Impact of model size on tree ensemble prediction accuracy and optimization time

Eetu Reijonen

School of Science

Bachelor's thesis
Espoo 1.11.2023

Supervisor

Prof. Fabricio Oliveira

Advisor

DSc (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 Eetu Reijonen

Title Impact of model size on tree ensemble prediction accuracy and optimization time

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 DSc (Tech.) Nikita Belyak

Date 1.11.2023 **Number of pages** 25 **Language** English

Abstract

Tree ensembles are a type of machine-learning model used for regression and classification tasks. The best-known types include random forests and gradient-boosted trees. These models are used in numerous fields and can have distinct advantages such as interpretability when compared to other machine-learning models. In this thesis, tree ensemble optimization is addressed, i.e., the problem of choosing an input such that the prediction of the tree ensemble is maximized or minimized. For practical applications where some of the input variables are controllable, this allows for optimal decision-making. A previously developed mixed-integer optimization formulation is used to solve the optimization problem. The focus of our research is determining the tradeoff between the predictive accuracy of the ensemble and the optimization time of the corresponding mixed-integer optimization problem when choosing the parameters for the tree ensemble, namely tree count and maximum depth of trees. This is achieved with computational experiments, which consist of training tree ensembles with different parameters on different datasets, determining the predictive accuracies of those models, and finally solving the corresponding mixed-integer problems. The testing is conducted on real-world datasets and standard hardware. Our results indicate that tree ensemble models with a maximum depth of 5 or 7 and a tree count of 200 achieve good predictive accuracies and that the optimization of the corresponding mixed-integer problems is feasible with optimization times of only around ten seconds.

Keywords mixed-integer optimization, tree ensemble models, decision trees

Aalto University
School of Science

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

Tekijä Eetu Reijonen

Työn nimi Impact of model size on tree ensemble prediction accuracy and optimization time

Koulutusohjelma Teknistieteellinen kandidaattiohjelma

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Prof. Fabricio Oliveira

Työn ohjaaja TkT Nikita Belyak

Päivämäärä 1.11.2023 **Sivumäärä** 25 **Kieli** Englanti

Tiivistelmä

Puumallit ovat koneoppimismallityyppi, jota käytetään regressio- ja luokittelutehtäviin. Tunnetuimpia puumalleja ovat satunnaismetsät ja gradienttivahvistetut puut. Puumalleja hyödynnetään monilla aloilla, ja muihin koneoppimismalleihin verrattuna niiden selkeä etu on tulkittavuus.

Tässä kandidaatintyössä keskitytään puumallin optimointiongelmaan, eli siihen, kuinka syöte tulisi valita, jotta mallin ennuste olisi maksimaalinen tai minimaalinen. Käytännön sovelluksissa, joissa syötemuuttujat ovat hallittavissa, optimointiongelman ratkaiseminen mahdollistaa optimaalisen päätöksenteon. Optimointiongelman ratkaisemiseen käytetään aiemmin kehiteltyä kokonaislukuoptimointimallia.

Tutkimuksen tavoitteena on selvittää suhde puumallin ennustustarkkuuden ja vastaavan kokonaislukuoptimointiongelman ratkaisuajan välillä valittaessa puumallin parametrejä, tarkemmin puiden lukumäärää ja maksimisyvyyttä. Tutkimus toteutettiin laskennallisina kokeina, joissa ensiksi koulutettiin puumalleja eri parametreillä ja tietoaineistoilla, minkä jälkeen määritettiin näiden mallien ennustustarkkuudet ja lopulta vastaavien optimointiongelmien ratkaisuajat. Kokeet suoritettiin todellisilla tietoaineistoilla ja tavallisella tietokoneella.

Tulokset osoittavat, että puumalleilla, joissa puiden maksimisyvyys on 5 tai 7 ja lukumäärä on 200, saavutetaan hyvä ennustustarkkuus, ja että vastaavien kokonaislukuoptimointiongelmien ratkaiseminen on mahdollista nopeasti noin kymmenessä sekunnissa.

Avainsanat kokonaislukuoptimointi, puumallit, päätöspuut

Contents

1 Introduction

Tree ensembles are a class of machine-learning models commonly used for regression and classification tasks. Some of the most widely used tree ensemble models are random forests (Breiman, 2001) and gradient-boosted trees (Friedman, 2001). These have been used successfully in numerous applications across various fields from particle physics (Lalchand, 2020) to website ranking in search engines (Cossock and Zhang, 2008).

Tree ensemble models consist of decision trees, which can be used to model complex nonlinear functions. The easy-to-understand structure of decision trees makes them interpretable and explicable, and thus suitable for areas where explicability is required of decisions, e.g., insurance and medicine. Although tree ensemble models sacrifice some of the interpretability of decision trees, their predictive accuracy is better. Nevertheless, there exist methods for converting an ensemble model into a single decision tree aiming to amend the aforementioned interpretability issue. (Sagi and Rokach, 2021; Vidal and Schiffer, 2020).

This thesis focuses on the problem of tree ensemble optimization: how to find an input that maximizes (or minimizes) the tree ensemble model output? In applications where some variables are in our control, this optimization problem becomes of great practical interest and use, since obtaining solutions can aid us in decision-making. For example, such problems appear in the context of predicting concrete quality based on the ratios of its constituents (Ni and Wang, 2000). By first training a tree model to predict the strength of concrete, we can then solve the optimization problem and discover how to make the strongest concrete by knowing the optimal values for its constituents. This technique can be utilized in an even wider context by training a tree ensemble model to approximate any nonlinear function and then optimizing the tree ensemble model, thus finding an approximation of the arguments of the maxima (or minima) of the original function that otherwise might be impossible to find analytically or heuristically.

The purpose of our research is to investigate the relationship between the tree ensemble model size (namely the number of decision trees and their maximum depth) that directly impacts the tree ensemble prediction accuracy and the computational feasibility of optimizing this tree ensemble model. We analyze the tradeoff between the prediction accuracy of the tree ensemble and the computational time required for its optimization. This investigation is motivated by the fact that for some applications one would prefer a tree ensemble with poorer prediction accuracy but faster optimization time.

In this thesis, we first give a brief review of the research combining machine learning models and mathematical optimization in the second chapter, as well as a summary of the paper by Mišić (2020) that introduced the mixed-integer optimization (MIO) formulation of a decision tree ensemble used in our experiments. Then, the third chapter details the MIO formulation and the tree ensemble model used. In the fourth chapter, the design of our experiments is described, and the MIO formulation is applied to tree ensembles trained with publicly available real-world datasets. Finally, in the fifth and sixth chapters, the results are analyzed and summarized.

2 Related work

With the increasing amount of data and the demand for its analysis, the intersection of mathematical optimization and machine learning has been actively explored over the past years. For instance, the area of mixed-integer optimization has been efficiently explored in the context of training and optimizing machine-learning models, leading to improvements in model prediction accuracy and training time. As an example, the technique of reformulation as a mixed-integer problem has been used for both training (Dua, 2010) and optimizing (Fischetti and Jo, 2018) neural networks.

Training a single decision tree using mixed-integer optimization was originally explored by Bertsimas and Dunn (2017). By converting the training procedure into solving a mixed-integer optimization problem, one can construct a globally optimal decision tree - the decision tree with the highest predictive accuracy possible given the maximum tree depth. Compared to heuristic training methods improvements up to multiple percentage points in predictive accuracy were observed.

The problem of finding an input that maximizes or minimizes the output of a trained tree ensemble model posed as an optimization problem has also been the focus of a number of studies. Early papers used heuristic search methods for finding locally optimal solutions (Martelli and Montanari, 1978), but following the development of mathematical optimization software, more efficient methods for finding globally optimal solutions have been developed. For example, Ferreira et al. (2016) developed a mixed-integer optimization approach for optimizing decision trees as a part of their work related to marketing.

Recently, Mišić (2020) introduced a different MIO formulation of the tree ensemble optimization problem which was demonstrated to perform significantly better in terms of the computational time than the one developed by Ferreira et al. (2016) making it suitable to be applied to tree ensembles trained with high-dimensional datasets. Furthermore, Mišić (2020) provided an algorithm for generating some of the constraints during the solution process, which decreases solution time considerably. The authors used publicly available datasets to analyze the performance, provide a realistic case study, and demonstrate the computational tractability of the formulation.

This formulation developed by Mišić (2020) has also been utilized in recent studies. For example, it is an integral part of a comprehensive framework developed for optimizing decision trees called ENTMOOT which includes uncertainty estimates and domain-specific knowledge built around the MIO formulation (Thebelt et al., 2021). ENTMOOT has been used at least in the context of optimizing energy systems (Thebelt et al., 2022). To the best of our knowledge, no MIO problem formulation for optimizing tree ensembles exists as of today that can be solved faster than the formulation proposed by Mišić (2020).

3 Models used

3.1 Decision trees and tree ensembles

A decision tree consists of nodes, which map the observation (independent variables) to one of the leaves. With each leaf, there is an associated prediction value, and that gives the prediction of the tree (dependent variable). Each node has two children and imposes a condition on the observation. If the condition is met, the observation is mapped to the left child, and if it is not met, the right child is chosen. This querying and mapping process on the observation starts from the root of the tree and continues until a terminal node, i.e., a leaf, is reached, ultimately providing the tree prediction. The prediction of a tree ensemble model is given as a weighted sum of the predictions of the individual decision trees.

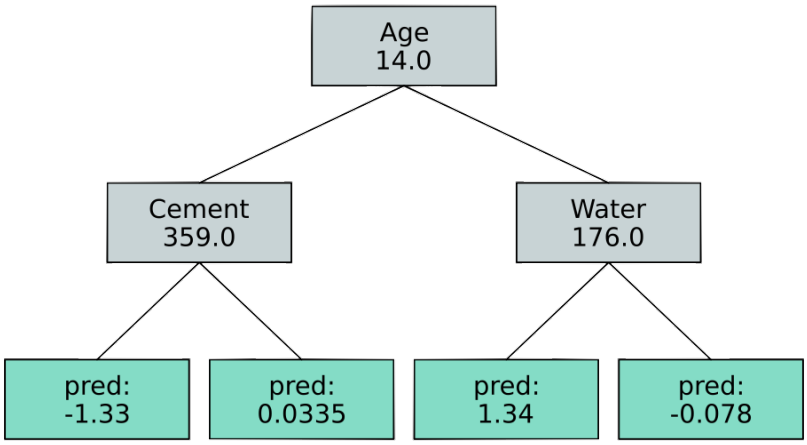

Figure 1: An example of a decision tree predicting the strength of concrete from its setting time and the amount of cement and water.

An illustrative example of a decision tree is presented in Figure 1. This tree is a part of an ensemble, and thus the seen predictions are only used to slightly change the full model prediction. Low setting time and low amount of cement seem to weaken the concrete, whereas long setting time and a moderate water amount strengthen it.

In this thesis, we focus on regression trees, i.e., decision trees that assign only numerical predictions to the leaves. A regression tree can be seen as a function $f : \mathbf{X} \to Y$ mapping a set of independent variables $(X_1, X_2, ...)$, which can be noted with a vector $\mathbf{X}$, to the dependent variable Y. At each split node, a condition of the form $X_n \leq S$, where S is the numerical value associated with the split, is imposed on $\mathbf{X}$. Split nodes can also be associated with a categorical variable, in which case the condition is of the form $X_n \in C$, where C is a subset of possible values for X_n. The prediction of an ensemble is given as $\sum_{t=1}^{T} w_i \cdot f_t(\mathbf{X})$, where T is the number of trees, w_t the weight of tree number t in the ensemble and f_t the prediction of the tree.

3.2 Gradient-boosted trees

We chose gradient-boosted trees due to their usually superior prediction accuracy over random forests, another tree ensemble model utilizing a different training process. A gradient-boosted tree model is based on two machine-learning principles: boosting and gradient descent. In general, boosting involves iteratively creating weak learners (predictors that might perform only slightly better than random chance) and combining them to create a strong learner - the final machine-learning model. Gradient descent is used to minimize the residual error, the difference between the observed value from data and the value predicted by the model, at each step. In gradient-boosted tree models, decision trees are the weak learners that are iteratively created, each minimizing the remaining prediction error of the tree ensemble.

Gradient-boosted trees were explicitly introduced by Friedman et al. (2000) and Friedman (2001), although boosting and gradient descent had been previously explored together (Breiman, 1999). Despite their early introduction, gradient-boosted trees are still commonly used in machine learning. XGBoost is a well-known implementation of gradient-boosted trees, introduced by Chen and Guestrin (2016).

Here we describe a simplified version of the training process of a gradient-boosted regression tree model. The objective of the training is to minimize the so-called regularized objective, which consists of the loss caused by the prediction error, as well as a penalty induced by the model complexity. Regression trees are added to the model one at a time, each constructed in a way to minimize the regularized objective. This iterative training process continues until a desired prediction accuracy or a certain number of trees is reached. A more detailed description of the training process can be found in Chen and Guestrin (2016), which also details how to prevent overfitting, find the optimal trees, and implement the algorithm efficiently in practice.

3.3 MIO formulation

The mixed-integer optimization formulation developed by Mišić (2020) gives a systematic way to solve the tree ensemble optimization problem. In the MIO formulation, the dependent variable Y is represented in terms of the leaves on which the solution falls, and the independent variables $\mathbf{X}$ in terms of the bins the inputs fall in. We state the MIO formulation (1) as it was introduced by the authors.

The objective of the optimization (1a) is to maximize (or minimize) the tree ensemble model prediction, which is a weighted sum of the predictions of the individual trees. The sum is calculated from the values associated with the leaves of the tree, denoted $\mathbf{leaves}(t)$. The variable $y_{t,l}$ represents whether the observation is mapped to leaf l of tree t. Constraint (1h) ensures the nonnegativity of $y_{t,l}$, and with all the other constraints, it is enforced to be a binary variable. The parameter w_t states the weight of the given tree and the parameter $p_{t,l}$ states the prediction of leaf l in tree t. In each tree, the observation can only be mapped to one leaf. This is guaranteed by constraint (1b).

Let us denote the set of numerical variables with $\mathcal{N}$ and the set of categorical variables with $\mathcal{C}$. In this thesis, we only consider numerical variables but the MIO

formulation supports categorical variables as well. Then the number of variables n is given by $n = |\mathcal{N}| + |\mathcal{C}|$. Associated with each numerical variable, there are multiple so-called split points, each indicating a condition value in a split node. For example, if a node has a condition $X_1 \leq 6$, then the number 6 would be a split point of X_1. All condition values of the split nodes, i.e., split points, are grouped in ascending order by numeric variables. Thus each numerical variable has a group of ordered split points associated with it. Only unique values are considered; two split nodes with the same condition value count as one split point. For convenience, the number of split points associated with input variable X_i is denoted by K_i.

For numerical input variables, let $x_{i,j}$ be a binary variable (guaranteed by (1g)) being 1 if numerical input variable X_i should be less than or equal to its j:th split point and 0 otherwise. Note the difference between uppercase X_i which represents an input variable and lowercase $x_{i,j}$ which is part of the MIO formulation. Constraint (1f) enforces the numerical ascending order of split points; if an input variable should be smaller than or equal to some split point, it should also be smaller than or equal to any split point greater than the aforementioned split point.

For categorical input variables, let $x_{i,j}$ be 1 if categorical input variable X_i should take a certain value indexed by j from the set of its possible values and 0 otherwise. The number of possible values for categorical variable i is denoted by K_i. Thus $j \in \{1, \ldots, K_i\}$. Each categorical variable assumes exactly one value, which is ensured by constraint (1e).

In a decision tree, each split node corresponds to a query on the observation. If the answer to the query is affirmative, the left path is chosen, and the query of the left child node is considered next. Thus all nodes to the right are inaccessible. Let $\mathbf{right}(s)$ be the set of leaves following the right branch of split s, and correspondingly, $\mathbf{left}(s)$ is the set of leaves following the left branch. The index of the input variable participating in split s is indicated by $\mathbf{V}(s)$. The set of values participating in the split query is denoted by $\mathbf{A}(s)$. For numerical variables, $\mathbf{A}(s) = \{j\}$ (index of the split point associated with the split node), and for categorical variables, $\mathbf{A}(s) \subseteq \{1, \ldots, K_{\mathbf{V}(s)}\}$.

The left and right split constraints, (1c) and (1d), ensure the structure of the decision tree. If a node query, i.e., a split condition, is true ($\sum_{j \in \mathbf{A}(s)} x_{\mathbf{V}(s),j} = 1$), constraint (1d) ensures that the values associated with all leaves to the right of the node must be zero. Comparably, constraint (1c) ensures that the left leaves are not active, i.e. the values associated with them are zero when the condition is not satisfied.

The solution of the optimization problem consists of the sparse matrices $\mathbf{x}$ and $\mathbf{y}$, which store every value of $x_{i,j}$ and $y_{t,l}$, respectively. It does not contain explicit numerical values for the optimum, but rather the indices of the split points which border the optimal values for the variables. Since decision trees are piecewise functions, there is a range for each input variable that produces equal output for the tree ensemble. This range can be extracted after the optimization, when $\mathbf{x}$ is known and the split points, i.e., the condition values, are known beforehand.

$$\max_{\mathbf{x},\mathbf{y}} \quad \sum_{t=1}^{T} \sum_{l \in \mathbf{leaves}(t)} w_t \cdot p_{t,l} \cdot y_{t,l} \tag{1a}$$

$$\text{s.t.} \quad \sum_{l \in \mathbf{leaves}(t)} y_{t,l} = 1, \qquad \forall t \in \{1, \ldots, T\} \tag{1b}$$

$$\sum_{l \in \mathbf{left}(s)} y_{t,l} \leq \sum_{j \in \mathbf{A}(s)} x_{\mathbf{V}(s),j}, \qquad \forall t \in \{1, \ldots, T\}, s \in \mathbf{splits}(t) \tag{1c}$$

$$\sum_{l \in \mathbf{right}(s)} y_{t,l} \leq 1 - \sum_{j \in \mathbf{A}(s)} x_{\mathbf{V}(s),j}, \qquad \forall t \in \{1, \ldots, T\}, s \in \mathbf{splits}(t) \tag{1d}$$

$$\sum_{j=1}^{K_i} x_{i,j} = 1, \qquad \forall i \in \mathcal{C} \tag{1e}$$

$$x_{i,j} \leq x_{i,j+1}, \qquad \forall i \in \mathcal{N}, j \in \{1, \ldots, K_i - 1\} \tag{1f}$$

$$x_{i,j} \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, K_i\} \tag{1g}$$

$$y_{t,l} \geq 0, \qquad \forall t \in \{1, \ldots, T\}, l \in \mathbf{leaves}(t) \tag{1h}$$

3.3.1 Split constraint generation algorithm

Instead of formulating the MIO problem with all of the split constraints ((1c) and (1d)), they can be introduced during the optimization process. This can shorten the solution time, since depending on what the optimization solver does, not all constraints necessarily get added. For a candidate solution of the MIO problem, only the split constraints associated with the active nodes (the nodes that lead to the output leaf) are relevant. Instead of $2^{depth-1} - 1$ constraints for all of the split nodes, only $depth - 1$ split constraints are needed to completely determine the solution in a single tree. The necessary constraints cannot be known beforehand, but it is likely that during the optimization process, a solution is found before all of the possible constraints are added. This is especially useful for deep trees, where the number of split constraints is large.

The principle of the split constraint generation algorithm is to traverse the tree from the root to the output leaf and check whether any split constraints are violated along the path. A constraint is violated if the active output leaf for the solution is on the wrong side of a given split, e.g., if the input variable X_i associated with the condition of the root node is stated by $x_{i,j}$ to be smaller than the root node split value, but the output falls to a leaf that is a right-hand-side child of the root node. The algorithm is utilized as follows. First, the MIO problem is solved without any of the split constraints, returning a candidate solution. Then violated split constraints are searched from all trees and added to the formulation, and the optimization problem is re-solved. This is repeated until no more violated constraints are found and the solver terminates with the solution.

The pseudocode of the split constraint generation algorithm is presented in Algorithm 1. This algorithm was also introduced in the paper by Mišić (2020). One can efficiently implement it with lazy constraints, a technique designed for problems

where it is beneficial or imperative for the constraints to be added during the solution process.

Algorithm 1 Split constraint generation algorithm

Require: Candidate solution $(\mathbf{x}, \mathbf{y})$ satifying constraints (1b), (1e), (1f), (1g), (1h)
  for $t \leftarrow 1$ to T **do** ▷ Check all trees
    $n \leftarrow \mathbf{root}(t)$ ▷ Start from the root node
    while $n \notin \mathbf{leaves}(t)$ **do** ▷ Traverse until reaching a leaf
      if $\sum_{j \in \mathbf{A}(n)} x_{\mathbf{V}(n),j} = 1$ **then** ▷ Split condition is true - left side chosen
        if $\sum_{l \in \mathbf{right}(n)} y_{t,l} > 0$ **then** ▷ Solution found among right leaves
          add (1d) associated with n to MIO formulation (1)
          break ▷ Check the next tree
        else
          $n \leftarrow 2 \cdot n$ ▷ Move to the left child
        end if
      else ▷ Split condition is false - right side chosen
        if $\sum_{l \in \mathbf{left}(n)} y_{t,l} > 0$ **then** ▷ Solution found among left leaves
          add (1c) associated with n to MIO formulation (1)
          break ▷ Check next tree
        else
          $n \leftarrow 2 \cdot n + 1$ ▷ Move to right child
        end if
      end if
    end while
  end for

4 Computational experiments

For our set of experiments, the objective was to examine the tradeoff between predictive accuracy (measured with the coefficient of determination R^2) and the time required to solve the optimization problem (referred to simply as the optimization time from now on) when choosing the design (number of trees and maximum tree depth) of the tree ensemble model. Thus, the task was twofold: 1) determining the tree ensemble model design required to achieve desired predictive accuracy and 2) establishing how large tree ensembles can be while still having computationally feasible MIO formulations in terms of optimization time.

4.1 Setup of the experiments

The MIO formulation was implemented using the Julia programming language (Bezanson et al., 2012) because of its efficient and user-friendly mathematical optimization framework JuMP (Lubin et al., 2023). Versions used for testing were Julia 1.9.2 and JuMP 1.12.0. Gradient-boosted tree package EvoTrees.jl (Evovest,

2023) (version 0.15.0) was used for training and creating the tree ensemble models. It utilizes the same algorithm as XGBoost, for example, but is fully implemented in Julia. For optimization of the MIO problem, Gurobi solver (Gurobi Optimization, LLC, 2023) version 10.0.2 was used. It is a robust and widely used commercial solver that provides free licenses for academic use. Gurobi.jl package (The JuMP Dev Team, 2023) (version 1.0.1) was used as the interface between the solver and Julia. The code developed for our testing is available on Github (gamma-opt, 2023).

All tests were performed on a 2016 HP laptop with a 2-core Intel Core i7 processor, 16 GB of RAM, an M.2 SSD, and Windows 10 as the operating system.

Three different datasets were used. One dataset reports on the compressive strength of concrete when the amounts of its constituents and the hardening time are known (Yeh, 2007). The two other datasets are from Ma et al. (2015). They were created for a machine learning competition held by Merck and hosted by Kaggle, where the task was to model quantitative structure–activity relationships (QSAR) for candidate drug molecules. The chosen drug design datasets are named 3A4 and OX2. They contain information on the activity caused by a molecule with certain structure groups (Ma et al., 2015). The number of variables and the number of observations of the datasets are presented in Table 1. For the concrete dataset, 75% of the observations were randomly selected for training and the rest were reserved for testing. With the molecule datasets, separate files for training and testing were provided by the authors.

Table 1: Summary of the datasets used

Dataset	No. variables	No. observations (train)	No. observations (test)
Concrete	9	772	258
OX2	5790	11151	3704
3A4	9491	37241	12338

4.2 Methodology

First, EvoTrees models were trained with each dataset and saved to a file to be able to be reused. Maximum tree depths of 3, 5, 7, 9, and 12 were tested, resulting in five EvoTrees models for each dataset. Note that in the rest of the thesis, we sometimes refer to the maximum tree depth as simply "tree depth" or "depth". Number of trees used in each model was set to 1000. The training time of every model was measured.

Because of the iterative nature of gradient-boosted trees, with a model containing 1000 trees, the behavior of a model with any lower number of trees can be matched by limiting the number of trees used in the prediction. Using this method, models with forest sizes of 50, 100, 200, 350, 500, 750, and 1000 trees could be used. Except for the tree depth and the forest size, default EvoTrees parameters were used (Evovest, 2023). The predictive qualities of the resulting 35 different models for each dataset were evaluated using the testing part of each dataset and the coefficient of determination (R^2) as a metric.

In the second part of our experiments, corresponding MIO problems were formulated from the EvoTrees models, and then solved using Gurobi. Each of the EvoTrees models was first loaded from a file, then the information of the trees was extracted, and the variables, constraints, and the objective of the MIO problem were formulated with JuMP. Then, each MIO problem was solved both with and without the split constraint generation algorithm (Algorithm 1). The split constraint generation algorithm was implemented using Gurobi callbacks through MathOptInterface (Legat et al., 2021). A time limit of two hours was imposed on the solving of each MIO problem. If completed, the optimal objective value was saved, and in case of a termination due to time-out, the optimality gap was saved. Gurobi was used with default parameters, except for the presolve that was set to 0. For more information about the Gurobi parameters see Gurobi documentation (Gurobi Optimization, LLC, 2023). The time taken by solving each MIO problem was measured using Julia's *@elapsed*-macro. Additionally, the number of split points, the number of leaves, and the number of split constraints were recorded.

Due to the nature of Julia's compiler, the very first compilation adds additional time to the execution, which might skew time comparison results, especially for scripts or programs where the execution takes less than a second. For this reason, when the EvoTrees training was conducted, a starting model of depth 2 was trained before the actual tree ensembles used in our experiments. Similarly, during optimization, an MIO problem formulated for a tree ensemble model with ten trees was optimized first.

5 Results

In this section, both the tree ensemble models' predictive accuracy and their corresponding MIO problem optimization time are presented. First, the numerical results are discussed, and in section 5.4 they are analyzed and the research questions are answered. All tests were executed as described in section 4.2 except that an EvoTrees model for the 3A4 dataset could not be trained at a depth of 12 because this caused Julia to crash for an unknown reason each time despite multiple attempts. Also, if an MIO problem solution time reached the limit of 2 hours, no further tests were conducted for MIO problems of tree ensembles with the same depth but a larger number of trees since they would also time out as the optimization problem only becomes more difficult. The full test results are available on Github (gamma-opt, 2023).

5.1 Predictive accuracy

To illustrate the predictive accuracies of the different EvoTrees models, we plotted the accuracies on graphs where the horizontal axis represents the number of trees and the vertical axis represents the coefficient of determination. We use different colors to indicate EvoTrees models with different depths. Individual data points are marked with a cross symbol.

The results for the concrete dataset are presented in Figure 2. It can be seen from the graph that the best-performing model was clearly the one with maximum depth set to 5. The model with depth 3 shows clear under-fitting at least for low numbers of trees and models with greater depth than 5 overfit the data. EvoTrees models with a high number of trees also overfit which is manifested in the test data prediction accuracy not improving with models having more than 200 trees. For this dataset, a very high coefficient of determination of around 0.93 was achieved with an EvoTrees model having 200 trees and a maximum depth of 5.

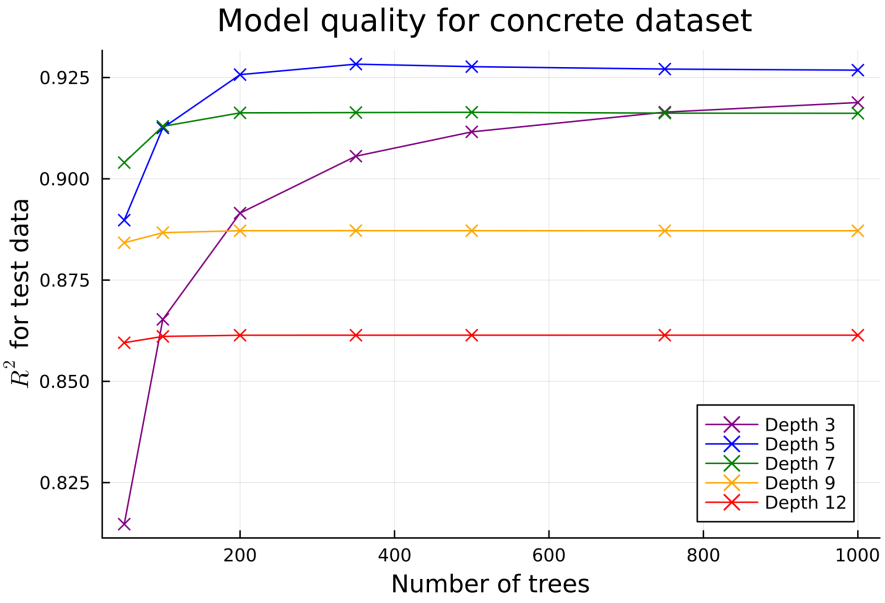

Figure 2: The predictive accuracies of the EvoTrees models for the concrete dataset

The R^2 scores of the EvoTrees models trained with the OX2 dataset are presented in Figure 3. Being considerably higher-dimensional and having a higher number of observations, the OX2 dataset required an EvoTrees model with a maximum depth of 7 for the best predictive accuracy. Still, tree ensembles with a higher maximum depth than that overfit and produced increasingly worse R^2 scores. The highest R^2 achieved was considerably lower than that of the concrete dataset at around 0.58. Even with the OX2 dataset, the improvements in R^2 scores plateaued at 200 trees for depth 7 which was the maximum depth value in the model producing the best R^2 scores.

Figure 4 shows the coefficients of determination of the EvoTrees models trained with the 3A4 dataset which is the dataset with the largest number of observations. Thus, to achieve the best predictive accuracy, the largest EvoTrees models in our experiments in terms of the maximum tree depth and the forest size were beneficial. When looking at the data of the EvoTrees models with a certain maximum depth, improvements in R^2 were observed by increasing the number of trees up to 350 trees. Models with a maximum depth of 3 or 5 showed improvements by increasing the number of trees even further. However, EvoTrees models with a maximum depth of 7 were able to produce the highest R^2 scores. With 350 trees a model of this

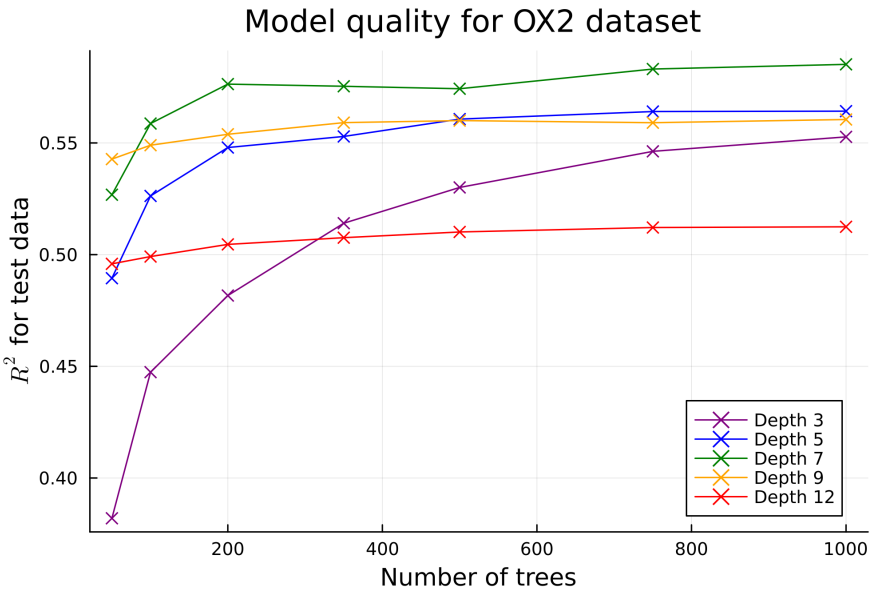

Figure 3: The predictive accuracies of the EvoTrees models for the OX2 dataset

depth produced an R^2 of 0.49 and by increasing the number of trees no signicant improvements (more than 0.01) could be gained. For reasons mentioned previously, we were not able to train an EvoTrees model with a depth of 12, but based on the results we have obtained for the other datasets it likely would have produced worse R^2 scores than the model with a depth of 9.

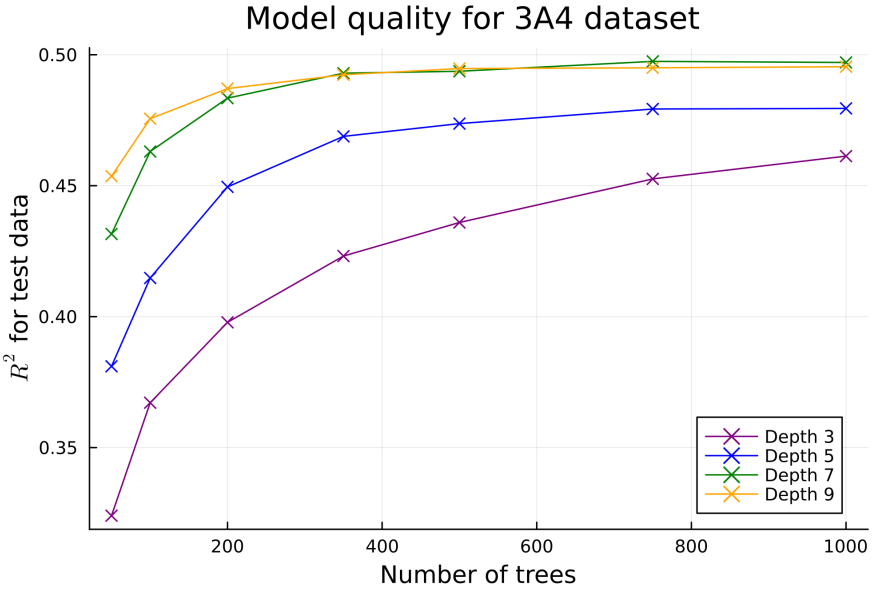

Figure 4: The predictive accuracies of the EvoTrees models for the 3A4 dataset

5.2 Training time

By observing the logarithmic scale plot in Figure 5, it can be seen that the increase in the training time for the EvoTrees models was exponential with increasing depth. The training time of the EvoTrees models with the larger datasets 3A4 and OX2 was considerably higher ranging from multiple minutes to multiple hours compared to the training time of the models with the concrete dataset. For every model trained with the concrete dataset, the training finished in under ten seconds. It should be noted that each EvoTrees model contained 1000 trees - a number demonstrated excessive by the predictive accuracy testing. Due to the nature of the iterative training process of gradient-boosted trees, a reduction in the number of trees should yield linear improvements in training time, e.g., the training of a 200-tree model taking only a fifth of the time.

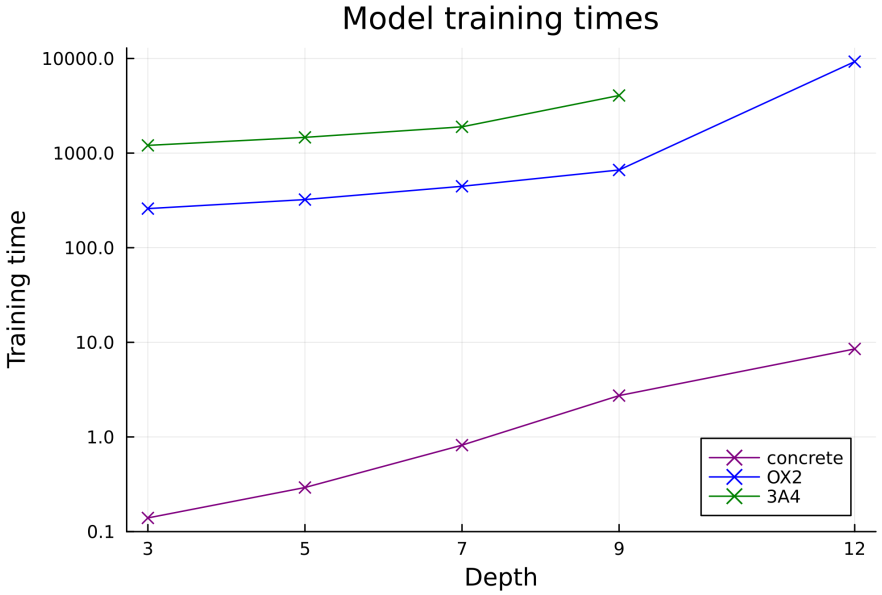

Figure 5: Training time for the EvoTrees models on a logarithmic-scale plot

5.3 Optimization time

In this section, Figures 6, 7, and 8 are discussed. They present only the optimization time (time taken by the solver). The numbers do not include the MIO problem formulation time and the time needed to extract the EvoTrees model information such as the split points and the leaf prediction values. However, the experiments indicated that the time required to extract the information and to formulate the MIO problem is insignificant compared to the optimization time. Hereinafter, when referring to optimizing an MIO problem corresponding to an EvoTrees model, we simply state we are optimizing a model. This simplification is used to make the text more concise.

For the concrete dataset, the increase in optimization time along with the increase in the number of trees both with and without the split constraint generation algorithm

is presented in Figure 6. In the bottom graph, the split constraint generation algorithm is used, and in the top graph, it is not. The corresponding MIO problem (1) of every EvoTrees model trained with the concrete dataset was solved to optimality within 2 hours. An exponential increase in optimization time with the number of trees can be observed. The EvoTrees models producing the best R^2 scores (models with depth 5) could be optimized in under 20 seconds for every number of trees using the split constraint generation algorithm (1) and in under 3 minutes without it. In the previous section, a depth beyond 5 and a tree number above 200 were seen to provide no improvements in R^2. The model with these parameters (depth 5 and 200 trees) was optimized in under a second.

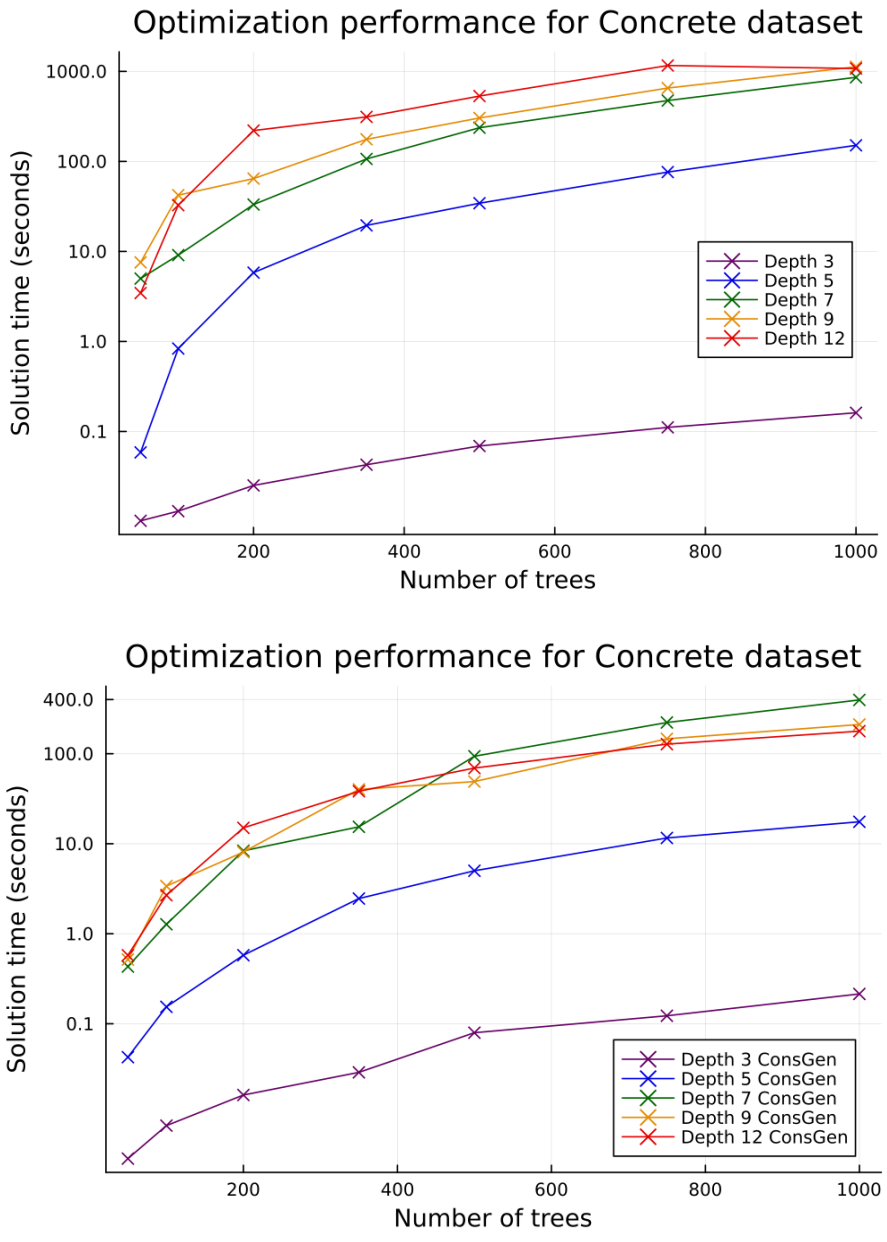

Figure 6: Optimization time for the concrete dataset

The optimization task was considerably more difficult for the drug design datasets since they contain thousands of variables compared to less than ten variables in the concrete dataset. This means that the optimization time was a lot longer for the drug models. The optimization results for the OX2 dataset are presented in Figure 7. Optimization time both with and without the split constraint generation algorithm is presented. The algorithm time is marked with a lighter color and can be seen mostly below the normal time. With the OX2 dataset, models with depth 5 could not be optimized within the two-hour time limit if they had over 500 trees. For the model with depth 7, this tree number was 200, and models with depths 9 and 12 could have 100 trees before timing out. Still, the model with a depth of 7 and 200 trees could be optimized in only 11 seconds. There were no EvoTrees models that had a meaningfully better prediction accuracy than this model as seen in section 5.1.

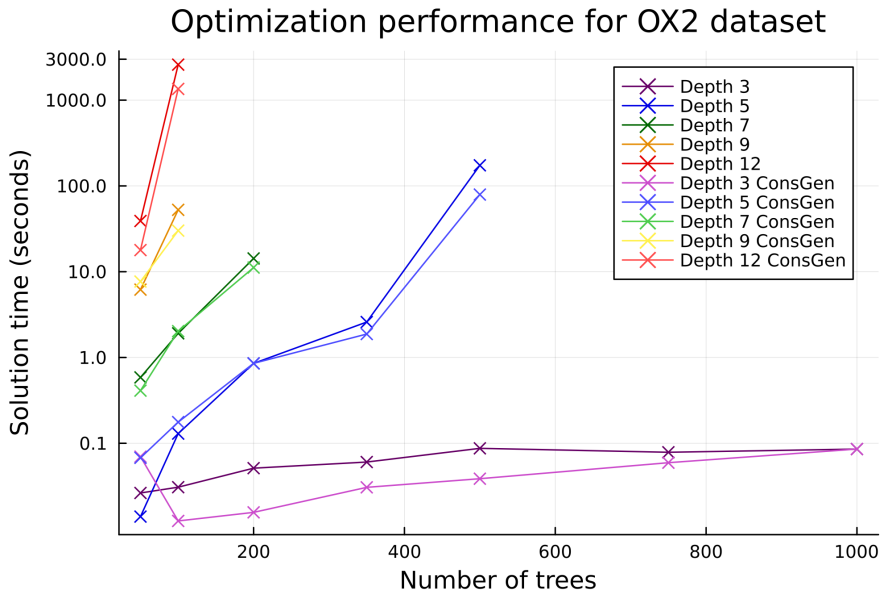

Figure 7: Optimization time for the OX2 dataset.

Finally, Figure 8 shows the optimization time of the models trained with the 3A4 dataset. Similar markings to the previous Figure 7 are used here. Compared to the OX2 models, models with more trees could be optimized within the two-hour-time limit. Again, the model with parameters demonstrated to produce the best predictive accuracy in the previous section 5.1 was optimized successfully. The optimization time for this model of depth 7 and 350 trees was under 20 minutes. Considerable time reductions can be gained by limiting the EvoTrees model complexity: a model with depth 7 and 200 trees could be optimized in about 10 seconds.

5.3.1 Split constraint generation algorithm

Figure 6 shows the optimization time with and without the split constraint generation algorithm (1) of the models trained with the concrete dataset. There are five different depths and seven different tree counts, resulting in 35 models. In optimizing 32 out

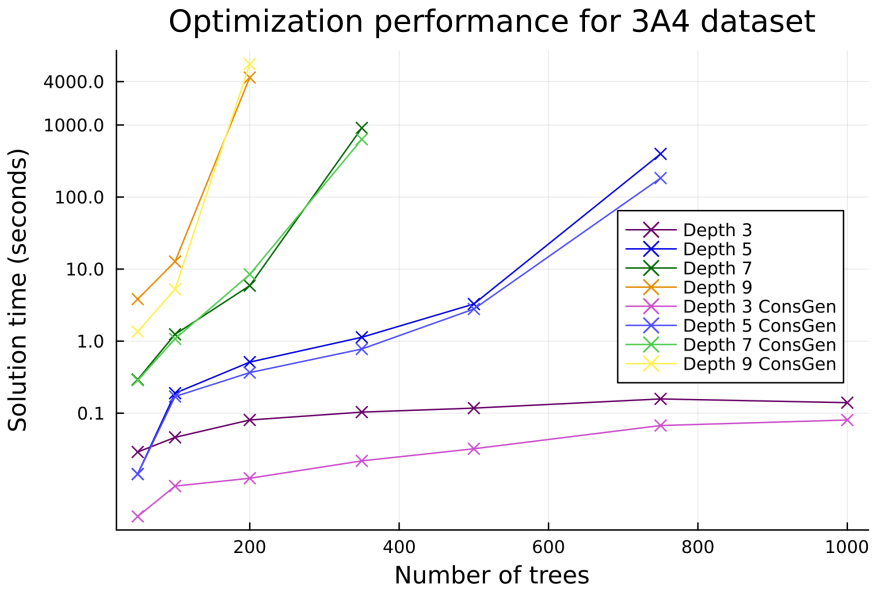

Figure 8: Optimization time for the 3A4 dataset.

of these 35 models, using the split constraint generation algorithm shortened the time compared to solving the MIO problem without the algorithm. The average reduction in optimization time was 173 seconds. Using the algorithm was slower only for models for which optimization required less than a second, and the time difference between optimization with and without the algoritm in these cases was never more than a second.

In Figure 7, the corresponding time differences are depicted for models trained with the OX2 dataset. Here only 19 different models could be optimized within the two-hour time limit. When using the split constraint generation algorithm in optimizing these models, 12 out of 19 times a reduction in optimization time was observed with an average of 72 seconds. Again, the optimization time with the algorithm never exceeded the optimization time without the algorithm by more than two seconds.

It can be seen from Figure 8 that when optimizing the models trained with the largest dataset 3A4, using the split constraint generation algorithm created on average a 97-second time reduction and was faster with 18 out of the 20 models that could be optimized within two hours. In one of the two cases where using the split constraint generation algorithm was slower than not using it, the difference was insignificant (less than three seconds), but the optimization of the model with a depth of 9 and a tree count of 200 took about 41 minutes longer using the algorithm. This is the only instance in our experiments where the optimization time was significantly increased by using the algorithm.

5.4 Conclusions

For datasets with ten or fewer variables, such as the concrete dataset, our results indicate that a gradient-boosted tree model with a maximum depth of 5 and a tree count of 200 achieves practically the highest predictive accuracy attainable with a model of any depth and number of trees. For datasets with thousands of variables, such as the drug design datasets, a model with a depth of 7 and a tree count between 200 and 350 is required to achieve the best attainable accuracy. Thus, it can be concluded that relatively small gradient-boosted tree models, compared to all sizes we tested, are adequate for achieving good predictive accuracy. Furthermore, models with a higher maximum depth and more trees predicted less accurately at worst or the improvements in accuracy were marginal at best.

With the gradient-boosted tree model for the concrete dataset, a very high coefficient of determination (R^2) of 0.93 could be achieved. This score indicates that the model explained the variance in the data to a high degree and could be used relatively confidently to predict the strength of concrete from unobserved data. For the tree models trained with the 3A4 and OX2 datasets, their R^2 scores were much worse at 0.49 and 0.58, respectively. These numbers seem low but one can compare them to the results of the machine learning competition where the OX2 and 3A4 datasets were used in addition to 13 similar datasets. The competitors were tasked with developing a model to achieve the highest predictive accuracy. The result for each of the teams was the average testing R^2 for the 15 datasets. In the competition, the winning team achieved an R^2 of 0.49 (Kaggle, 2012). If one assumes that our EvoTrees models would perform similarily as with the OX2 and 3A4 datasets accross the 15 datasets, our R^2 scores of 0.49 and 0.58 are comparable to the best achieved in the competition.

The optimization of the aforementioned tree ensemble models with depths of 5 or 7 and 200 to 350 trees was computationally feasible using the MIO formulation and the split constraint generation algorithm. For each of the datasets, a model with the best or near-best predictive accuracy could be optimized in seconds. The optimization of larger tree models does not seem as feasible, however. Although all of the models for the concrete dataset could be optimized within half an hour, the models trained with the drug design datasets which had more variables and observations could not be optimized within two hours. Since the corresponding MIO problem solution time seems to grow exponentially with the number of variables in the dataset used to train the tree model, the optimization of the model trained with the OX2 dataset with a depth of 12 and 1000 trees could take much longer than two hours. However, as the predictive accuracy experiments demonstrated, a model of this size is not necessary to achieve the highest maximum attainable accuracy even for a dataset with thousands of variables.

The split constraint generation algorithm reduced optimization time in almost every case. For the models trained with the concrete dataset, the reductions in optimization time were multiple and in some cases even tenfold. Smaller time reductions were seen with the models trained with the drug datasets but many of them still significant. Despite the single observation where the optimization time

was significantly increased, the use of the split constraint generation algorithm is demonstrated to be an efficient technique with this MIO formulation.

As mentioned previously, the practically highest predictive accuracy attainable with tree models of any depth and number of trees could be achieved with a model of depth of 5 or 7 and 200 trees in our experiments. Choosing a model with fewer trees or not as much depth than this is not beneficial since the optimization time of the aforementioned models is already tens of seconds at most and the prediction accuracy of a model smaller than this rapidly decreases as the number of trees and the depth are decreased. Choosing larger models is not beneficial either since they predict only marginally more accurately at best and the tradeoff in increasing optimization time is very costly based on our numerical results.

5.5 Limitations

Despite the insights provided by our study, there are still some limitations in the numerical experiments that one should take into account. The most significant of them is the limited variation in the datasets used. The concrete dataset had less than ten variables while the drug design datasets had thousands and therefore our results cannot necessarily be extrapolated for datasets that have tens or hundreds of variables. Furthermore, using data from other fields would provide more variability. As an example for why variation is needed, a conclusion that tree ensembles are almost fully accurate predictors, drawn from the concrete dataset results, would fail with the drug design datasets.

In addition to real-world datasets, datasets generated with optimization testing functions, i.e., functions with known properties such as the shape and the extrema, could have been used in the experiments. As these functions can be non-continuous and have multiple extrema, both modelling them with tree ensembles and optimizing the resulting tree ensemble models could prove difficult and provide more insight into the limitations of this optimization method.

Another limitation of our study was using only gradient-boosted trees. We did not consider random forests, for example, which is another type of tree ensemble model. A comparison could have been useful in determining with which type of tree ensemble one can produce models with fewer trees and a shallower depth while having the same predictive accuracy. The tree ensemble type with which smaller models suffice would be more suitable for the optimization method and the MIO formulation discussed in this paper since the MIO formulations of smaller tree ensembles contain fewer parameters and thus are faster to solve.

Lastly, the optimization of each of the models was performed only once. Conducting the experiment multiple times and taking the time average would have added more certainty to the results since in a standard computer, the number of processes and their resource use is constantly changing, so the amount of resources allocated to the optimization task by the operating system could be slightly different each time. Also, with more powerful hardware, we might have been able to optimize some of the larger models within two hours. Because in many commercial and academic applications computational power is not a significant limitation, having

more information about the optimization time of larger tree ensembles would be useful.

6 Summary

This thesis verified the work of Mišić (2020) and contributed to the field of machine learning and mathematical optimization. We were able to repeat the results of the authors in demonstrating the computational feasibility of tree ensemble optimization. In addition, we expanded on the experiments of the original authors by analyzing the tradeoff between prediction accuracy and optimization time when choosing the tree ensemble model size, i.e., the number of trees and the maximum tree depth. Also, we used gradient-boosted trees whereas Mišić (2020) used random forests.

With gradient-boosted tree models having depths of 5 or 7 and a forest size of 200, high predictive accuracies relative to the datasets were achieved and the optimization of these models could be completed in ten seconds or less even without powerful hardware. We found that models with a higher depth or more trees did not have a significantly better predictive accuracy and required exponentially more time to optimize as the size increased.

Further research is needed comparing the optimization performance of different types of tree ensembles as well as using more datasets from different fields. Experiments with testing functions for optimizations could also be conducted. The use of more powerful hardware is also worth exploring.

References

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

Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

Leo Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7): 1493–1517, 1999.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

David Cossock and Tong Zhang. Statistical analysis of bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

Vivek Dua. A mixed-integer programming approach for optimal configuration of artificial neural networks. *Chemical Engineering Research and Design*, 88(1):55–60, 2010.

Evovest. Models EvoTreeRegressor. https://evovest.github.io/EvoTrees.jl/stable/models/, 2023. Accessed: 2023-09-18.

Kris Johnson Ferreira, Bin Hong Alex Lee, and David Simchi-Levi. Analytics for an online retailer: Demand forecasting and price optimization. *Manufacturing & service operations management*, 18(1):69–88, 2016.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

gamma-opt. ML_as_MO: Machine Learning as an Optimization Problem. GitHub repository, 2023. URL https://github.com/gamma-opt/ML_as_MO/.

Gurobi Optimization, LLC. Gurobi Optimizer. https://www.gurobi.com/documentation/, 2023. Accessed: 2023-09-18.

Kaggle. Merck Molecular Activity Challenge. https://www.kaggle.com/competitions/MerckActivity/leaderboard, 2012. Accessed: 2023-09-18.

Vidhi Lalchand. Extracting more from boosted decision trees: A high energy physics case study. *arXiv preprint arXiv:2001.06033*, 2020.

Benoît Legat, Oscar Dowson, Joaquim Dias Garcia, and Miles Lubin. MathOptInterface: a data structure for mathematical optimization problems. *INFORMS Journal on Computing*, 34(2):672–689, 2021. doi: 10.1287/ijoc.2021.1067.

Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. Jump 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 2023. In press.

Junshui Ma, Robert P Sheridan, Andy Liaw, George E Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55(2):263–274, 2015.

Alberto Martelli and Ugo Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039, 1978.

Velibor V Mišić. Optimization of tree ensembles. *Operations Research*, 68(5): 1605–1624, 2020.

Hong-Guang Ni and Ji-Zong Wang. Prediction of compressive strength of concrete by neural networks. *Cement and Concrete Research*, 30(8):1245–1250, 2000.

Omer Sagi and Lior Rokach. Approximating xgboost with an interpretable decision tree. *Information Sciences*, 572:522–542, 2021.

The JuMP Dev Team. Gurobi.jl: Julia interface to gurobi, 2023. URL https://github.com/jump-dev/Gurobi.jl.

Alexander Thebelt, Jan Kronqvist, Miten Mistry, Robert M Lee, Nathan Sudermann-Merx, and Ruth Misener. Entmoot: a framework for optimization over ensemble tree models. *Computers & Chemical Engineering*, 151:107343, 2021.

Alexander Thebelt, Calvin Tsay, Robert M Lee, Nathan Sudermann-Merx, David Walz, Tom Tranter, and Ruth Misener. Multi-objective constrained optimization for energy applications via tree ensembles. *Applied Energy*, 306:118061, 2022.

Thibaut Vidal and Maximilian Schiffer. Born-again tree ensembles. In *International conference on machine learning*, pages 9743–9753. PMLR, 2020.

I-Cheng Yeh. Concrete Compressive Strength. UCI Machine Learning Repository, 2007. DOI: https://doi.org/10.24432/C5PK67.