Training decision trees using mixed-integer optimisation

Joel Vääräniemi

School of Science

Bachelor's thesis
Espoo 16.6.2023

Supervisor

Prof. Fabricio Oliveira

Advisor

MSc (Tech.) Nikita Belyak

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Joel Vääräniemi

Title Training decision trees using mixed-integer optimisation

Degree programme Bachelor's Programme in Science and Technology

Major Mathematics and Systems Sciences **Code of major** SCI3029

Teacher in charge Prof. Fabricio Oliveira

Advisor MSc (Tech.) Nikita Belyak

Date 16.6.2023 **Number of pages** 23 **Language** English

Abstract

Decision trees are popular machine learning methods used in classification and regression problems which nowadays have numerous applications in the real world. Traditionally, decision trees have been trained by greedy methods yielding locally optimal solutions. Although the problem of training a globally optimal decision tree is NP-hard, developments in optimal decision tree methods started to appear in the literature after the recent improvements in hardware and solver software.

In this thesis, we study a mixed-integer optimisation (MIO) approach used to define optimal classification trees (OCT). OCT trains an optimal tree for a given depth making axis-aligned splits. The objective of OCT is to minimise the misclassification error of training data given the preference on the complexity of the tree structure, i.e., the number of splits made. OCT has shown better prediction accuracy than similar greedy methods such as CART but the training time required by the OCT method is not well studied. Therefore, we perform an analysis of OCT on a chosen data set focusing on its training time, also considering training accuracy, for different combinations of hyper-parameters. These results are compared to corresponding ones generated by CART in order to evaluate the difference in performance between the two methods. Furthermore, we study how much the training time of OCT improves when using CART-generated initial solutions as a warm start for OCT. In addition, we examine how the model performs when the maximum amount of splits is strictly predetermined without other mechanisms for penalising the complexity of the tree.

The results show that training OCT is burdensome already for medium depths as the 30-minute time limit was not enough to train the trees with depths of 3 or higher when no penalty on tree complexity was used. Although we were able to train trees up to depths of 4 within the time limit with a higher penalty on complexity, the training accuracies were not considerably higher compared to those trained with CART. However, using warm starts provided reductions in training times up to a factor of 1.6. Additionally, we were able to prove the ability of OCT to significantly outperform CART in terms of training accuracy when using the maximum number of splits approach.

Keywords optimal classification trees, decision trees, mixed-integer optimisation

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Tekniikan kandidaatintyön tiivistelmä

Tekijä Joel Vääräniemi

Työn nimi Training decision trees using mixed-integer optimisation

Koulutusohjelma Teknistieteellinen kandidaattiohjelma

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Prof. Fabricio Oliveira

Työn ohjaaja DI Nikita Belyak

Päivämäärä 16.6.2023 **Sivumäärä** 23 **Kieli** Englanti

Tiivistelmä

Päätöspuut ovat suosittuja koneoppimismalleja, joita käytetään luokittelu- ja regressio-ongelmiin monissa käytännön sovelluksissa. Perinteisesti päätöspuita on koulutettu ahneilla menetelmillä, jotka tuottavat lokaalisti optimaalisia ratkaisuja. Vaikka globaalisti optimaalisen päätöspuun kouluttaminen on NP-kova ongelma, laskentatehon ja ratkaisijaohjelmistojen kehityksen seurauksena myös toimivia optimaalisia päätöspuumenetelmiä on esitetty viime aikoina.

Tässä työssä tutkitaan optimaalisia luokittelupuita (OCT), menetelmää, joka kouluttaa optimaalisia luokittelupuita kokonaislukuoptimointia (mixed-integer optimisation) hyödyntäen. Menetelmä kouluttaa optimaalisen puun annetulle syvyydelle käyttäen akseleihin nähden kohtisuoria jakoja. OCT:n tavoitteena on minimoida datapisteiden luokittelun virhe samalla huomioiden annetun preferenssin puun kompleksisuudesta, eli jakojen määrästä. OCT:n ennustustarkkuuden on osoitettu olevan korkeampi kuin vastaavien ahneiden päätöspuuvaihtoehtojen, kuten CART:n, mutta OCT:n kouluttamisajoista ei ole paljoa tutkimusta. Tästä syystä OCT:n suorituskykyä tutkittiin valitulla datasetillä useilla eri ennakkoparametrien yhdistelmillä keskittyen kouluttamisaikoihin, kuitenkin kouluttamistarkkuutta silmällä pitäen. Näitä tuloksia verrattiin vastaaviin CART:n avulla saatuihin tuloksiin suorituskykyjen erojen tarkastelun vuoksi. Lisäksi OCT:n kouluttamisajan nopeutumista tutkittiin, kun CART:n avulla tuotettuja alkuratkaisuja käytettiin lämpiminä käynnistyksinä (engl. warm start). Lopulta tarkasteltiin, kuinka OCT suoriutuu, kun tehtyjen jakojen määrä rajoitetaan ennalta määrättyyn lukuun ilman muita kompleksisuutta rajoittavia mekanismeja.

Saadut tulokset osoittavat, että OCT:n kouluttaminen on raskasta jo keskisuurille puun syvyyksille. Asetettu 30 minuutin aikaraja ei riittänyt kouluttamaan optimaalista puuta ilman kompleksisuuteen kohdistuvaa rankaisua, kun puun syvyys oli 3 tai enemmän. Vaikka optimaalisia puita saatiin koulutettua aikarajan puitteissa syvyyteen 4 asti rankaisemalla kompleksisuutta enemmän, eivät niiden kouluttamistarkkuudet olleet huomattavasti korkeampia vastaaviin CART:n avulla koulutettuihin puihin verrattuna. Kuitenkin lämpimät käynnistykset osoittivat lyhentävän optimaalisten puiden kouluttamisaikoja kymmeniä prosentteja. Lisäksi OCT:n avulla kyettiin aikarajan puitteissa kouluttamaan puita, joiden kouluttamistarkkuus oli merkittävästi korkeampi CART:hen verrattuna, kun kompleksisuutta rajoitettiin ainoastaan määrittämällä jakojen enimmäismäärä.

Avainsanat optimaaliset luokittelupuut, päätöspuut, kokonaislukuoptimointi

Contents

1 Introduction

A decision tree is a machine learning method commonly used in classification and regression problems. The design of a decision tree consists of branch and leaf nodes. When training a decision tree, branch nodes are assigned branching rules based on the attributes of the data. According to these branching rules, every data point is eventually allocated to one of the leaf nodes. Based on the labels of data points that were allocated to a designated leaf node, the leaf node is given a corresponding indicator that is commonly considered to be the most frequent label among the points. After training the tree, each leaf indicator label is further used as a prediction value or class for new data points that get allocated to that leaf node.

Due to the generally high prediction accuracy, i.e., the ability to make correct predictions for unseen data, decision trees are considered to be useful tools in classification and regression problems. In addition, decision trees are highly interpretable due to their uncomplicated rule-based design. Due to their interpretability, decision trees are broadly exploited in real-world problems. They are widely used for example in the healthcare sector, for the problems such as diagnosis and clinical decision-making (Podgorelec et al., 2002). Also, decision trees have been used in the banking industry for credit approval (Chitra and Subashini, 2013) to mention just a few of the possible applications. At the moment there are numerous off-the-shelf decision tree software available for use, thus making them an appealing alternative amongst different machine learning techniques.

Traditionally, decision trees have been trained with greedy top-down algorithms such as CART (Breiman et al., 1984). Although being computationally affordable, CART and similar algorithms construct the decision tree by sequentially making locally optimal splits in the input space, thus ultimately providing a decision tree structure that might not necessarily be optimal in terms of training accuracy. Training accuracy represents how accurately a model captures the characteristics of the training data, which is normally used as the metric for training classification and regression models. If a trained machine learning model is too tailored for given training data and does not generalise well for new data, the model is said to overfit. In order to avoid overfitting with decision trees, pruning is usually needed.

In the past, an approach of creating a globally optimal decision tree, i.e., the structure that provides the highest training accuracy, has been computationally infeasible for moderate-sized datasets. However, as computational hardware and optimisation software have improved (Bixby, 2012), studies targeting the methodologies for creating globally optimal decision tree methods started to appear in the literature.

In this thesis, we address the methodology proposed in Bertsimas and Dunn (2017) named optimal classification trees (OCT). An OCT approach represents a decision tree as a mixed-integer optimisation (MIO) problem and has been proven to find optimal trees for medium-sized trees. The OCT approach generates a globally optimal decision tree. That is due to the fact that when deciding on the position of the splits in the input space, the OCT algorithm considers all of the splits simultaneously. For classification tasks, the training accuracy is measured by in-sample accuracy,

i.e., the portion of correctly classified data points within training data. Moreover, the prediction accuracy is measured by out-of-sample accuracy, i.e., the portion of correctly classified data points from unseen data. The objective of OCT is to minimise the in-sample accuracy while taking into account the given requirement regarding the decision tree complexity, i.e., the total number of splits. Results show that OCT reaches on average better out-of-sample accuracy than CART.

In this thesis, we study the performance and behaviour of OCT on a chosen data set. In particular, the focus is on the time taken for training the decision tree as there is no distinguished literature thereof. We also compare the results of globally optimal OCT against a greedy approach, CART, to analyse the trade-off between training time and training accuracy. In addition, we study how OCT behaves when the maximum amount of splits allowed in the model is strictly predetermined, or when a greedily generated initial solution is utilised as a warm start for solving the OCT MIO problem.

2 Related work

Binary decision trees perform a binary decision at each branch node, i.e., divide the points into two branches. This is the most popular decision tree type and it has been known for a long time that constructing an optimal binary decision tree is an NP-hard problem (Laurent and Rivest, 1976). Therefore, in the past, the problem of training a decision tree was tackled by greedy top-down algorithms making one split after another sequentially. Greedy algorithms were not necessarily considered to be superior in terms of prediction accuracy compared to optimal models. Yet, due to being computationally tractable and requiring smaller training time, greedy methods were considered to be a standard approach for decision trees.

In contrast to greedy approaches, it is possible to train optimal decision trees. We define an optimal decision tree to be the tree which is optimal with respect to a given objective under the constraints of the tree structure. Usually, the objective is to minimise the training error of the tree. In case of a classification problem that would mean minimising the misclassification error, i.e., maximising the in-sample accuracy. However, as the objective may vary, one should be careful with the meaning of optimality associated with decision trees.

In recent years, there were numerous attempts to solve the optimal decision tree problem in the literature. Norouzi et al. (2015) utilise continuous optimisation to optimise the upper bound for tree loss with the help of Stochastic Gradient Descent. Although the authors use an approach which optimises a global objective function, this method cannot be generally used for defining the optimal decision tree for given parameters. Also Blanquero et al. (2021) make use of continuous optimisation in their new approach called optimal randomized classification trees (ORCT). In ORCT, randomisation is used to train an optimal decision tree model without integer variables. That is because the number of integer variables is usually the bottleneck for minimising the computational time required to solve MIO problems.

Verhaeghe et al. (2020) considered a constraint programming model for clas-

sification problems combining several existing algorithms. However, the model is applicable only for binary classification tasks. Another recent paper utilises Dynamic Programming (Lin et al., 2020). The data is preprocessed such that it preserves optimality guarantee but on the other hand generates a considerable number of binary variables. Therefore, the training algorithm applies only to moderate-scale data sets.

A breakthrough considering optimal decision trees took place with the appearance of OCT (Bertsimas and Dunn, 2017) allowing for a novel MIO classification tree formulation which is considered in this thesis. OCT constructs optimal binary classification trees with respect to maximising in-sample accuracy. In the OCT formulation, the maximum depth of the tree is bounded by a predefined value. The tree complexity, i.e., the number of splits made in the tree, can be limited also by altering the value of the complexity parameter implemented in the model. OCT design considers univariate splits (axis-aligned splits) at each branch node, i,e., each branch node considers only a single feature to split on. However, in the same paper (Bertsimas and Dunn, 2017) a multivariate alternative, OCT-H, is introduced. This approach allows for the consideration of hyperplane splits that take multiple variables into account at a time. Often multivariate models are considerably more complicated than their univariate counterparts, yet OCT-H is actually as easy to train as OCT.

Inspired by OCT, other optimal MIO decision tree approaches have also emerged. Verwer and Zhang (2019) introduce BinOCT which uses binary encoding to reduce the number of decision variables. This method is less dependent on the training data size which reduces the search space drastically. Although the BinOCT is able to operate with higher tree depths, the binary preprocessing sacrifices guarantee of the optimality for the resulting decision tree. Aghaei et al. (2021) propose flowOCT, which operates merely on problems with binary features. For these types of problems, flowOCT appears to be more efficient compared to OCT in terms of computational time.

3 Methodology

3.1 Background for OCT

The design of the OCT formulation is inspired by greedy alternatives such as CART. However, rather than finding a locally optimal tree, it aims to consider the problem holistically and choose the splits made in a globally optimal manner.

CART is a top-down decision tree algorithm which starts the procedure from the root node, i.e., the initial branch node. At the root node CART makes an observation of the entire training data and it splits the data into two parts as CART builds a binary tree. The splitting criterion with CART in classification problems is the Gini Impurity. The algorithm chooses the split which minimises the Gini Impurity measure among all of the possible splits. The Gini impurity measure for a given split

is calculated by:

$$Gini = \frac{n_l}{n}(1 - \sum_{i=1}^{K} p_{L,i}^2) + \frac{n_r}{n}(1 - \sum_{j=1}^{K} p_{R,j}^2),$$

where n is the number of data points in the given branch node, n_l and n_r the number of data points falling in the left and right branches respectively, K is the number of possible classes, $p_{L,i}^2$ is the probability of points from the left branch belonging to class i and $p_{R,j}^2$ is the probability of points from the right branch belonging to class j. As CART performs univariate splits, it selects the optimal feature to split on, and additionally, the optimal value of the chosen feature as a point to split the data on. After CART completes the split for the root node, the algorithm continues downwards recursively making the splits in this manner for every branch node until the termination criterion is met, i.e., the tree has grown up to the maximum depth. If a node does not proceed to branch, it is a leaf node and thereby predicts a label based on the most common label from the data points in that node. As the algorithm executes locally optimal splits recursively, it constructs a locally optimal tree in terms of minimising the misclassification error. For more details on CART please refer to Breiman et al. (1984).

The MIO model behind OCT mimics the CART tree design yet produces an optimal solution, i.e., an optimal tree. This is achieved by formulating the MIO problem such that it makes decisions regarding the possible branches at the same time. Solving the MIO problem produces the optimal tree for a given data set. Thus, solving the MIO problem represents the training phase for OCT. OCT uses the misclassification error of the tree as the objective function for the optimisation problem. Therefore, unlike CART, OCT does not need splitting criterions such as the Gini Impurity measure in order to generate the splits.

3.2 MIO formulation of the OCT model

In this thesis, we consider the version of the OCT formulation presented in Bertsimas and Dunn (2019). The proposed formulation is closely followed in our presentation of the model. Some minor modifications are made and mentioned further in the text.

In the following notation, ordinary letters $(n, K, ...)$ express scalars, lowercase bold letters $(\mathbf{x}, ...)$ express vectors, uppercase bold letters $(\mathbf{X}, ...)$ express matrices and calligraphic type letters $(\mathcal{T}, ...)$ express sets. Furthermore, $[n]$ expresses the set $\{1, ..., n\}$.

The starting point for the classification problem is as follows. We have training data $(\mathbf{X}, \mathbf{y})$ which consists of n observations $(\mathbf{x}_i, y_i)$, $i \in [n]$. Each observation $i \in [n]$ has p features $\mathbf{x}_i \in \mathbb{R}^p$ and a class label $y_i \in K$ indicating which of the possible K classes the observation is assigned to. Without loss of generality, the training data is normalized to the 0-1 interval such that $\mathbf{x}_i \in [0, 1]^p$, $\forall i \in [n]$.

As previously mentioned, the OCT generates a binary tree using only univariate splits. In order to successfully create an MIO formulation which captures the architecture of the tree and is able to find the optimal solution, we have to predetermine

the value for the maximum depth of the tree D. This allows us to create a maximal tree for the depth D which has $T = 2^{D+1} - 1$ nodes. A decision tree is a maximal tree if every branch node applies a split and every leaf node is located at the maximal depth. The nodes are indexed by $t \in [T]$. The maximal tree with a depth of 2 is shown in Figure 1.

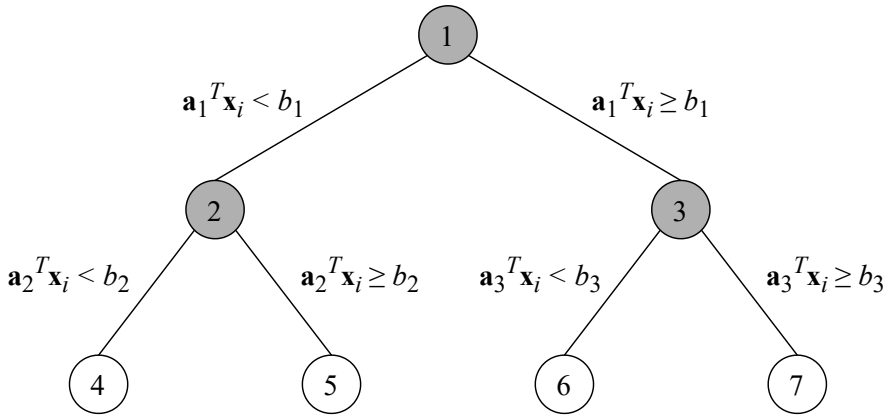

Figure 1: The maximal tree for depth of 2. Branch nodes are grey and leaf nodes are white.

The nodes in the tree are separated into two sets: branch nodes $t \in \mathcal{T}_B = \{1, ..., \lfloor T/2 \rfloor\}$ and leaf nodes $t \in \mathcal{T}_L = \{\lfloor T/2 \rfloor + 1, ..., T\}$. The split rule applied to a branch node is of the form $\mathbf{a}^\mathsf{T}\mathbf{x} < b$. A data point proceeds to the left branch if the inequality is satisfied, and proceeds to the right branch otherwise. When a data point reaches a leaf node, the leaf node makes a class prediction for it.

For the node t, the notation $p(t)$ refers to its immediate parent node and $\mathcal{A}(t)$ defines the set of its all ancestors. We also define $\mathcal{A}_L(t)$ as the set of ancestors of t, from which the left branch was followed when forming a path from the root node to t. Similarly $\mathcal{A}_R(t)$ are the ancestors whose right branch was followed on the path from the root node to t and $\mathcal{A}(t) = \mathcal{A}_L(t) \cup \mathcal{A}_R(t)$. For instance, in the tree in Figure 1, $\mathcal{A}(6) = \{1, 3\}, \mathcal{A}_L(6) = \{3\}$, and $\mathcal{A}_R(6) = \{1\}$.

A split made at node $t \in \mathcal{T}_B$ is identified with variables $\mathbf{a}_t \in \mathbb{R}^p$ and $b_t \in \mathbb{R}$. In order to ensure that each split is univariate, the hyperplane formed by the corresponding split takes only one feature into account. To ensure this, we assume the elements of variable vector $\mathbf{a}_t$ to take binary values such that their sum is 1. The splitting rule is based on the inequalities associated with each branch node as shown in Figure 1 for nodes 1, 2 and 3. If the selected feature value of a given point is less than the splitting value b_t, the data point advances to the left branch, and otherwise to the right branch. This branching procedure continues until a leaf node is reached where the point classification is executed.

Furthermore, it should be possible not to apply a split at a branch node. This is done by introducing binary variables d_t to track whether the splitting occurred for each branch node, such that $d_t = 1$ when a split is made, and $d_t = 0$ otherwise. If a branch node does not apply a split, then we set $\mathbf{a}_t = \mathbf{0}$ and $b_t = 0$. In this situation, the inequality for the left branch $0 < 0$ is never true, and hence all points proceed to

the right branch. This allows us to stop growing the tree early without having to introduce new variables to deal with points which would not reach the bottom of the tree otherwise. The previous mechanism is enforced by the following constraints:

$$\begin{aligned} &\sum_{j=1}^{p} a_{jt} = d_t, && \forall t \in \mathcal{T}_B, \\ &0 \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B, \\ &a_{jt} \in \{0,1\}, && \forall j \in [p], t \in \mathcal{T}_B. \end{aligned} \tag{1}$$

Inequality (1) is valid for b_t because we know that $\mathbf{x}_i \in [0,1]^p$. The elements of $\mathbf{a}_t$ are zero except for one, which is 1 when $d_t = 1$. Therefore, the inequality $0 \leq \mathbf{a}_t^\mathsf{T}\mathbf{x}_i \leq d_t$ holds for all i and t, and values for b_t have to be considered only for the same range from 0 to d_t.

Next, we ensure the hierarchical design of the tree. We prohibit a split on a branch node in the case where its parent node did not allow a split. This is ensured by the following constraints:

$$d_t \leq d_{p(t)}, \qquad \forall t \in \mathcal{T}_B \setminus \{1\}. \tag{2}$$

We do not set a constraint for the root node in (2).

Now we have formulated the model such that it captures the structure of the tree. Next, we have to introduce binary variables z_{it} to keep track of the points in the tree. If a point i appears in node t, $z_{it} = 1$, and $z_{it} = 0$ otherwise. Binary variables l_t are also introduced, where $l_t = 1$ if a leaf node t contains any points, and $l_t = 0$ if it contains none. With these variables, we can define the following constraints to enforce the minimum number of data points N_{min} in each leaf node:

$$z_{it} \leq l_t, \qquad \forall t \in \mathcal{T}_L,$$

$$\sum_{i=1}^{n} z_{it} \geq N_{min} l_t, \qquad \forall t \in \mathcal{T}_L, \tag{3}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \qquad \forall i \in [n], \tag{4}$$

where inequality (3) ensures that restriction for the minimum number of points is not applied if the leaf node is empty. Equality (4) ensures that every point is assigned to exactly one leaf node.

The structure of the tree is formed by the splits made. We establish the splits by applying constraints:

$$\begin{aligned} &\mathbf{a}_m^\mathsf{T}\mathbf{x}_i < b_m + M_1(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_L(t), \\ &\mathbf{a}_m^\mathsf{T}\mathbf{x}_i \geq b_m - M_2(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_R(t), \end{aligned} \tag{5}$$

where M_1 and M_2 are constants large enough such that when $z_{it} = 0$, the constraints are always satisfied. The mechanism for choosing the values for these constants is discussed further. However, we have a strict inequality in (5) which is not tractable

for MIO solvers. Therefore, we have to transform the strict inequality into a non-strict one. That is done by adding a small constant ϵ on the left-hand side of the inequality:

$$\mathbf{a}_m^\top \mathbf{x}_i + \epsilon \leq b_m + M_1(1 - z_{it}), \qquad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_L(t).$$

In order to avoid any numerical instabilities in the MIO solver, ϵ must not be too small. Therefore, ϵ is chosen to be the biggest small enough value such that it does not affect the functionality of the model. This could be implemented by choosing individual values ϵ_j for each j. We could consider the smallest non-zero distance between adjacent values of feature j as the largest valid value for ϵ_j. This distance can be calculated as:

$$\epsilon_j = \min\left\{ x_j^{(i+1)} - x_j^{(i)} \;\middle|\; x_j^{(i+1)} \neq x_j^{(i)}, i \in [n-1] \right\},$$

where the values of jth feature are sorted from the smallest to largest, thereby $x_j^{(i)}$ is the ith smallest value. The source which we utilise as a reference for the MIO formulation contained a misprint, as the values were sorted from the largest to the smallest (Bertsimas and Dunn, 2019). Here, we use a corrected version of the distance formulation. We choose to use different ϵ_j accordingly for every feature which is used in the split:

$$\mathbf{a}_m^\top(\mathbf{x}_i + \boldsymbol{\epsilon} - \epsilon_{min}) + \epsilon_{min} \leq b_m + M_1(1 - z_{it}), \qquad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_L(t),$$

where $\epsilon_{min} = \min_j\{\epsilon_j\}$. As $\mathbf{a}_m$ has only one element being 1 and others 0 when a split is made, the wanted element from ϵ is yielded by a vector multiplication.

The values for the constants M_1 and M_2 must also be predefined. As we know, both $\mathbf{a}_t^\top \mathbf{x}_i \in [0, 1]$ and $b_t \in [0, 1]$. Therefore, the largest possible value for $\mathbf{a}_t^\top(\mathbf{x}_i + \boldsymbol{\epsilon}) - b_t$ is $1 + \epsilon_{max}$, where $\epsilon_{max} = \max_j\{\epsilon_j\}$, and largest possible value for $b_t - \mathbf{a}_t^\top \mathbf{x}_i$ is 1. Therefore, we set $M_1 = 1 + \epsilon_{max}$ and $M_2 = 1$. Using these values, the final constraints which enforce the splits in the tree are:

$$\begin{aligned}
&\mathbf{a}_m^\top(\mathbf{x}_i + \boldsymbol{\epsilon} - \epsilon_{min}) + \epsilon_{min} \leq \\
&\qquad b_m + (1 + \epsilon_{max})(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_L(t), \\
&\mathbf{a}_m^\top \mathbf{x}_i \geq b_m - (1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_R(t).
\end{aligned}$$

The objective of the MIO problem is to minimise the misclassification error. Thus, we set the cost of an incorrect label prediction to 1 and the cost of a correct one to 0. We assume N_{kt} to be the number of points in node t labelled as k, and N_t is assumed to be the total number of points in node t:

$$\begin{aligned}
N_{kt} &= \sum_{i:y_i=k} z_{it}, && \forall t \in \mathcal{T}_L, k \in [K], \\
N_t &= \sum_{i=1}^{n} z_{it}, && \forall t \in \mathcal{T}_L.
\end{aligned}$$

In addition, we have to associate every leaf node t with a label, which we denote by $c_t \in [K]$. The leaf label is defined as the most common label from all the points assigned to a given leaf:

$$c_t = \arg\max_{k \in [K]} \{N_{kt}\}. \tag{6}$$

To track the assigned class label for each node, we use binary variables $c_{kt} = \mathbb{1}\{c_t = k\}$. We have to assign exactly one label for each leaf node that contains points:

$$\sum_{k=1}^{K} c_{kt} = l_t, \qquad \forall t \in \mathcal{T}_L.$$

We now know how to assign the optimal label for a given leaf node t using (6). Therefore, we define the optimal misclassification loss L_t in each leaf node t to be the number of points with the labels different from the most common class label:

$$L_t = N_t - \max_{k \in [K]} \{N_{kt}\} = \min_{k \in [K]} \{N_t - N_{kt}\},$$

which is linearized as:

$$\begin{aligned}
&L_t \geq N_t - N_{kt} - M(1 - c_{kt}), && \forall t \in \mathcal{T}_L, k \in [K], \\
&L_t \leq N_t - N_{kt}, && \forall t \in \mathcal{T}_L, k \in [K], \\
&L_t \geq 0, && \forall t \in \mathcal{T}_L,
\end{aligned}$$

where M is a constant with a large enough value to ensure that the uppermost constraint always holds. We can use $M = n$ as an appropriate value.

With the above definitions in mind, the total misclassification error can be defined as $\sum_{t \in \mathcal{T}_L} L_t$. And the complexity of the tree C, i.e., the number of splits made in the tree, is given by:

$$C = \sum_{t \in \mathcal{T}_B} d_t. \tag{7}$$

In order to make the complexity parameter α independent from the size of the data set used, the misclassification error has to be normalized against the baseline accuracy $\hat{L}$. The baseline accuracy is calculated simply by predicting the most common class from the whole data set. Hence, we can define the objective function as:

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot C. \tag{8}$$

Now the objective function (8) depicts the aim to classify the data as accurately as possible concurrently taking into account the preferable complexity of the tree predefined by complexity parameter α.

With the aforementioned definitions and constraints, we can formulate the complete MIO problem as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot C && && (9)\\
\text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), && \forall t \in \mathcal{T}_L, k \in [K],\\
& L_t \leq N_t - N_{kt}, && \forall t \in \mathcal{T}_L, k \in [K],\\
& L_t \geq 0, && \forall t \in \mathcal{T}_L,\\
& N_{kt} = \sum_{i: y_i = k} z_{it}, && \forall t \in \mathcal{T}_L, k \in [K],\\
& N_t = \sum_{i=1}^{n} z_{it}, && \forall t \in \mathcal{T}_L,\\
& \sum_{k=1}^{K} c_{kt} = l_t, && \forall t \in \mathcal{T}_L,\\
& C = \sum_{t \in \mathcal{T}_B} d_t, &&\\
& \mathbf{a}_m^\mathsf{T} \mathbf{x}_i \geq b_m - (1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_R(t),\\
& \mathbf{a}_m^\mathsf{T} (\mathbf{x}_i + \boldsymbol{\epsilon} - \epsilon_{min}) + \epsilon_{min} \leq \\
& \qquad b_m + (1 + \epsilon_{max})(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{A}_L(t),\\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n],\\
& z_{it} \leq l_t, && \forall t \in \mathcal{T}_L,\\
& \sum_{i=1}^{n} z_{it} \geq N_{min} l_t, && \forall t \in \mathcal{T}_L,\\
& \sum_{j=1}^{p} a_{jt} = d_t, && \forall t \in \mathcal{T}_B,\\
& 0 \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B,\\
& d_t \leq d_{p(t)}, && \forall t \in \mathcal{T}_B \setminus \{1\},\\
& z_{it}, l_t, c_{kt} \in \{0, 1\}, && \forall i \in [n], k \in [K], t \in \mathcal{T}_L,\\
& a_{jt}, d_t \in \{0, 1\}, && \forall j \in [p], t \in \mathcal{T}_B.
\end{aligned}
$$

In the formulation (9) we have three predefined hyper-parameters to specify: D, N_{min} and α. D is the maximum depth of the tree. N_{min} defines the smallest number of data points required to be allocated to a leaf node. α is the complexity parameter of the tree, which adjusts the trade-off between the complexity of the tree in terms of the number of splits and the in-sample accuracy.

3.3 Warm starting OCT

To speed up the optimisation process of the MIO solver, we can utilise warm starts. A warm start is a feasible initial solution typically obtained by heuristics. Providing a

strong initial solution to an MIO problem can reduce the computational time required by the solver to find the optimal solution for the MIO problem (Jiménez-Cordero et al., 2021).

CART can be used to generate a warm start solution for OCT problems. Due to the analogous tree design used in CART and OCT, a CART solution can be considered as a feasible solution for the MIO problem (9). For our purposes, providing only the values for the binary variables of the OCT model from the CART output is sufficient.

3.4 The maximum number of splits approach

OCT's objective function considers a trade-off between complexity and in-sample accuracy by means of predefined complexity parameter α. By increasing the value of the complexity parameter, the model has an incentive to train a tree with fewer splits, which, in turn, helps to avoid overfitting. In order to minimise overfitting, either the value of the complexity parameter can be maximised or the number of splits can be minimised. As an alternative to the complexity parameter, we can predetermine the maximum number of splits allowed to be used in the tree. Hereinafter we refer to such method as to the maximum number of splits approach.

We can easily transform the formulation (9) to introduce the maximum number of splits approach. First of all, we have to transform (7) into an inequality:

$$\sum_{t \in \mathcal{T}_B} d_t \leq C,$$

so that the number of splits is bounded by the value of C. In addition, the number of splits C should not be treated as a variable but rather as a constant. The value of this constant can be altered before training the OCT with the maximum number of splits approach. Finally, the objective function must be modified accordingly as follows:

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t.$$

Introducing the aforementioned changes to the original model (9) the maximum number of splits model does not have a penalising mechanism with regard to the number of splits made.

4 Experiments

4.1 Design of the experiments

We test OCT performance on the Iris data set, which can be found on the UCI machine learning repository (Dua and Graff, 2017). The Iris data has 150 data points with 4 explanatory variables, which are all decimal numbers. Each data point is labelled with one of the 3 different classes with each class containing 50 data points.

Computations were done on a modern laptop (i7-8650U with 16 GB of RAM) and the cut-off time for optimisation of a single MIO problem is set to 30 minutes (1800 seconds).

The experiments were implemented in Julia 1.8.4. The OCT was formulated using the JuMP package (version 1.8.2) and the trees were solved using Gurobi 10.0.1. CART solutions were discovered using the DecisionTree package (version 0.12.3). The implementation of the model can be found at https://github.com/gamma-opt/OptimalDecisionTrees.jl.

We train OCT for different combinations of hyper-parameters in order to test how they affect the results and to discover well-performing combinations in the context of the trade-off between OCT accuracy and training time. The maximum depth D is limited to 4 because solving the OCT formulation for higher depths requires significant computational time that does not comply with our experiments. We also do not consider the case $D = 1$, i.e., the case when only a single split is possible due to its excessive simplicity. The values of the parameter N_{min} that correspond to the minimum number of data points in the leaf are chosen to be 5% and 10% of the total number of points in the training dataset. In absolute values N_{min} was considered to be 8 and 15 data points from the whole data set of 150 points. Finally, we choose 4 different values for the complexity parameter α: 0, 0.1, 0.5 and 0.9.

As a prediction quality of a trained decision tree, we consider the in-sample accuracy of the trained OCT. We compare the in-sample accuracy as well as the computational time required for training of OCT to the corresponding units acquired from generating the decision tree using CART. To make sure the OCT algorithm complies with CART we ensured that CART does not utilise pruning. Commonly, the prediction quality of a decision tree is considered to be its out-of-sample accuracy. In this case, already in-sample accuracy gives us meaningful information for comparing the two approaches. This is because OCT and CART have the same restrictions for their splits: they are both binary classification trees making only univariate splits and they are trained with the same hyper-parameters.

Furthermore, we investigate the possibility of decreasing the computational time required for training OCT by using warm starts, which are acquired by running CART. The procedure for using a warm start is as follows. Before starting the search algorithm for the MIO problem of OCT we run CART using the predefined hyper-parameters and collect the values of the originated splits. Using these collected values we can compose a corresponding feasible solution for the MIO problem which functions as a starting point for OCT. We inject the binary values of this solution into the according decision variables of the OCT formulation. Therefore, when initially running CART we in fact consider a tree analogous to OCT yet we consequently reduce the search space needed for OCT to find the optimal solution. As OCT is a deterministic optimisation model, its optimal solution does not change with regard to whether or not a warm start is used.

Finally, we investigate the behaviour of the OCT training time in cases when instead of a complexity penalty the maximum number of splits is considered.

4.2 Results

Table 1: Performance for OCT and CART on chosen parameters. Training times are in seconds.

D	N_{min}	α	Training time	In-sample accuracy	CART training time	CART in-sample accuracy
2	8	0.000	41.260	0.960	0.000	0.953
		0.100	39.070	0.960		
		0.500	15.880	0.960		
		0.900	18.660	0.667		
	15	0.000	55.870	0.960	0.000	0.953
		0.100	30.050	0.960		
		0.500	17.150	0.960		
		0.900	14.220	0.667		
3	8	0.000	1800.000	–	0.000	0.960
		0.100	65.070	0.960		
		0.500	8.990	0.960		
		0.900	9.650	0.667		
	15	0.000	1800.000	–	0.000	0.953
		0.100	42.830	0.960		
		0.500	5.050	0.960		
		0.900	20.330	0.667		
4	8	0.000	1800.000	–	0.000	0.960
		0.100	234.120	0.960		
		0.500	36.490	0.960		
		0.900	14.870	0.667		
	15	0.000	1800.000	–	0.000	0.953
		0.100	371.820	0.960		
		0.500	16.090	0.960		
		0.900	24.940	0.667		

The results of computational experiments for OCT and CART algorithms are presented in Table 1. The table represents training times in seconds and in-sample accuracies for OCT and CART for different combinations of the chosen hyper-parameters. Complexity parameter α relates only to OCT as we are using a version of CART which does not take α as an input. That is, CART is analogous to OCT with $\alpha = 0$, as CART does not have a penalising mechanism for tree complexity. As one can notice, OCT does not significantly outperform CART with respect to in-sample accuracy for cases when the OCT model is solved to optimality within the predefined solution time limit. The in-sample accuracy improves by less than 1% at best and not at all in some cases when using OCT compared to CART. However, this is reasonable as in-sample accuracy is relatively close to perfect with CART even on a depth of 2. The OCT model is unavailable to train a tree, i.e., obtain an optimal solution within 30 minutes, for $D = 3$ and $D = 4$ when $\alpha = 0$. As lower values of α reduce the emphasis on the simplicity of the tree structure, the in-sample accuracy of the OCT tree could have been higher for low values of α. However, this phenomenon can not be verified as finding the optimal OCT tree already for depths of 3 or more exceeds the predefined solution time limit for $\alpha = 0$. Nevertheless, from Table 1 one can pinpoint the correlation between the parameter α value and OCT training time as well as in-sample accuracy. For example, when $D = 3$ and $N_{min} = 15$, OCT obtains equally accurate results when $\alpha = 0.1$ or $\alpha = 0.5$ with the latter computed

significantly faster. On the other hand, when $\alpha = 0.9$, the OCT algorithm generates the tree with a substantially lower value of the in-sample accuracy. As we can see, training times for CART are negligible when compared to the times for OCT.

Table 2: Training times for OCT with and without CART-generated warm start. Here $\alpha = 0$ and training times are in seconds.

D	N_{min}	Normal OCT	With CART warm start
2	8	41.260	25.410
	15	55.870	40.510
3	8	1800.000	1800.000
	15	1800.000	1800.000
4	8	1800.000	1800.000
	15	1800.000	1800.000

Table 2 presents the training times for OCT with and without a CART-generated warm start. CART-generated warm starts are used for OCT only in cases where $\alpha = 0$ to ensure the equivalence in the number of nodes between the decision trees resulting from CART and OCT. That is because the CART algorithm trains the tree without taking the complexity penalty into account. Due to the technical limitations we faced with the DecisionTree package, we were not able to implement a variant of CART which takes complexity parameter into account. As one can observe from table 2, utilising CART-generated warm starts do improve the training time for the OCT model. Training time decreases significantly while using a warm start when $D = 2$, yet not to the extent Bertsimas and Dunn (2017) have reported. They discovered speed-ups around a factor of 2.5 with our corresponding results having improvements up to a factor of 1.6 at most. Nevertheless, the training procedure of the trees with depths $D = 3$ or $D = 4$ has shown to be unreasonably time-consuming even with warm starts.

Table 3: OCT training time for the maximum number of splits approach. Training time is in seconds.

D	N_{min}	C	Training time	In-sample accuracy
2	8	3	40.020	0.960
		2	10.200	0.960
	15	3	55.870	0.960
		2	20.630	0.960
3	8	4	1800.000	–
		3	583.900	0.973
		2	17.340	0.960
	15	4	1800.000	–
		3	655.310	0.973
		2	15.710	0.960
4	8	4	1800.000	–
		3	677.540	0.973
		2	10.090	0.960
	15	4	1800.000	–
		3	577.740	0.973
		2	20.190	0.960

Table 3 presents the training time required by the OCT method assuming the maximum number of splits being fixed. As one can notice from Table 3 even in the case of considering a maximum number of splits in the tree the OCT approach struggles to find an optimal solution for the trees with a depth of 3 and more when a maximum of 4 splits are allowed. When $D = 3$ or $D = 4$, the OCT model is solved to optimality within the predefined solution time limit only in cases when the maximum number of splits C is up to 3. It is essential to highlight that solving the OCT model with the maximum number of splits fixed might not necessarily guarantee superior solution time compared to using the complexity parameter α instead. For example, when $D = 3$ and $N_{min} = 15$, solving the OCT model using the maximum of 2 splits takes 15.71 seconds and results in the tree with an in-sample accuracy of 0.960. However, solving the OCT model with $\alpha = 0.5$ results in the decision tree with the same number of splits and takes 5.05 seconds, as shown in Table 1. Nevertheless, Table 3 suggests that if one was to consider the maximum number of splits $C = 3$ the optimal OCT tree has the potential to have significantly higher in-sample accuracy than CART. However, finding such a solution requires a longer solution time compared to the one used here. As an example, for depths of 3 and 4, the optimal OCTs have in-sample accuracy of 0.973 compared to the corresponding accuracy of 0.960 of the session tree produced by CART, as shown in Tables 3 and 1.

One should bear in mind that all the experiments were conducted considering a single data set and hence, the conclusions are primarily applicable for the cases where the training data has a similar design. The Iris data set used in the experiments has fairly little noise. For instance, we can always manually separate the points from the first class, Iris setosa, from the other classes using solely a single split. More close analysis of the trained OCT and CART tree structures suggests that this split was commonly introduced into the dataset space. It should be recognised that in situations where a similar split was not possible OCT and CART are likely to have different results. Nevertheless, for a more thorough and general analysis, the OCT model should be tested considering different data sets with diverse properties.

5 Conclusions

In this thesis, we studied the training time and the in-sample accuracy, i.e., the training accuracy, of OCT on the Iris data set using different combinations of hyper-parameters: the maximum depth of the tree, the minimum leaf size and the complexity parameter. The results were compared to the corresponding ones generated by CART which is a greedy approach commonly used in the literature for the generation of a decision tree.

Notably, the complexity parameter value had a significant impact on the training time as well as the in-sample accuracy of the resulting OCT. The OCT method was able to find optimally trained trees with large complexity parameter values for tree depths up to 4. However, when the complexity parameter value was set to zero, i.e., the complexity of the trained tree was not penalised, the training process

of a single tree was excessively time demanding and did not complete within the 30-minutes time limit. On the other hand, training overly simple-structured trees can lead to drastically weaker in-sample accuracies, as was demonstrated in numerical experiments when using the complexity parameter value 0.9. Considering the cases where OCT was able to train a tree within a given time limit (30 minutes), the improvement in in-sample accuracy compared to CART did not exceed 1%.

Numerical experiments suggest that utilising warm starts for training OCT reduces computational time. We used CART-generated optimal solutions as feasible starting values for analogous OCT problems. The training time for a tree was reduced up to a factor of 1.6. However, the speedup is unknown for cases with depths of 3 and 4 as the solver was not able to find a solution within the 30-minute time limit.

Finally, we also considered the case when OCT was trained such that the maximum number of splits was predetermined to a strict value rather than penalising complexity otherwise. This paradigm revealed the potential of optimal tree approaches further as we were able to find solutions with significantly higher in-sample accuracies compared to the normal OCT approach within the specified 30-minute time limit even when the maximum depth was 3 or 4. However, with these depths, the solver was not able to find optimal trees within the given time limit with the maximum number of splits being 4.

For a more thorough analysis, the experiments should be conducted on a diverse collection of data sets. Furthermore, in order to derive conclusions regarding the prediction quality of OCT trees we should also analyse the out-of-sample accuracy. As a logical next step for further research, similar experiments as in this thesis could be conducted considering the OCT-H method which allows multivariate splits.

References

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

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

D. Bertsimas and J. Dunn. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC, Belmont, MA, 2019.

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

R. Blanquero, E. Carrizosa, C. Molero-Río, and D. Romero Morales. Optimal randomized classification trees. *Computers Operations Research*, 132:105281, 2021.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth and Brooks, Monterey, CA, 1984.

K. Chitra and B. Subashini. Data mining techniques and its applications in banking sector. *International Journal of Emerging Technology and Advanced Engineering*, 3(8):219–226, 2013.

D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

A. Jiménez-Cordero, J. M. Morales, and S. Pineda. Warm-starting constraint generation for mixed-integer optimization: A machine learning approach. *arXiv preprint arXiv:2103.13074*, 2021.

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

J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer. Generalized and scalable optimal sparse decision trees. *arXiv preprint arXiv:2006.08690*, 2020.

M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. *Advances in neural information processing systems*, 28, 2015.

V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman. Decision trees: an overview and their use in medicine. *Journal of medical systems*, 26:445–463, 2002.

H. Verhaeghe, S. Nijssen, G. Pesant, C. G. Quimper, and P. Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25:226–250, 2020.

S. Verwer and Y. Zhang. Learning optimal classification trees using a binary linear program formulation. *In Proceedings of the AAAI conference on artificial intelligence*, 33(01):1625–1632, 2019.