\ &R(s_t, a_t) = R(a_{x_i}) = r(a_i), \\ &\pi(\cdot) : \mathcal{X} \rightarrow p(\mathcal{A} = a|\mathcal{X}) \end{aligned}$$

Note that in our case, since $T = 1$ and there is only one state, the reward depends only on the action taken given the context $\boldsymbol{x}$. Thus, there is no need to consider any transition dynamics over the trajectory and the resulting MDP has only one state.

In our scenario the reward brings information about the predictive ability of the trees of the forest. Both the ground truth values and the predictions of the trees, which act like signals from the environment, contribute to the computation of the reward. We define two different rewards for binary classification and regression problems.

In the classification context we use the function:

$$r(a_x) = \begin{cases} +1 & h_{a_x}(\boldsymbol{x}) = y \\ -1 & \text{otherwise,} \end{cases} \tag{5.4}$$

where $y \in \{-1, 1\}$ is the ground truth value corresponding to the observed feature vector $\boldsymbol{x}$ and $h_{a_x}(\boldsymbol{x}) \in \{-1, 1\}$ is the prediction provided by the tree selected

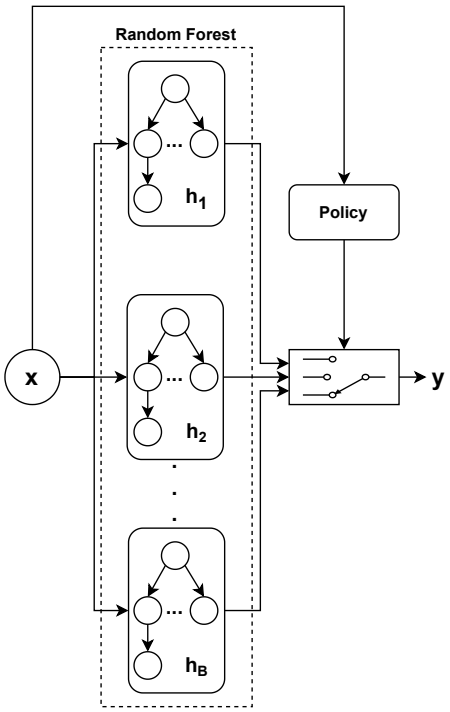


Figure 5.1: A high level view of our system: a Deep Neural Network (DNN) based policy selects the tree of a trained Random Forest that will make the final prediction. Each data point x is observed as context by the agent and the policy aims to recommend the best tree of the Random Forest in terms of predictive performances.

through the action a_x. Concerning regression problems, the reward function depends on the squared prediction error $d = (h_{a_x}(\boldsymbol{x}) - y)^2$ as follows:

$$r(a_x) = 1 - 2d \tag{5.5}$$

where, for each example, d is normalized in $[0, 1]$ respect to the squared errors of the best and worst tree in the forest. The use of a linear reward (w.r.t. the distance), rather than a step wise one, allows to give greater importance to the well-aimed actions of the agent. Indeed, using a step-wise reward like the one chosen for the classification case, would cause the agent to over exploit the tree in the forest with the minimum squared error with respect to the target and this may lead to overfitting.

We employ a DNN of parameters θ to approximate the policy (Deep RL). In the training phase, we used the classical RL trial-error approach to update the policy parameters θ for every batch of examples (see Algorithm 4).

The main problem in multi-armed bandits is the need for balancing *exploration* and *exploitation*. We want to avoid the agent to greedily select just those trees that seem to appear best, as they may in fact be suboptimal due to imprecision in the knowledge of the agent. Thus, during the training, we want the policy to explore the action space by choosing also seemingly not good trees to obtain more informa-

tion about them. For this reason, we introduced a regularization term $H_\theta(a_{x_i}|x_i) = -\pi_\theta(a_{x_i}|x_i)log(\pi_\theta(a_{x_i}|x_i))$ in our loss that gives importance also at the entropy of the distribution of the actions in the current batch. In this way, we are able to obtain a good *exploration/exploitation* trade-off, preventing the agent to rapidly converge to suboptimal tree recommendations.

Algorithm 4 Policy Update

Input: θ_k, $\{(x_i, y_i)\}_{i=1}^N$ (batch)
for $i = 1$ to N **do**
    Sample actions $a_{x_i} \sim \pi_{\theta_k}(a_{x_i}|x_i)$
    Compute rewards $r(a_{x_i})$
end for
Obtain $\mathcal{D} = \{(x_i, r(a_{x_i}))\}_{i=1}^N$
Define $\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^N(\log \pi_\theta(a_{x_i}|x_i)r(a_{x_i}) + \lambda H_\theta(a_{x_i}|x_i))$
Compute gradient $\nabla_\theta \mathcal{L}(\theta)$
Update parameters $\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta \mathcal{L}(\theta)$
Return θ_{k+1}

Predictions on new data are finally obtained with the support of the trained RL system which dynamically selects trees based on the input data. In particular, for a given new instance $\boldsymbol{x}_{\text{new}}$, the system detects a suitable tree predictor $h_{i^*}(\cdot)$ selecting the index i^* as

$$i^* = \arg\max_{i \in \mathcal{T}} \pi_\theta(a_{\boldsymbol{x}_{\text{new}}}|\boldsymbol{x}_{\text{new}}).$$

Final predictions are computed following the path of decision rules from $h_{i^*}(\cdot)$.

5.3 Numerical Experiments

We applied our method both for binary classification and regression problems. We used benchmark datasets from LIBSVM (`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`) and UCI (`https://archive.ics.uci.edu/ml/datasets.php`) data repositories to assess our method. A description of datasets involved in our experiments is reported in Table 5.1 and Table 5.2.

The aim of the experiments is to compare the predictive performances of CCT and CRT with the ones of CART, Random Forest (the one on which our method is built on) and a supervised baseline (to bring an empirical evidence on the need of our RL approach). Both for CART and Random Forest we set the maximum depth of trees at most equal to four. This choice is related to *comprehensibility*: it is important that a human can rely on easily understandable explanations. This fact is dependent on psychological and social implications (Molnar, 2020). Setting the maximum tree

depth equal to four ensures also that the Miller's law (Miller, 1956) holds. Furthermore, we grew up the Random Forest with fifty tree predictors: a forest with hundreds of trees would compromise the exploration phase of the agent, slowing down the convergence during the training of the policy.

The original whole dataset is split up in three parts (64 % training set, 16 % validation set, 20 % test set) and the training phase consists of two steps. The first one involves the training of both CART and Random Forest, while the second is about the training of the neural network to approximate the policy.

We first used a grid search strategy, exploiting the validation set, to choose the best maximum depth for the baseline CART model. As anticipated above, in order to favour the interpretability, only maximum depths of 2, 3 and 4 have been considered as candidates. Subsequently, a Random Forest of 50 decision trees predictors, each grown up to the validated maximum depth, is trained on the training set. The other hyperparameters of both CART and Random Forest are set equal to their default values from the scikit-learn package (Pedregosa et al., 2011).

As network structure, we employed a 3-layers architecture with ReLU activations and a 0.2 Dropout layer (Srivastava et al., 2014). An early stopping strategy (Montavon et al., 2012) is used to stop the training according to the predictive performances on the validation set. For both classification and regression problems we have used ADAM optimizer (Kingma and Ba, 2014) with an initial learning rate of 10^{-3} and no weight decay in combination with a cosine scheduler (Loshchilov and Hutter, 2017) to slightly reduce the learning rate after each epoch. Finally, we set the regularization term λ, related to the entropy of the policy output distribution, to 10^{-4} since we empirically observe to be a good tradeoff between exploration and exploitation.

Regarding the supervised baseline, we used the same network structure with a softmax activation in the last layer and the same hyperparameters. In classification context, we created the dataset labeling each sample with the label of the tree with the greatest probability among the set of the trees that lead to the correct prediction. In case there is no tree of the forest that correctly predicts the example, the label is randomly chosen according to a uniform distribution over the Random Forest trees. Similarly, in the case of regression, the chosen label is the one related to the tree in the forest that produces the smallest quadratic error with respect to the target. We maintained the same dataset split configuration and, also for this model, an early stopping strategy is used respect to the predictive performances on the validation set.

	N	p
Phishing	11055	68
Biodeg	1055	41
Heart	270	25
Spam	4601	57
A2A	2265	119

Table 5.1: Overview of the datasets for binary classification

	N	p
California	20640	8
Friedman	5000	20
Yearpred	463715	90
Abalone	4177	8
CPU	8192	12

Table 5.2: Overview of the datasets for regression

Dataset	**CCT**	**CART**	**Supervised**	**RF**	**Tree Depth**
Phishing	**0.9394 ± 0.001**	0.9209	0.589 ± 0.009	0.9294	4
Biodeg	**0.8025 ± 0.016**	0.7867	0.624 ± 0.012	0.8057	4
Heart	**0.7963 ± 0.030**	0.7778	0.605 ± 0.038	0.8333	3
Spam	**0.9233 ± 0.003**	0.8881	0.706 ± 0.013	0.9077	4
A2A	**0.8293 ± 0.008**	0.8190	0.718 ± 0.010	0.7748	3

Table 5.3: Accuracy and standard deviation for classification tasks. We report in bold the best results between CCT, CART (Decision Tree) and the Supervised approach among five different seeds. For the sake of completeness we also report here the results obtained by the Random Forest (RF).

5.4 Discussion

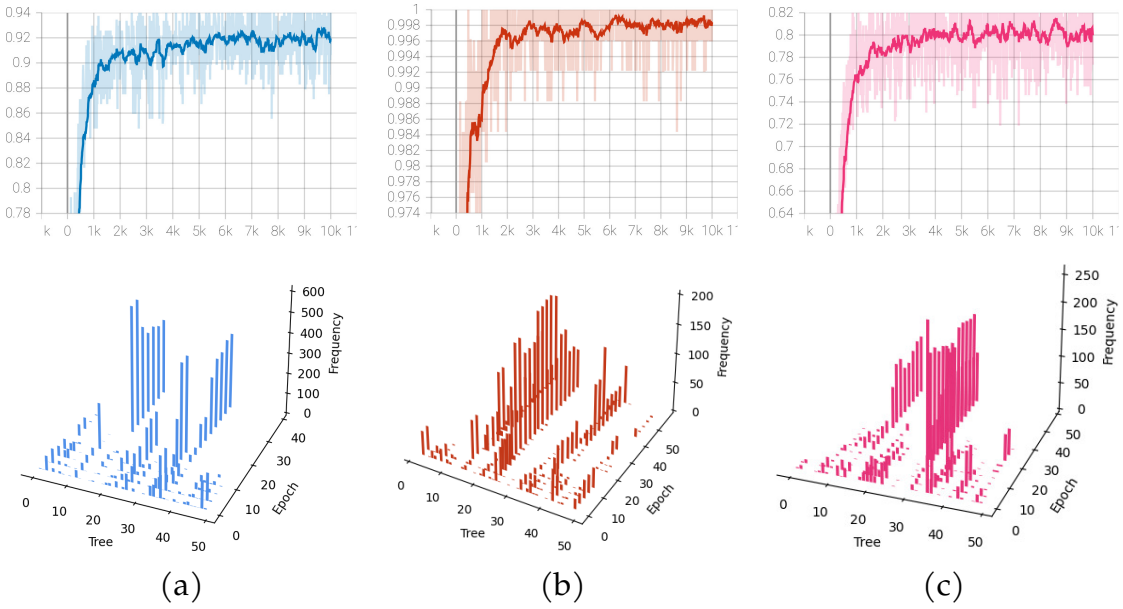
The obtained predictive results, reported in Tables 5.3, 5.4, indicate that our method substantially outperforms CART and the supervised approach in both regression and classification tasks.

The supervised strategy has proved particularly unsuccessful in the classification and regression scenario due to the ill-defined mechanism of assignment of labels. We believe that a full supervised approach restricts the exploration of action space and adversely affects the ability to generalize on unseen data.

Conversely, the more accurate predictive results, obtained by our method, are strongly linked to the capability of the agent to more thoroughly explore the action space. In this respect, the role of the entropy regularization is fundamental to encourage exploration. A weak regularization with respect to the entropy of the action

Dataset	Scale	CRT	CART	Supervised	RF	Tree Depth
California	10^4	7.456 ± 0.032	7.828	**7.420 ±0.015**	7.602	4
Friedman	10^0	**3.014 ± 0.013**	3.040	3.122 ±0.008	2.718	4
Yearpred	10^1	**0.995 ± < 0.001**	1.010	1.007±<0.001	1.001	4
Abalone	10^0	**2.379 ± 0.004**	2.389	2.487 ± 0.045	2.307	4
CPU	10^0	**4.205 ± 0.026**	4.566	4.281 ± 0.023	4.173	4

Table 5.4: Root Mean Squared Error and standard deviation for regression tasks. We report in bold the best results between CRT, CART (Regression Tree) and the Supervised approach among five different seeds. For the sake of completeness we also report here the results obtained by the Random Forest (RF).



(a) (b) (c)

Figure 5.2: Reward (above) and Tree selection histograms (below) during training in classification: (a) Spam; (b) Digits; (c) A2A.

distribution inevitably would lead to a very greedy tree selection and therefore our method will tend to perform similar to the supervised baseline. Figure 5.2 highlights the trade-off between exploration and exploitation varying the training epochs. It is indeed possible to note that rewards do not rise considerably after 2000 epochs and, moreover, after the first tens epochs, the RL agent learns to select the same two or three trees of the Random Forest. This latter point evidences that the agent has got sufficient information about the reward obtained from the estimators and therefore the policy tends to converge choosing always a subset of trees that provides better performance. This means that only few estimators of the Random Forest contribute to the increase of the reward over the training epochs, basically excluding a large

part of the trees. As a matter of fact, some works (Bernard et al., 2009; Zhang and Wang, 2009; Latinne et al., 2001; Tripoliti et al., 2010), although with different arguments, agree on the existence of well performing sub-forests which constitute the core of the whole ensemble.

We also underline that the obtained results are particularly significant since good predictive accuracy is obtained without losing in terms of interpretability.

5.5 Concluding Remarks

In this Chapter, we described a feature-based method which is able to recommend a single shallow tree in a Random Forest. We modelled this problem as a contextual multi-armed bandit and solved by RL methods. Since decision trees operate on input features to form their decision paths, our intuition is that an intelligent system able to appropriately recommend a single estimator of a grown Random Forest, can be advantageous in terms of predictive performance. A very important characteristic of our method is that an interpretable model is returned. More specifically, in order to compute a new prediction, differently from Random Forest, only one of its decision trees is in turn employed.

Our method outperforms CART and a supervised baseline in both regression and classification tasks and the obtained results are also comparable with the ones of Random Forest.

Interestingly, we also observed that the trained system chooses the same two or three decision trees after the first tens of epochs. This fact suggests that it is possible to obtain sub-forests with a very small number of trees but with performance that are competitive with the whole forest.

Chapter 6

Conclusions

In this thesis work, we have dealt with optimization problems related to the learning of interpretable models. Specifically, emphasis was put on decision trees and risk scores that are very common choices in many field where transparency is a major requirement. In this context, we gave a brief review of linear models that are, still now, the easiest choice in supervised learning. Then, we addressed the problem of learning classification trees and we proposed two different methods that are capable to obtain competitive performance with respect to the state-of-art optimization algorithms. Subsequently, we introduced a study focused on refining the expressiveness of risk score models. This improvement is achieved by expanding the scope of these estimators within a more generalized framework. Additionally, within the context of local explanations, we proposed a novel recommendation system designed specifically for random forests. This system dynamically associates each data point with a single shallow tree within the ensemble, contributing to a more nuanced understanding of the decision-making process of the model. On the basis of the results achieved, new possibilities of research may include:

- The extention of the loss-optimal classification tree framework to the multi-class setting;
- The definition of different types of rules to improve the performance of generalized risk score models.

Appendix A

Publications

Journal Papers

T. Aldinucci, M. Lapucci, "Loss-Optimal Classification Trees: A Generalized Framework and the Logistic Case" , *TOP, An Official Journal of the Spanish Society of Statistics and Operations Research* (2024), To appear.

Bibliography

Aghaei, S., Gómez, A., and Vayanos, P. (2021). Strong optimal classification trees. *arXiv preprint arXiv:2103.15965*.

Aitkenhead, M. J. (2008). A co-evolving decision tree classification method. *Expert Systems with Applications*, 34(1):18–25.

Aldinucci, T. (2023). A novel memetic strategy for optimized learning of classification trees. *arXiv preprint arXiv:2305.07959*.

Aldinucci, T., Civitelli, E., Di Gangi, L., and Sestini, A. (2022). Contextual decision trees. *arXiv preprint arXiv:2207.06355*.

Aldinucci, T. and Lapucci, M. (2024). Loss-optimal classification trees: A generalized framework and the logistic case. *TOP*. To appear.

Antman, E. M., Cohen, M., Bernink, P. J., McCabe, C. H., Horacek, T., Papuchis, G., Mautner, B., Corbalan, R., Radley, D., and Braunwald, E. (2000). The timi risk score for unstable angina/non–st elevation mi: a method for prognostication and therapeutic decision making. *Jama*, 284(7):835–842.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

Bach, F., Jenatton, R., Mairal, J., Obozinski, G., et al. (2012). Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106.

Barros, R. C., Basgalupp, M. P., De Carvalho, A. C., and Freitas, A. A. (2011). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312.

Bennett, K. P. (1992). Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Bennett, K. P. and Mangasarian, O. L. (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization methods and software*, 1(1):23–34.

Bennett, K. P. and Mangasarian, O. L. (1994). Multicategory discrimination via linear programming. *Optimization methods and Software*, 3(1-3):27–39.

Bernard, S., Heutte, L., and Adam, S. (2009). On the selection of decision trees in random forests. In *2009 International Joint Conference on Neural Networks*, pages 302–307. IEEE.

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

Bertsimas, D., Kung, J., Trichakis, N., Wang, Y., Hirose, R., and Vagefi, P. A. (2019). Development and validation of an optimized prediction of mortality for candidates awaiting liver transplantation. *American Journal of Transplantation*, 19(4):1109–1118.

Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012:107–121.

Blanco-Justicia, A., Domingo-Ferrer, J., Martínez, S., and Sánchez, D. (2020). Machine learning explainability via microaggregation and shallow decision trees. *Knowledge-Based Systems*, 194:105532.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Chapman and Hall/CRC, Boca Raton London New York Washington, D.C.

Burgess, E. W. (1928). Factors determining success or failure on parole. *The workings of the indeterminate sentence law and the parole system in Illinois*, pages 221–234.

Burkart, N. and Huber, M. F. (2021). A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317.

Cantú-Paz, E. and Kamath, C. (2003). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68.

Carreira-Perpinán, M. A. and Tavallali, P. (2018). Alternating optimization of decision trees, with application to learning sparse oblique trees. *Advances in neural information processing systems*, 31.

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

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., and Elhadad, N. (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730.

Chan, K.-Y. and Loh, W.-Y. (2004). Lotus: An algorithm for building accurate and comprehensible logistic regression trees. *Journal of Computational and Graphical Statistics*, 13(4):826–852.

Črepinšek, M., Liu, S.-H., and Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, 45(3):1–33.

Czajkowski, M. and Kretowski, M. (2012). Does memetic approach improve global induction of regression and model trees? In *Swarm and Evolutionary Computation: International Symposia, SIDE 2012 and EC 2012, Held in Conjunction with ICAISC 2012, Zakopane, Poland, April 29-May 3, 2012. Proceedings*, pages 174–181. Springer.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine learning*, 6(1):81–92.

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213.

D'Onofrio, F., Grani, G., Monaci, M., and Palagi, L. (2022). Margin optimal classification trees. *arXiv preprint arXiv:2210.10567*.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dua, D. and Graff, C. (2017). Uci machine learning repository.

Dunn, J. W. (2018). *Optimal trees for prediction and prescription*. PhD thesis, Massachusetts Institute of Technology.

Durand, A., Achilleos, C., Iacovides, D., Strati, K., Mitsis, G. D., and Pineau, J. (2018). Contextual bandits for adapting treatment in a mouse model of de novo carcinogenesis. In Doshi-Velez, F., Fackler, J., Jung, K., Kale, D., Ranganath, R., Wallace, B., and Wiens, J., editors, *Proceedings of the 3rd Machine Learning for Healthcare Conference*, volume 85 of *Proceedings of Machine Learning Research*, pages 67–82. PMLR.

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer, Heidelberg New York Dordrecht London.

Figueiredo, M. A., Nowak, R. D., and Wright, S. J. (2007). Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):586–597.

Fonti, V. and Belitser, E. (2017). Feature selection using lasso. *VU Amsterdam research paper in business analytics*, 30:1–25.

Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing*, pages 819–845. Springer, Verlag Berlin Heidelberg 2003.

Friedman, J. H. et al. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Computers*, 26(4):404–408.

Gage, B. F., Waterman, A. D., Shannon, W., Boechler, M., Rich, M. W., and Radford, M. J. (2001). Validation of clinical classification schemes for predicting stroke: results from the national registry of atrial fibrillation. *Jama*, 285(22):2864–2870.

Gampa, P. and Fujita, S. (2019). Banditrank: Learning to rank using contextual bandits. *arXiv preprint arXiv:1910.10410*.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., and Scheinberg, K. (2021). Optimal decision trees for categorical data via integer programming. *Journal of global optimization*, 81(1):233–260.

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual.

Hastie, T. and Tibshirani, R. (1995). Generalized additive models for medical research. *Statistical methods in medical research*, 4(3):187–196.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, New York, NY.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*, volume 43.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, One Broadway 12th Floor Cambridge, MA 02142.

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

Intrator, J., Allan, E., and Palmer, M. (1992). Decision tree for the management of substance-abusing psychiatric patients. *Journal of substance abuse treatment*, 9(3):215–220.

Jia, D., Zheng, G., and Khan, M. K. (2011). An effective memetic differential evolution algorithm based on chaotic local search. *Information Sciences*, 181(15):3175–3187.

John, G. H. (1995). Robust linear discriminant trees. In *Pre-proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 285–291. PMLR.

Jones, G. (1998). Genetic and evolutionary algorithms. *Encyclopedia of Computational Chemistry*, 2:1127–1136.

Jovanovic, M., Radovanovic, S., Vukicevic, M., Van Poucke, S., and Delibasic, B. (2016). Building interpretable predictive models for pediatric hospital readmission using tree-lasso logistic regression. *Artificial Intelligence in Medicine*, 72:12–21.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Kenny, E. M. and Keane, M. T. (2021). Explaining deep learning using examples: Optimal feature weighting methods for twin systems using post-hoc, explanation-by-example in xai. *Knowledge-Based Systems*, 233:107530.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283.

Kretowski, M. (2008). A memetic algorithm for global induction of decision trees. In *SOFSEM 2008: Theory and Practice of Computer Science: 34th Conference on Current Trends in Theory and Practice of Computer Science, Novỳ Smokovec, Slovakia, January 19-25, 2008. Proceedings 34*, pages 531–540. Springer.

Latinne, P., Debeir, O., and Decaestecker, C. (2001). Limiting the number of trees in random forests. In *International workshop on multiple classifier systems*, pages 178–187. Springer.

Laurent, H. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17.

Li, L., Chu, W., Langford, J., and Schapire, R. (2010). A contextual-bandit approach to personalized news article recommendation. *Computing Research Repository - CORR*.

Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. (2020). Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR.

Liu, E., Hu, T., Allen, T. T., and Hermes, C. (2023). Optimal classification trees with leaf-branch and binary constraints applied to pipeline inspection. *Available at SSRN 4360508*.

Loh, W.-Y. and Vanichsetakul, N. (1988). Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83(403):715–725.

Loshchilov, I. and Hutter, F. (2017). Sgdr: Stochastic gradient descent with warm restarts. *arXiv: Learning*.

Mansueto, P. and Schoen, F. (2021). Memetic differential evolution methods for clustering problems. *Pattern Recognition*, 114:107849.

McGinley, A. and Pearse, R. M. (2012). A national early warning score for acutely ill patients. *Bmj*, 345.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Montavon, G., Orr, G., and Müller, K.-R. (2012). Neural networks-tricks of the trade second edition. *Springer, DOI*, 10:978–3.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989.

Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 135(3):370–384.

Olanow, C. W., Watts, R. L., and Koller, W. C. (2001). An algorithm (decision tree) for the management of parkinson's disease (2001):: Treatment guidelines. *Neurology*, 56(suppl 5):S1–S88.

Pan, F., Cai, Q., Tang, P., Zhuang, F., and He, Q. (2019). Policy gradients for contextual recommendations. In *The World Wide Web Conference*, page 1421–1431, New York, NY, USA. Association for Computing Machinery.

Papagelis, A. and Kalles, D. (2001). Breeding decision trees using evolutionary techniques. In *ICML*, volume 1, pages 393–400.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Ross, A., Lage, I., and Doshi-Velez, F. (2017). The neural lasso: Local linear sparsity for interpretable explanations. In *Workshop on Transparent and Interpretable Machine Learning in Safety Critical Environments, 31st Conference on Neural Information Processing Systems*, volume 4.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, pages 206–215.

Sato, T., Takano, Y., Miyashiro, R., and Yoshise, A. (2016). Feature subset selection for logistic regression via mixed integer optimization. *Computational Optimization and Applications*, 64:865–880.

Shen, W., Wang, J., Jiang, Y.-G., and Zha, H. (2015). Portfolio choices with orthogonal bandit learning. In *Twenty-fourth international joint conference on artificial intelligence*.

Song, Y.-Y. and Ying, L. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Teasdale, G. and Jennett, B. (1974). Assessment of coma and impaired consciousness: a practical scale. *The Lancet*, 304(7872):81–84.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tripoliti, E. E., Fotiadis, D. I., and Manis, G. (2010). Dynamic construction of random forests: Evaluation using biomedical engineering problems. In *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine*, pages 1–4. IEEE.

Ustun, B. and Rudin, C. (2016). Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102:349–391.

Ustun, B. and Rudin, C. (2019). Learning optimized risk scores. *J. Mach. Learn. Res.*, 20(150):1–75.

Vincent, J. L., Moreno, R., Takala, J., Willatts, S., De Mendonça, A., Bruining, H., Reinhart, C., Suter, P., and Thijs, L. G. (1996). The sofa (sepsis-related organ failure assessment) score to describe organ dysfunction/failure: On behalf of the working group on sepsis-related problems of the european society of intensive care medicine (see contributors to the project in the appendix).

Wells, P., Hirsh, J., Anderson, D., Lensing, A. A., Foster, G., Kearon, C., Weitz, J., D'Ovidio, R., Cogo, A., Prandoni, P., et al. (1995). Accuracy of clinical assessment of deep-vein thrombosis. *The Lancet*, 345(8961):1326–1330.

Wells, P. S., Anderson, D. R., Bormanis, J., Guy, F., Mitchell, M., Gray, L., Clement, C., Robinson, K. S., and Lewandowski, B. (1997). Value of assessment of pretest probability of deep-vein thrombosis in clinical management. *The Lancet*, 350(9094):1795–1798.

Wells, P. S., Anderson, D. R., Rodger, M., Forgie, M., Kearon, C., Dreyer, J., Kovacs, G., Mitchell, M., Lewandowski, B., and Kovacs, M. J. (2003). Evaluation of d-dimer in the diagnosis of suspected deep-vein thrombosis. *New England Journal of Medicine*, 349(13):1227–1235.

Wells, P. S., Anderson, D. R., Rodger, M., Stiell, I., Dreyer, J. F., Barnes, D., Forgie, M., Kovacs, G., Ward, J., and Kovacs, M. J. (2001). Excluding pulmonary embolism at the bedside without diagnostic imaging: management of patients with suspected pulmonary embolism presenting to the emergency department by using a simple clinical model and d-dimer. *Annals of internal medicine*, 135(2):98–107.

Wells, P. S., Owen, C., Doucette, S., Fergusson, D., and Tran, H. (2006). Does this patient have deep vein thrombosis? *Jama*, 295(2):199–207.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Wolf, S. J., McCubbin, T. R., Feldhaus, K. M., Faragher, J. P., and Adcock, D. M. (2004). Prospective validation of wells criteria in the evaluation of patients with suspected pulmonary embolism. *Annals of emergency medicine*, 44(5):503–510.

Wu, X. and Che, A. (2019). A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, 82:155–165.

Zhang, H. and Wang, M. (2009). Search for the smallest random forest. *Statistics and its Interface*, 2(3):381.

Zhang, M., Wang, G., Ren, L., Li, J., Deng, K., and Zhang, B. (2021). Metonr: A meta explanation triplet oriented news recommendation model. *Knowledge-Based Systems*, page 107922.