Building Random Forests with Optimal Decision Trees

Jord Molhoek

Delft University of Technology

June 25, 2021

Abstract

Decision trees are most often made using the heuristic that a series of locally optimal decisions yields a good final decision tree. Optimal decision trees omit this heuristic and exhaustively search - with many optimization techniques - for the best possible tree. In addition, training an ensemble of decision trees with some randomness has proven to outperform a single decision tree. This technique is called random forests.

This research brings the techniques of optimal decision trees and random forests together to construct Forests of Optimal Trees. The two most important categories of random forest generation techniques are tree-generation randomization and input-data randomization. Whereas the first is not directly applicable with optimal decision trees as that would disqualify every guarantee of optimality, the latter technique is compatible. In terms of accuracy, Forests of Optimal Trees outperform the heuristic random forests in some cases but are inferior in other cases. This difference is data-dependent. The main disadvantage of Forests of Optimal Trees is that the generation of these forests can be a few orders of magnitude slower than the heuristic forests. Nonetheless, in scenarios where a small gain in classification accuracy can have important advantages, and the cost of the time and computation power are worth it, Forests of Optimal Trees can be useful classifiers.

1 Introduction

1.1 Background and Motivation

Because of their simplicity, decision trees are widely used in machine learning. The most well-known algorithms for building decision trees from a given dataset, such as ID3 [1], C4.5 [2] and CaRT [3], are greedy algorithms. They locally find the best possible split. However, this does not guarantee that the resulting decision tree is the best tree possible. Perhaps by making a sub-optimal choice locally, the overall resulting tree could perform much better in terms of accuracy. This gave rise to the field of optimal decision trees. The problem of constructing an optimal decision tree is proven to be NP-complete [4]. A brute-force exhaustive search of all possible decision trees is infeasible for most practical situations. Fortunately, a recent advancement in this field is the MurTree [5], making use of dynamic programming. It has been demonstrated that the MurTree algorithm can have great performance benefits.

Nonetheless, the well-known problem of *overfitting* is a problem that cannot be ignored for heuristic nor optimal decision trees. On the one hand, too few nodes might limit the performance of a decision tree. On the other hand, too many nodes might result in poor generalizability to unseen data, because the tree can start to model the noise or irrelevant patterns in the training data. Fortunately, algorithms exist that prune the tree, in order to make it more generalizable to unseen data. However, these algorithms also rely on heuristics.

Random forests are an ensemble of decision trees that are generated by incorporating some randomness in each tree. These trees can be generated by randomizing some of the input data or by randomizing the tree-generation procedure slightly. The final classification is determined by a majority vote of these trees. As mentioned by Breiman [6, p. 29]:

> *Random forests are an effective tool in prediction. Because of the Law of Large Numbers they do not overfit. Injecting the right kind of randomness makes them accurate classifiers and regressors.*

Hence, random forests have proven to reduce the overfitting of heuristic decision trees before. Furthermore, because optimal decision trees may still overfit or underfit, this studies combines the techniques of optimal decision trees and random forests.

1.2 Research Questions

The main research question is: *Can optimal decision trees be used to produce a better forest, compared to the forest obtained using heuristic algorithms?* The question is interesting because optimal decision trees are a strategy to improve the performance of a decision tree, and so are random forests [6]. A combination of these two strategies has the potential to be an even better-performing machine learning algorithm. This main question is split up into the following sub-questions:

1. What are ways to construct random forests, making use of optimal decision trees?
2. How do the newly constructed Forests of Optimal Trees compare to heuristic random forests and to a single optimal decision tree in terms of accuracy on unseen data?
3. Is the runtime for the newly constructed random forest still feasible for practical use cases (in the order of seconds or minutes)?

To start, Section 2 provides an overview of some related literature. Then, Section 3 will describe some preliminaries in more technical details. Afterwards, in order to answer the first sub-question, the compatibility of random forests and optimal decision trees will be discussed in section 4. This section also describes what techniques are used to generate the Forests of Optimal Trees. In order to answer sub-question 2, the performance of the forest will be analysed in section 5. The final sub-question is discussed in section 6.

2 Related Work

Although no previous research on Optimal Forests or Forests of Optimal Trees could be found, a lot of work is already done in the fields of heuristic decision trees, optimal decision trees and random forests. The most important related work is briefly discussed here. A more detailed overview of the relevant technical details is given in section 3.

The most well-known algorithms to construct a heuristic decision tree from a given dataset are proposed by [1], [2] and [3]. These algorithms all use one important heuristic; they locally find the best split and use that split as a node. This makes the algorithms relatively efficient, and the resulting decision tree works fine for many use cases.

The problem with using this heuristic is that a series of locally optimal decisions does not guarantee that the resulting tree is optimal. There exist many algorithms that generate optimal decision trees. One such algorithm is DL8 [7], which was later improved to DL8.5 [8]. Most importantly, the MurTree was introduced by [5], which makes use of many optimization techniques and applies dynamic programming. This research focuses on combining MurTrees with random forests.

There exist many different ways of building random forests, and a detailed overview is given in section 3.4. Breiman introduced the idea of bagging predictors in [9], where an ensemble classifier is created by sampling N items from a dataset of size N with replacement. Consequently, classifiers are trained on those bootstraps. Later, he published [6], which in literature is often referred to as the random forest algorithm, even though many such algorithms exist. Another important technique in the field of random forests is the random subspace method [10], which is similar to bagging. The difference is that the random subspace method keeps all N items and samples their features. This technique is also known as *feature bagging*.

3 Preliminaries

Classification problems are well-known in the field of machine learning. Because many formal descriptions exist, all with slightly different notation and terminology, an overview of the terminology and notation used in this paper will be given here. Because this studies builds on top of the existing MurTree, most of the formal notation is synonymous with that of [5, p. 4-6]. Furthermore, existing machine learning strategies to solve this problem are described. The described strategies are heuristic decision trees, the MurTree, random forests and boosting.

3.1 Description of the Classification Problem

For the classification problem, any relevant real-world object needs to be abstracted into a *feature vector*. Intuitively, this can be a row in any table; for example, a table with data about students or patients, where one row represents one person. Let $\mathcal{F}$ be the set of features in the vector. A feature represents a characteristic or a piece of information of the corresponding object. Therefore, the i-th feature of any two feature vectors corresponds to the exact same characteristic. For this studies, it is assumed that there are only binary features, evaluating to $True$ (1) or $False$ (0). If an object has non-binary features, such as continuous values, these need to be binarised beforehand; for example by choosing a threshold value. The i-th feature of a feature vector $\mathbf{fv}$ is denoted as $f_i \in \mathbf{fv}$, given that this characteristic is $True$. Likewise, if this characteristic is $False$, it is denoted as $\overline{f_i} \in \mathbf{fv}$. In the dataset that is used to solve the *supervised learning problem*, each feature vector has a corresponding target class, which is also assumed to be binary. If this is not the case, it can be solved by applying a *one-versus-all* approach. The dataset is denoted by $\mathcal{D}$. $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$, where $\mathcal{D}^+$ denotes the instances where the target class is $True$ and $\mathcal{D}^-$ denotes the instances where the target class is $False$.

3.2 Heuristic Decision Trees

Decision trees are a popular machine learning approach that can be used to solve classification problems, most often created by making use of heuristics. A simple example of a decision tree is given in figure 1. Formally, a decision tree is a binary tree that maps a given feature vector **fv** to a class, starting from the root node. For a good decision tree, the number of *misclassifications* should be low. The tree is composed of *classification nodes* and *predicate nodes*. A classification node is a leaf node with a binary value v. Any **fv** that ends up in this leaf node is assigned to the class v. Predicate nodes are non-leaf nodes with an integer value i, where $i \in [0, |\mathcal{F}|\rangle$. For an **fv** that arrives at a predicate node, if $f_i \in$ **fv** the path is continued to the right child of the predicate node. Likewise, if $\overline{f_i} \in$ **fv** the path is continued to the left child. The exact structure of the tree and all the values v and i are learned by finding patterns in $\mathcal{D}$ [5]. The definition of *depth* can vary in the existing literature. In this studies, the depth d is defined as the maximum number of predicate nodes until a classification node is reached. I.e. classification nodes do not count towards the depth.

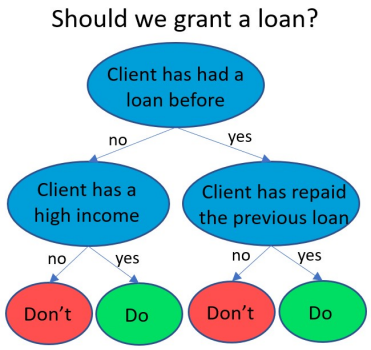

Figure 1: A simple decision tree

In general, a decision tree does not have to be a binary tree. A split can also be made on categorical variables with more than two categories. Also, the continuous features do not need to be binarised. Multiple thresholds can be used and the values of these thresholds do not need to be determined beforehand. However, because the most recent version of the MurTree [5] is limited to binary decision trees and binary datasets, so is this studies.

There are multiple algorithms that construct decision trees with heuristics, such as ID3 [1], C4.5 [2] and CaRT [3]. Essentially, all these algorithms work the same. At the root, each feature is tried as a candidate for a split on that feature. The 'best split' f_i is selected, and the left and right child trees of the root are calculated recursively. There are a variety of metrics that decide which split is the best, but the *Gini Impurity* [3] and the *Gain Ratio* [1], [2] are the most popular metrics.

3.3 MurTree

The current state-of-the-art for calculating an optimal decision tree is the MurTree [5] algorithm. The input of the algorithm is a dataset $\mathcal{D}$, a maximum depth d and a maximum number of predicate nodes n. Internally the MurTree algorithm makes use of many optimization techniques. Two of the most important optimization techniques are a specialized algorithm that calculates an optimal decision tree of depth-two, as well as memoization [5].

The specialised algorithm to calculate an optimal decision tree of depth two calculates exactly that, within a single pass over the data using $|\mathcal{F}|^2$ frequency counters. These counters count the pairwise occurrences of all pairs of features. With this information at hand, the optimal depth-two tree is calculated in $\mathcal{O}(|\mathcal{F}|^2)$. Since $|\mathcal{D}|$ is often much larger than $|\mathcal{F}|$, this algorithm is a huge speedup compared to other techniques.

Memoization is done by using a cache. Once an optimal sub-tree is calculated, this tree is stored in a cache. This way, if later on in the calculation this subtree is needed again, it does not need to be recalculated, but it can simply be taken from the cache in $\mathcal{O}(1)$. The cache

maps a branch to an optimal tree, where a branch is a set of features that are considered so far. For example, if the root considers feature f_i, and the left subtree is needed, the branch is denoted as $\{\overline{f_i}\}$. An important insight here is that the order of these features does not matter; this is what provides the speedup. If the optimal tree is calculated for a certain branch, for any permutation of that branch the calculation is omitted and the result is directly taken from the cache. To keep the cache lightweight, it does not store an entire tree for each branch. Instead, only the predicate of the root node of the corresponding optimal tree is stored. The optimal tree can easily be recovered by querying the cache again with a branch augmented with the predicate.

The original MurTree algorithm applies more optimizations. Examples are *incremental computation* and *similarity-based bounding*. These techniques are not implemented for this studies. The technical details are described in-depth in [5].

3.4 Random Forests

A random forest is an ensemble of decision trees, where each tree is trained with some randomness. This technique loses the simplicity and interpretability that makes decision trees popular, but the accuracy is often significantly better. Trees in a random forest are typically grown as deep as possible, without any pruning [6], [11].

Breiman [6] has proven that random forests converge to a minimum generalization error as the number of trees in the forest increases. This means that at some point, adding more trees does not make the ensemble classifier better. Breiman also introduced the concepts of *strength* s and *correlation* $\overline{\rho}$. Strength is based on "the extent to which the average number of votes for the right class exceeds the average vote for any other class" [11, p.4]. In order to calculate the correlation, the pairwise correlations of each pair of decision trees in the forest need to be averaged [11]. A good random forest minimizes the ratio $\frac{\overline{\rho}}{s^2}$, but how this optimization can be done is unknown [6].

Sections 3.4.1, 3.4.2 and 3.4.3 discuss different ways to build the trees in these forests, because many different ways of incorporating this randomness exist. Section 3.4.4 explains how all these trees can be combined to make the ensemble classifier. Finally, section 3.4.5 describes a metric to estimate the correlation.

3.4.1 Bagging

A well-known technique to create an ensemble classifier is *bagging*. Bagging is an acronym for ***B**ootstrap **Agg**regat**ing*** [9]. First many bootstraps of a training dataset are created. Then on each of these bootstraps, a single classifier is trained. Classifiers in the ensemble finally vote in order to classify a new instance. Different techniques for vote aggregation are explained in section 3.4.4.

A bootstrap of a dataset $\mathcal{D}$ of size $|\mathcal{D}|$ is a dataset of the same size that is randomly sampled from $\mathcal{D}$ with replacement. This means that some entries in the dataset occur more than once in the bootstrap, and that approximately 37% of the entries are not included in the bootstrap [9]. Technically, the bootstrap does not have to be of size $|\mathcal{D}|$, but it can also be bigger or smaller. However, it was found that a bootstrap of size $|\mathcal{D}|$ works well [9].

There are two important advantages of bagging. The first is that bagging performs well when there is noise in the dataset [6], [12]. The second advantage is that the *out-of-bag data* can be used as a test set, containing unseen instances, in order to analyse the performance of a tree in the forest.

3.4.2 Random Subspace Method

A technique comparable to bagging is the *random subspace method*, as introduced by [10]. This method is also known as feature bagging. Whereas bagging takes a sample of size $|\mathcal{D}|$ from $\mathcal{D}$ with replacement, the random subspace method takes all the items from $\mathcal{D}$, but only takes into account a subset of size p of the features $\mathcal{F}$ (without replacement). It was found out empirically that size $p = \frac{|\mathcal{F}|}{2}$ works best [10]. This method has proven to outperform bagging in many cases, especially when the data has a large number of samples and features. Where many other classifiers are tormented by the *curse of dimensionality*, the random subspace method takes advantage of high dimensionality. For datasets with relatively few features, the random subspace method works best if the dataset is first augmented by a number of extra features that are sums, products, boolean combinations or linear combinations of existing features [10].

3.4.3 Tree-generation randomization

In contrast to randomizing the input data, techniques to randomize the tree-generation procedure also exist. *Random split selection* is an example of such a technique. Instead of locally picking the best possible split, random split selection calculates the n best possible splits and uniformly picks one of them [12]. Another example is *random feature selection*. Instead of calculating the best split or the n best splits, random feature selection first randomly generates a subset of $\mathcal{F}$ in each node. After that, the best split from this subset is used for the predicate node [6]. Many variations of these techniques exist. For example, the probabilities in random split selection could be depending on the 'goodness' of the split, rather than uniform. Another variation is to apply these techniques on at most k nodes in the tree. These techniques can also be combined with bagging or the random subspace method. For example, [6] applies bagging and random feature selection.

Another technique to randomize the tree-generation procedure is by arbitrarily predefining the predicate of the root node. Using this technique, forests of size 1 to $|\mathcal{F}|$ can be constructed [13]. This technique is less effective because the randomization by no means depends on how good a certain split is [10].

3.4.4 Vote Aggregation

Just as there are many possible ways to construct the trees in a random forest, likewise, there are many ways to aggregate the votes of all the trees. The most straightforward aggregation technique is a simple majority vote; each tree returns either 0 or 1 and the class with the most votes is considered the classification result of the forest. Formally, if there are t trees in the forests and the vote of the i-th tree on instance (feature vector) x is denoted as $tree_i(x)$ where $tree_i(x) \in \{0, 1\}$, then the classification of the forest is:

$$\text{classify}(x) = \begin{cases} 1, & \text{if } f(x) > 0.5 \\ 0, & \text{if } f(x) < 0.5 \end{cases} \tag{1}$$

where

$$f(x) = \sum_{i=1}^{t} \frac{tree_i(x)}{t} \tag{2}$$

In this case, $f(x) = 0.5$ is a special case where the vote is a tie. This can be resolved in different ways. One way is to use an odd amount of trees. Another way is to let a single tree

resolve the tie, where this tree is preferably not a tree that is generated with randomness. It could for example be a tree that is constructed using the ID3 or MurTree algorithm on the full training dataset. Another more simple way to resolve a tie is to simply classify the instance to the class with the highest prior probability, or even arbitrarily.

This voting technique can be altered by making some trees weigh more, and some trees weigh less. Equation 1 remains unchanged and 2 then becomes:

$$f(x) = \sum_{i=1}^{t} w_i \cdot tree_i(x) \tag{3}$$

where w_i is the weight of tree i and the sum of all weights is equal to 1.

From this, it can be seen that equation 2 is actually a special case of equation 3 where all weights are equal. These weights can be based on different metrics, but they should be set proportional to the individual accuracies of the trees [12]. This can be measured as the accuracy on the training set, on a set-aside validation set or it can be set inversely proportional to the error on the out-of-bag data, which is only applicable if bagging was applied. This final technique was applied by [14] and it was shown that this can significantly improve the accuracy compared to regular (non-weighted) bagged random forests.

3.4.5 Correlation

If the correlation of a forest needs to be calculated, the correlation of each pair of trees in the forest needs to be averaged. For calculating the correlation of a pair of trees, every possible feature vector is needed. This means that $2^{|\mathcal{F}|}$ feature vectors would need to be generated. This is computationally infeasible in many cases. Therefore, [10] proposed to apply Monte-Carlo integration. However, this technique would also generate feature vectors that are not at all in the neighbourhood of the original data. Ho [10] defines an estimate for *tree agreement* of tree i and j as:

$$agreement_{i,j}(\mathcal{G}) = \frac{1}{|\mathcal{G}|} \sum_{k=1}^{|\mathcal{G}|} agree\_on\_instance_{i,j}(x_k) \tag{4}$$

$$agree\_on\_instance_{i,j}(x_k) = \begin{cases} 1, & \text{if } tree_i(x_k) = tree_j(x_k) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Where $x_k \in \mathcal{G}$ and $\mathcal{G}$ is a subset of the input data. Ho [10] used the test set because there the tree generation did not depend on the correlation. An estimate of the correlation of a forest is the average agreement of all pairs of trees in the forest.

3.5 Boosting

A technique different from, but similar to random forests is boosting. Many boosting algorithms exist. Because most of those algorithms apply to regression models and because the MurTree can only be used for classification, this section will focus on the AdaBoost algorithm. The AdaBoost algorithm generates an ensemble classifier, where each classifier has a different amount of say in the final classification. The training data are given misclassification penalties, which are initially all equal. After the first classifier is generated, these penalties are recalculated such that the next classifier will focus more on the previously misclassified instances. Therefore, the classifiers in the ensemble are not independent. The

AdaBoost algorithm is often used with decision trees that have only one predicate node, also called *decision stumps*. If the AdaBoost algorithm is combined with (optimal) decision trees, the output is a forest. This is technically no longer a random forest because the AdaBoost algorithm does not add any randomness. Nevertheless, the similarities are evident. The technical details of the AdaBoost algorithm are described in [15].

4 Forests of Optimal Trees

In this section, the techniques for making random forests are combined with optimal decision trees. Section 4.1 describes which techniques are compatible and which algorithms are implemented. After that, section 4.2 describes a few sightly more advanced techniques that aim to reduce the correlation of the trees in the forest, in order to improve the accuracy. The code-base containing the algorithms is made publicly available through [16].

4.1 Compatibility of Random Forests and Optimal Decision Trees

When optimal decision trees and random forests need to be combined to create a Forest of Optimal Trees, techniques that randomize the tree-generation procedure are not directly applicable. That would disqualify every guarantee of optimality that the optimal decision tree algorithm can give. The only applicable technique of randomizing the tree-generation procedure is predefining the root node arbitrarily. Then, the left and right subtrees are still optimal trees. Furthermore, randomization of the input data can be directly applied. Therefore, the following Forests of Optimal Trees are implemented:

- Arbitrarily predefining the root node such that no feature is the root node twice;
- Bagging using equal weights;
- Bagging using weights proportional to the accuracy on a set-aside validation set;
- Bagging using weights inversely proportional to the error on a the out-of-bag-data;
- Random subspace method using equal weights;
- Random subspace method using weights proportional to the accuracy on a set-aside validation set;
- The AdaBoost algorithm.

In order to prevent ties, only forests with an odd number of trees are used. In the case of different weights, even with an odd number of trees, ties are extraordinarily unlikely but not impossible. Those ties are resolved by classifying to the class with the highest prior probability. Furthermore, bagging is applied with samples of size $|\mathcal{D}|$, the random subspace method samples half the columns and the set-aside sets for the weights are of size $0.5 \cdot |\text{trainingdata}|$.

4.2 Reducing the Correlation

The techniques for randomizing the tree-generation procedure have proven to be an effective technique for decreasing the correlation of the trees. Hence, in order to decrease the correlation of the trees in a Forest of Optimal Trees, other methods are developed for this studies. Although more methods may exist, some of the new ideas are explained in this section.

4.2.1 Restricting the Subspaces

Firstly, the correlation can be decreased by imposing a hard restriction; if a tree in the forest has a predicate node considering feature f_i, no other tree in the forest is allowed to consider feature f_i. This can relatively easily be accomplished by growing the first tree on the full dataset. Afterwards, that tree is traversed such that a list of all considered features is generated. These features are then removed from the dataset. On this new dataset, the next tree can be trained. This procedure can be repeated until enough trees are generated or until the number of features in the dataset is below a certain threshold.

Secondly, a softer restriction is used. Recall that in the random subspace method, features from $\mathcal{F}$ are sampled such that each feature has the same probability of ending up in the sample. This technique is leveraged by making these probabilities inversely proportional to the number of trees in the forest containing a predicate node considering that feature. Hence, as more trees consider feature f_i, the probability of feature f_i ending up in the next subspace will decrease.

4.2.2 Choosing Trees with Low Correlation

Finally, the notion that multiple optimal decision trees can exist is exploited. The number of misclassifications on the training dataset is equal for all these trees, but they can vary structurally. For this purpose, the MurTree algorithm has been altered such that it returns a set of all optimal decision trees instead of just one. Sets cannot contain equal values. Hence, a definition of tree-equality is needed. A simple recursive structural comparison starting from the root is the easiest. However, sometimes some small structural permutations yield a tree that behaves exactly the same. For example, the arguably equal trees from image 2 would be labelled as unequal under this definition. In order to reduce the memory usage of the altered algorithm, the equality definition and algorithm from [17] are applied. Therefore, a set can only contain either one of the trees from image 2. Because still many optimal trees can exist and because the algorithm is now significantly more complex, the algorithm is much slower. Therefore, the algorithm seems infeasible if the depth exceeds 4. Also, the *max_number_of_nodes* parameter that the MurTree supports has been discarded such that only full trees are calculated. With this algorithm at hand, it is possible to apply previous techniques like bagging or the random subspace method, but instead of simply calculating and adding the new tree, calculating the set of optimal decision trees and picking the tree that minimizes the sum of pairwise correlations with the existing trees in the forest. Because the tree is still one of the optimal decision trees, this technique should maintain the *strength*, while decreasing the *correlation*. Therefore, this should decrease the $\frac{\overline{\rho}}{s^2}$ ratio. The used correlation metric is described in section 3.4.5, with $\mathcal{G}$ equal to the training dataset. This gives a bias because the trees are trained on that data, but it is still a good estimate to recognize which trees are more correlated than others.

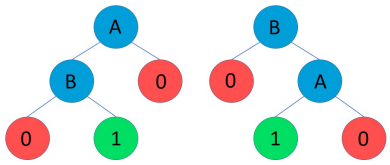

Figure 2: Two trees that classify to 1 if and only if B and not A.

5 Analysis of the Algorithms

5.1 Methodology

The algorithms are compared by running 50 iterations of two-fold-cross-validation on the algorithms, with a time-out of 100 minutes per result. In the following tables, time-outs are denoted as –. All of the algorithms where the tree generation is independent of earlier generated trees are parallelized such that each tree is generated by its own thread. The algorithms are run on a computer with six cores. The sampling for the cross-validation preserves the ratio of $\mathcal{D}^+/\mathcal{D}^-$, and the outcomes of the 50 iterations are averaged after outliers outside of the range $[Q1 - IQR, Q3 + IQR]$ are removed. Here, Qi means the i-th quartile and IQR is the inter-quartile range, equal to $Q3 - Q1$. Many other techniques using 10- or 5-fold-cross-validation with or without $\mathcal{D}^+/\mathcal{D}^-$-preserving sampling and with or without different kinds of outlier-removal have been considered and tried. The described method has proven to be the best for comparing the different algorithms reliably.

5.2 Baseline for Comparing the Algorithms

The described procedure is applied for ten openly available datasets [18] on different algorithms, with different parameters for the algorithms. The dimensions of these datasets are given in table 1. Table 1 also contains the performance of a variety of existing machine learning algorithms on these datasets. Here t means the number of trees in the forest and d means the maximum depth of the tree. These performances form the baseline that the Forests of Optimal Trees can be compared to. In order to easily compare the algorithms, the lowest generalization error per dataset is coloured grey.

Table 1: Dimensions of the datasets and the generalization error rates of C4.5, heuristic random forests and optimal decision trees on these datasets.

				Heuristic Forest [6]			Optimal Tree			
DataSet	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$	C4.5	t=11	t=31	t=101	d=2	d=3	d=4	d=5
anneal	812	93	0.13	0.13	0.12	0.12	0.18	0.15	0.15	–
hepatitis	137	68	0.20	0.17	0.17	0.17	0.20	0.22	0.23	0.22
hypothyroid	3247	88	0.02	0.03	0.03	0.03	0.02	0.02	0.02	–
kr-vs-kp	3196	73	0.01	0.02	0.02	0.01	0.13	0.06	0.05	–
lymph	148	68	0.21	0.18	0.17	0.17	0.22	0.21	0.22	0.22
primary-tumor	336	31	0.19	0.20	0.20	0.19	0.20	0.19	0.19	0.22
soybean	630	50	0.06	0.06	0.05	0.05	0.09	0.06	0.04	0.05
tic-tac-toe	958	27	0.09	0.09	0.06	0.05	0.32	0.26	0.20	0.12
vote	435	48	0.05	0.05	0.04	0.04	0.05	0.06	0.07	0.07
yeast	1484	89	0.29	0.28	0.27	0.26	0.31	0.30	0.31	–

5.3 Analysis of the Directly Compatible Algorithms

The algorithms described in section 4.1 are run for trees with a maximum depth of 2, 3, 4 and 5. Because the generalization errors of these random forests converge to a minimum as the number of trees increases, the number of trees in the forest is set to 101. It was found that in almost all cases, the error no longer decreased at this value. Hence, the convergence has finished. The generalization errors of these algorithms on the ten different datasets can be found in tables 2, 3, 4, 5, 6, 7 and 8. For easy comparison, the lowest generalization

Table 2: Generalization error rates for a Forest of Optimal Trees where the predicate of the root node is assigned arbitrarily without replacement.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.19	0.17	0.14	–	-0.02
hepatitis	0.17	0.18	0.19	–	0
hypothyroid	0.04	0.02	0.02	–	0
kr-vs-kp	0.27	0.13	0.06	–	-0.05
lymph	0.21	0.20	0.18	–	-0.01
primary-tumor	0.22	0.18	0.18	0.19	0.01
soybean	0.15	0.09	0.05	0.04	0
tic-tac-toe	0.30	0.23	0.13	0.05	0
vote	0.05	0.05	0.05	0.05	-0.01
yeast	0.30	0.30	0.29	–	-0.03

Table 3: Generalization error rates for a Forest of Optimal Trees using bagging with equal weights.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.15	–	–	-0.03
hepatitis	0.18	0.18	0.19	–	-0.01
hypothyroid	0.02	0.02	–	–	0
kr-vs-kp	0.13	0.06	–	–	-0.05
lymph	0.19	0.17	0.16	–	0.01
primary-tumor	0.19	0.18	0.17	0.19	0.02
soybean	0.09	0.05	0.04	–	0
tic-tac-toe	0.30	0.19	0.10	0.04	0.01
vote	0.05	0.04	0.05	–	0
yeast	0.30	0.29	–	–	-0.03

Table 4: Generalization error rates for a Forest of Optimal Trees using bagging with weights proportional to the accuracy on a set-aside validation set.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.16	–	–	-0.04
hepatitis	0.16	0.17	0.16	–	0.01
hypothyroid	0.02	0.02	–	–	0
kr-vs-kp	0.13	0.06	–	–	-0.05
lymph	0.17	0.18	0.16	–	0.01
primary-tumor	0.19	0.18	0.17	0.17	0.02
soybean	0.10	0.06	0.05	–	-0.01
tic-tac-toe	0.29	0.17	0.08	0.03	0.02
vote	0.05	0.04	0.04	–	0
yeast	0.30	0.28	–	–	-0.02

Table 5: Generalization error rates for a Forest of Optimal Trees using bagging with weights inversely proportional to the error on the out-of-bag data.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.15	–	–	-0.03
hepatitis	0.17	0.18	0.18	–	0
hypothyroid	0.02	0.02	–	–	0
kr-vs-kp	0.13	0.06	–	–	-0.05
lymph	0.19	0.17	0.17	–	0
primary-tumor	0.19	0.18	0.18	0.20	0.01
soybean	0.09	0.05	0.04	–	0
tic-tac-toe	0.29	0.19	0.10	0.04	0.01
vote	0.05	0.05	0.05	–	-0.01
yeast	0.30	0.29	–	–	-0.03

error per dataset per table is coloured. If the colour for a certain row is green, that means that the new algorithm has outperformed the algorithms in baseline table 1. Similarly, grey means that the lowest errors are equal and red means that at least one of the algorithms in the baseline table outperforms the new algorithm on that dataset. The error decrease with respect to the baseline table is also given.

The results for the AdaBoost algorithm combined with optimal decision trees can be found in table 8. Every dataset timed out on $d = 4$, so $d = 5$ would simply be another column with solely timeouts. Hence, the depth ranges from 1 to 4 for this experiment. Remember that in the AdaBoost algorithm, the next tree depends on the previous trees. Therefore, it cannot be parallelized like the previous algorithms. Furthermore, $d = 1$ is included because this means that each tree in the forest contains one predicate node and two classification nodes, this is also known as a decision stump. For decision stumps, a heuristic algorithm would yield the same classifier.

From all these tables it can be seen that the new Forests of Optimal Trees sometimes outperform the baseline algorithms. However, in most cases, the Forests of Optimal Trees perform comparable to or worse than the baseline algorithms. After inspecting the time-outs, it can be said that with respect to the runtime of the algorithms, the heuristic approaches outperform the optimal approaches by an order of magnitude.

Table 6: Generalization error rates for a Forest of Optimal Trees using the random subspace method with equal weights.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.16	0.14	–	-0.02
hepatitis	0.18	0.19	0.20	0.18	-0.01
hypothyroid	0.02	0.02	0.02	–	0
kr-vs-kp	0.13	0.06	0.04	–	-0.03
lymph	0.20	0.17	0.16	0.16	0.01
primary-tumor	0.20	0.18	0.18	0.19	0.01
soybean	0.13	0.07	0.05	0.05	-0.01
tic-tac-toe	0.31	0.20	0.13	0.06	-0.01
vote	0.04	0.05	0.04	0.05	0
yeast	0.31	0.30	0.29	–	-0.03

Table 7: Generalization error rates for a Forest of Optimal Trees using the random subspace method with weights proportional to the accuracy on a set-aside validation set.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.16	0.14	–	-0.02
hepatitis	0.16	0.16	0.18	0.18	0.01
hypothyroid	0.02	0.02	0.02	–	0
kr-vs-kp	0.13	0.06	0.04	–	-0.03
lymph	0.17	0.16	0.16	0.16	0.01
primary-tumor	0.19	0.17	0.17	0.18	0.02
soybean	0.14	0.08	0.06	0.05	-0.01
tic-tac-toe	0.29	0.20	0.13	0.06	-0.01
vote	0.05	0.04	0.04	0.04	0
yeast	0.30	0.29	0.28	–	-0.02

Table 8: Generalization error rate for a Forest of Optimal Trees using the AdaBoost algorithm.

DataSet	Depths d=1	d=2	d=3	d=4	Error decrease
anneal	0.15	0.13	0.13	–	-0.01
hepatitis	0.18	0.18	0.20	–	-0.01
hypothyroid	0.02	0.02	0.03	–	0
kr-vs-kp	0.04	0.01	0.01	–	0
lymph	0.17	0.17	0.17	–	0
primary-tumor	0.18	0.21	0.21	–	0.01
soybean	0.06	0.05	0.05	–	-0.01
tic-tac-toe	0.11	0.10	0.00	–	0.05
vote	0.04	0.05	0.05	–	0
yeast	0.28	0.28	0.29	–	-0.02

Table 9: Generalization error rate for a Forest of Optimal Trees using the soft-restricted random subspace method.

DataSet	Depths d=2	d=3	d=4	d=5	Error decrease
anneal	0.18	0.16	0.14	–	-0.02
hepatitis	0.18	0.20	0.20	–	-0.01
hypothyroid	0.02	0.02	–	–	0
kr-vs-kp	0.13	0.06	–	–	-0.05
lymph	0.21	0.18	0.16	–	0.01
primary-tumor	0.19	0.18	0.18	0.18	0.01
soybean	0.13	0.08	0.05	0.05	-0.01
tic-tac-toe	0.30	0.21	0.13	0.06	-0.01
vote	0.05	0.05	0.04	0.05	0
yeast	0.31	0.30	–	–	-0.04

5.4 Analysis of the Advanced Algorithms

The method where features that are considered by earlier trees are excluded from the subspace of the next trees, as described in section 4.2, has been implemented and benchmarked. The results if $t = 1$ are equal to the optimal tree results in table 1 because the first tree in the forest is simply the optimal tree of the corresponding depth. When more trees are added to the forest, the error increases instead of decreases. The most likely reason for this phenomenon is that some features have more predictive value than others. For example, if obesity needs to be predicted from patient data, the feature *weight* is much more valuable than the feature *favourite colour*. As a result, most of the good features are taken by the first tree. This implies that the second tree is a worse classifier than the first. This pattern continues, making the forest perform worse as more trees are added. Because this method performed much worse than the results in the baseline table, and because the error increased as the number of trees in the forest increased, these results are not reported in a table.

Because the hard restriction yielded no success, the soft restriction from section 4.2 has also been implemented and analysed. The performance of this method can be found in table 9. Although this technique outperforms the baseline slightly in some cases, in other cases the existing methods are still better.

Finally, two algorithms that actively attempt to minimize the correlation while maintaining the strength have been implemented. The first algorithm uses bagging where the added

Table 10: Generalization error rate for a Forest of Optimal Trees using bagging where the next tree is an optimal tree with low correlation.

DataSet	Depths d=2	d=3	d=4	Error decrease
anneal	0.18	–	–	-0.06
hepatitis	0.17	–	–	0
hypothyroid	0.02	–	–	0
kr-vs-kp	0.13	–	–	-0.12
lymph	0.18	–	–	-0.01
primary-tumor	0.19	0.18	–	0.01
soybean	0.10	0.06	–	-0.02
tic-tac-toe	0.29	0.18	0.10	-0.05
vote	0.04	–	–	0
yeast	0.30	–	–	-0.04

Table 11: Generalization error rate for a Forest of Optimal Trees using the random subspace method where the next tree is an optimal tree with low correlation.

DataSet	Depths d=2	d=3	d=4	Error decrease
anneal	0.18	–	–	-0.06
hepatitis	0.19	0.19	–	-0.02
hypothyroid	0.02	–	–	0
kr-vs-kp	0.13	0.06	–	-0.05
lymph	0.20	–	–	-0.03
primary-tumor	0.19	0.18	–	0.01
soybean	0.12	0.08	–	-0.04
tic-tac-toe	0.30	0.21	0.13	-0.08
vote	0.05	0.05	–	-0.01
yeast	0.30	0.30	–	-0.04

tree is the tree from the set of optimal decision trees that minimizes the sum of pairwise correlations. Its performance can be found in table 10. The second algorithm is very similar but uses the random subspace method instead of bagging. The performance of the second algorithm can be found in table 11. Because these algorithms are more complex and slower than the algorithms where only one optimal tree is calculated per iteration, the number of trees in the forest is set to 31 instead of 101. From the results and the time-outs it can be seen that in general, these Forests of Optimal Trees are much slower than the baseline algorithms. Also, in terms of accuracy, the baseline algorithms often outperform the forests.

6 Discussion

Because the research on optimal decision trees is still in its infancy, Forests of Optimal Trees are still limited. If in the future faster optimal decision tree algorithms are developed, it automatically follows that Forests of Optimal Trees can also be generated faster. Also, the implementation of the MurTree algorithm used for this research does not contain every optimization technique that the original algorithm uses; similarity-based bounding and incremental computation [5] are omitted for the sake of time. If the experiments are repeated with those techniques implemented, the reported error rates should not change, although fewer timeouts could be expected.

Regarding the feasibility of practical use of the Forests of Optimal Trees, if the maximum depth of the trees is smaller than five and the dataset is not too large, the runtime often stays within the order of seconds or minutes. The results in section 5 use a time-out of 100 minutes, but 50 runs of two-fold-cross-validation are done. This means that 100 forests are generated within those 100 minutes. Therefore, if a result in this report did not time-out, that means that it is certainly within feasible bounds. Larger datasets or deeper trees are not infeasible by definition, but as the datasets get larger or the trees get deeper, the runtime increases. Hence, scalability to larger datasets could be a problem. Nevertheless, most random forest algorithms allow easy parallelisation if the trees are independent. If enough cores and memory are available, generating a forest could theoretically take roughly as long as generating one tree. In conclusion, in scenarios where a small gain in classification accuracy can have important real-world advantages, and the cost of the time and computation power are worth it, Forests of Optimal Trees can be useful classifiers.

Furthermore, the hyperparameters of the algorithms could be explored further. For example, the algorithms that use the random subspace method all use $0.5|\mathcal{F}|$ as the size of the subspace because Ho [10] showed that this works best for heuristic random forests. Perhaps for Forests of Optimal Trees, other parameters work better. Also, all results are limited to optimal decision trees of depth 5 at maximum. If in the future algorithms for optimal decision trees improve, it will be interesting to investigate forests with deeper trees. This is especially interesting because trees in a heuristic random forest are typically grown as deep as possible, without any pruning [6], [11]. The MurTree algorithm also allows for optimization of the maximum number of predicate nodes in the tree. For this research, that parameter is always set to $2^{depth} - 1$, which implies a full tree. Optimizing this parameter might improve the performances of the forests.

Finally, because the current versions of optimal decision tree algorithms are limited to binary decision trees, so are the Forests of Optimal Trees. Better binarisation strategies could improve the performances of optimal decision trees as well as Forests of Optimal Trees. In this studies, the binarised datasets from [18] are taken as the ground truth. The heuristic decision trees and heuristic random forests may have performed better if they were trained on the original unbinarised data. This might have resulted in a slightly unfair comparison.

7 Conclusions and Future Work

In conclusion, many algorithms that combine random forests with optimal decision trees have proven to outperform heuristic random forests or a single optimal or heuristic decision tree in some cases. However, there are also many cases where the heuristic random forests outperform the Forests of Optimal Trees. Whether there is a gain in accuracy seems data-dependent. When there is a gain in accuracy, it is often only one or two per cent, whereas the increase in runtime is an order of magnitude. Because the research on optimal decision trees is still in its infancy, improvements such as faster or better-parallelized algorithms in this field will most likely result in faster algorithms for Forests of Optimal Trees.

In the future, it is certainly interesting to research a few ideas further. One idea is to alter the MurTree algorithm such that it returns all optimal trees as well as all trees that have m more misclassifications than the optimal trees. Here, m should be a low value. This would give the trees some leeway, while the returned trees remain of high quality. This algorithm can then be used in the same way as the algorithm for all optimal trees was used in section 4.2. The small sacrifice of tree quality has the potential to greatly decrease the correlation. Another idea is to investigate what data characteristics are responsible for the fact that sometimes a heuristic random forest and other times a Forest of Optimal Trees is the best option. If these characteristics can be clearly defined, it is easier to choose an appropriate machine learning algorithm.

Furthermore, another interesting aspect for future research is a more advanced hyper-parameter search. This would include optimizing the maximum number of nodes in the optimal decision trees, optimizing the size of the subspace in the random subspace method and optimizing the bootstrap size for Bagging. Also, more datasets could be benchmarked. For this future research, the code available through [16] can be used.

Finally, it is important to remember that a Forest of Optimal Trees is not the same as an Optimal Forest. Because algorithms to create a single optimal decision tree are already really time-consuming, Optimal Forests are most likely an order of magnitude worse. Although no research has gone into Optimal Forests so far, perhaps the future might bring some smart optimization techniques making these algorithms more feasible.

8 Responsible Research

8.1 Responsible Use of the Algorithms

Machine learning algorithms should never be used as black boxes. Therefore humans and algorithms should work together. Currently, if for example 51 trees vote for class 0 and 50 trees vote for class 1, this means that the instance is classified as 0. However, in a socially sensitive context, the uncertainty of the trees can also be taken into account. If the vote is (almost) unanimous, the outcome can be used directly, whereas if the vote is (almost) a tie, the data-point can be rejected such that a human domain expert can resolve that case manually. The boundary of how confident the algorithm should be, must be defined by the user. Even when the vote is almost unanimous, this can never absolutely guarantee that the outcome is correct. Hence, by means of sampling, humans should keep an eye on what the algorithm is doing and verify that the classifications are correct.

Sometimes a misclassification of a 0 as a 1 is not too bad, whereas a misclassification of a 1 as a 0 could be disastrous, or the other way around. The algorithms can meet the needs in these cases as well. For example, when patient data is used to try to find a rare illness. Marking a healthy patient as 'possibly ill', and consequently inviting the patient to do some tests when it turns out that the patient does not have the illness, can cause some stress for that patient and wastes some time and resources from the doctor, but the damage is certainly limited. However, if a patient with the illness is marked as 'healthy', and hence not invited for the tests, the consequences could be lethal. In this case, a misclassification of 'healthy' as 'possibly ill' is less bad than a misclassification of 'ill' as 'healthy'. In the original version of the MurTree algorithm, the algorithm minimizes the number of misclassifications. In the code-base implemented for this studies, the MurTree algorithm has been enhanced such that certain misclassifications can have different weights. Therefore the algorithm now minimizes the sum of weights of misclassified instances. This means that the original version of the algorithm is a special case of the enhanced algorithm, where all weights are equal. The relative weights of these misclassifications can be determined by the user.

8.2 Research Reproducibility

The experiments in this studies were run with openly available datasets. These datasets are available through [18]. The algorithms used are clearly explained in the text and are referenced for more detail and to give credit to the original authors. Furthermore, the used hyperparameters are also clearly specified in sections 4 and 5. Finally, the code that was implemented for this research is publicly available through [16] so that anyone can use this code-base to run more benchmarks on different datasets.

Acknowledgements

I would like to thank my supervisor, dr. Emir Demirović, for his valuable feedback and support during this studies. Furthermore, I would like to thank my peer students Abel Mălan, Ayush Patandin and Ola Wolska for sharing interesting thoughts and literature, while independently working on closely related projects.

References

[1] J. R. Quinlan, "Induction of decision trees", *Machine Learning*, vol. 1, pp. 81–106, 1986. DOI: 10.1007/BF00116251.

[2] J. R. Quinlan, *C4.5: Programs for Machine Learning*, ser. Ebrary online. Elsevier Science, 2014, ISBN: 9780080500584. [Online]. Available: https://books.google.nl/books?id=b3ujBQAAQBAJ.

[3] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, "Classification and regression trees (wadsworth statistics/probability)", 1984.

[4] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete", *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976, ISSN: 0020-0190. DOI: 10.1016/0020-0190(76)90095-8.

[5] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, *Murtree: Optimal classification trees via dynamic programming and search*, 2020. arXiv: 2007.12652 [cs.LG].

[6] L. Breiman, "Random forests", *Machine Learning*, vol. 45, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[7] S. Nijssen and E. Fromont, "Optimal constraint-based decision tree induction from itemset lattices", *Data Mining and Knowledge Discovery*, vol. 21, pp. 9–51, 2010. DOI: 10.1007/s10618-010-0174-x.

[8] G. Aglin, S. Nijssen, and P. Schaus, "Learning optimal decision trees using caching branch-and-bound search", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3146–3153, Apr. 2020. DOI: 10.1609/aaai.v34i04.5711.

[9] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, pp. 123–140, 1996. DOI: 10.1007/BF00058655.

[10] T. K. Ho, "The random subspace method for constructing decision forests", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998. DOI: 10.1109/34.709601.

[11] S. Bernard, L. Heutte, and S. Adam, "Towards a better understanding of random forests through the study of strength and correlation", in *Emerging Intelligent Computing Technology and Applications. With Aspects of Artificial Intelligence*, D. S. Huang, K. H. Jo, H. H. Lee, H. J. Kang, and V. Bevilacqua, Eds., Springer Berlin Heidelberg, 2009, pp. 536–545, ISBN: 978-3-642-04020-7. DOI: 10.1007/978-3-642-04020-7_57.

[12] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization", *Machine Learning*, vol. 40, pp. 139–157, 2000. DOI: 10.1023/A:1007607513941.

[13] S. W. Kwok and C. Carter, "Multiple decision trees", in *Uncertainty in Artificial Intelligence*, ser. Machine Intelligence and Pattern Recognition, R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer, Eds., vol. 9, North-Holland, 1990, pp. 327–335. DOI: 10.1016/B978-0-444-88650-7.50030-5.

[14] H. B. Li, W. Wang, H. W. Ding, and J. Dong, "Trees weighting random forest method for classifying high-dimensional noisy data", in *2010 IEEE 7th International Conference on E-Business Engineering*, 2010, pp. 160–163. DOI: 10.1109/ICEBE.2010.99.

[15] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997, ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504.

[16] [Online]. Available: https://github.com/jmolhoek/forests_of_optimal_trees.

[17] H. Zantema, "Decision trees: Equivalence and propositional operations", Jul. 1998.

[18] [Online]. Available: https://github.com/aia-uclouvain/pydl8.5/tree/master/datasets (visited on 11/06/2021).