RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences
Chair of Computer Science 13 (Computer Vision)
Prof. Dr. Bastian Leibe

(Pro)Seminar Report

Random Forests

Kaan İmamoğlu, 451439
Bora Deren, 438377

August 2024

Advisor: Stephanie Käs

Abstract

Decision tree learning is a supervised learning algorithm that is easy to interpret due to its structure and is useful for solving classification and regression problems. Although decision trees give us some accurate information about a given dataset, they cause several problems such as overfitting when we train them on complex or big datasets. In order to solve these problems, Leo Breiman gave a proper introduction to Random Forest in his article written in 2001 [Bre01]. Random forest, as an ensemble learning method has been significant by enhancing the accuracy in making predictions on datasets that would overfit a single decision tree. This method combines multiple decision trees to improve overall model robustness and predictive accuracy.

Contents

1 Introduction

A decision tree is a type of technique that helps an organization or institution understand its choices, risks, rewards, and goals on problems. Due to their simple structure and interpretability, decision trees, the basic components of random forests, have recently been used by many individuals, institutions, and organizations.
The key feature of decision trees is the recursive subsetting of the target space according to the values of associated input fields or predictors, and the creation of partitions and associated subsets of data, called leaves or nodes, containing increasingly similar within-leaf target values and increasingly different between-leaf or between-node values at any level of the tree [DV13]. Despite the many advantages of decision trees, they also have many disadvantages for their users. Decision trees can get overfit very fast, especially when dealing with large datasets. This is one of the main reasons why ensemble techniques and supervised machine learning algorithms such as random forests are created.

Random forest is a supervised machine learning algorithm widely used to solve both classification and regression problems and is a nice, easy, and effective method for data scientists. Random decision forests correct the habit of decision trees to overfit training sets. The first algorithm for random decision forests was created by Tin Kam Ho in 1995 using the random subspace method [Ho95], but the proper introduction of random forests was made in a paper by Leo Breiman.

Random forests introduced by Leo Breiman in 2001 [Bre01], is based on the concept of bagging to reduce the number of errors that can occur in decision trees or the correlation between trees and increase their stability. Random forests or random decision forests have become a very popular machine learning technique since they were introduced by Leo Breiman. Although there are many released ensemble methods, random forests are one of the fastest among these released ensemble methods due to the simple structure of the decision trees generated by algorithms such as ID3, C4.5, and CART.

Low correlation is the most important factor for random forest and the main purpose of the algorithm is to create uncorrelated decision trees. In this way, randomization is needed to obtain uncorrelated decision trees. By combining the results of multiple decision trees generated by different data, random forests achieve a balance between bias and variance, leading to generalization. This method not only correlates the trees but also improves the overall robustness and accuracy [LWZ12]. Random forests have become one of the most popular research methods in the field of data mining and knowledge transfer to the biological field. Random forests are employed in various fields, including medical diagnostics, and financial forecasting, where they improve predictive accuracy by combining the outputs of multiple decision trees.

2 Decision Trees

2.1 Introduction to Decision Trees

A decision tree is a classification model that uses a tree-like model of decisions and their possible outcomes. The decision tree consists of nodes that form a rooted tree, the root node does not have an incoming edge here. All other nodes have exactly one incoming edge. If a non-root node has edges sticking out, this node is called an internal node (or interior node). All other nodes are called leaves or leaf nodes [MR05], as we can see in Figure 1.

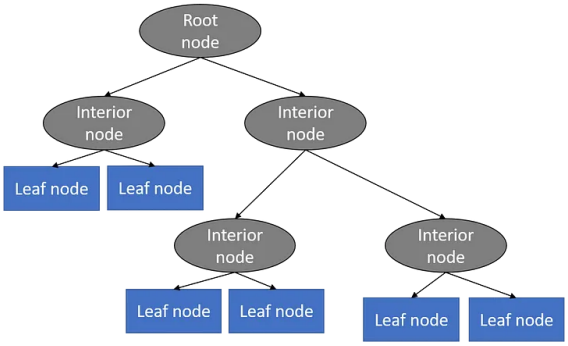


Figure 1: Structure of decision trees [Cha19]

In order to build decision trees, a set of training examples with known class labels are analyzed. They are then used to categorize examples that haven't been seen before. Highly accurate predictions can be made by decision trees when they are trained on high-quality data.

Decision trees typically support classification problems with more than two classes and can be modified to solve regression problems. Once decision trees are constructed, they quickly classify new items, but a small change in input data can cause so many changes in the constructed tree [KS08].

Considering these definitions and explanations, decision trees have many advantages. First of all, decision trees are non-parametric, allowing them to model complex relationships between inputs and outputs without making any assumptions. Moreover, they can combine data types, whether ordinal or categorical variables or a combination of both. Additionally, decision trees are robust to anomalies, label errors, and irrelevant or noisy variables. Thus due to their structure, decision trees are simple to interpret even for those who are unfamiliar with statistics [Lou14].

2.2 Classification Trees

As we said earlier, decision trees have two main use cases. One of them is classification problems. A classification problem is a problem, where you have objects and categories trying to find out which object belongs to which category. More precisely, you have a dataset that consists of independent variables and dependent variables. The dependent variables are also called the classes or labels. The goal is, to find an algorithm that truly classifies a new object given with its independent variables into its class. An example would be, giving a dataset consisting of photos of cats and dogs, training the model with this dataset using an algorithm, and then letting the model classify a newly given photo whether it is a cat or dog photo. In the case we have two classes, we call this a binary classification, and in case there are more than two classes, it is called multiple classification. This is where classification trees come in. This so-called "algorithm" could be reached with classification trees. Here is an example of binary classification: We have a simplified and edited version of the Titanic dataset [Kag] consisting of data of 10 passengers of the Titanic disaster with independent variables age, sex, cabin, and the classes "dead" and "survived". In Table 1 we can see the basic representation of our basic dataset.

Male is identified as 0 and Female as 1.
Cabins are assigned 0, 1, 2, and 3 with 0 being the lowest and 3 the highest.
Survived is identified as 1 and Dead as 0.

PassengerID	Age	Sex	Cabin	Survival Status
1	22	0	2	1
2	38	1	3	1
3	26	1	1	0
4	35	0	2	0
5	28	0	3	1
6	42	1	1	1
7	19	0	0	0
8	32	1	0	1
9	50	0	1	0
10	23	1	2	1

Table 1: Passenger data with independent variables and survival status.

When we now have a new passenger who is not part of the dataset, and we try to classify him/her as "Dead" or "Survived" according to the dataset and his independent variables, we can use a classification tree which is constructed by using the dataset's data.
After applying a classification tree algorithm which we will be discussing the details of it later, we get the classification tree in Figure 2.

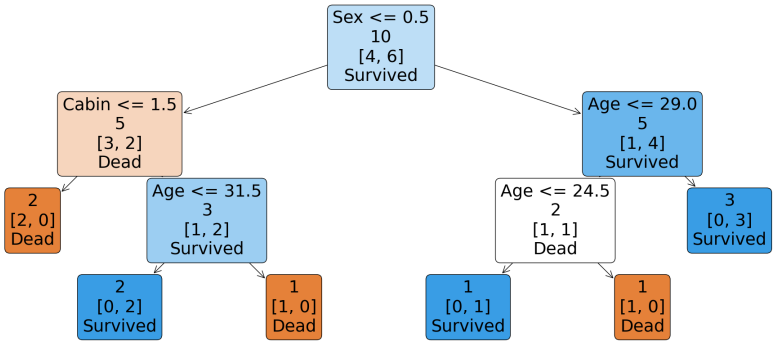


Figure 2: Classification Tree

- The first line of non-leaf nodes says which branching variable (feature) is used to branch and the threshold (the specific value of the feature that defines the split). E.g., the root node used "Sex" as the branching variable with a threshold of 0.5.
- The second line says how big the remaining dataset is (e.g., for the root, it is 10).
- The third line tells how the remaining dataset is divided by the branching variable (e.g., for the root, [4,6] means 4 of the remaining data points are from class "Dead", 6 of them are in class "Survived").
- The last line tells us in which category the new passenger would fall, according to the branchings done until that node (For the root, it is "Survived", because we have more survivors than dead in our dataset).

Assume the new passenger has this data: Age = 30, Sex = 0, Cabin = 1.
With the decision tree in Figure 2, it would be classified as "Dead" (through the most left leaf), because he is "Male" and his Cabin is 1.

2.3 Regression Trees

As we have seen, classification trees are used to classify objects to a class (label). The main difference between regression trees to classification trees is the type of the target. In classification problems, we have categorical (discrete) targets, whereas in regression problems we have numeric (contiguous) targets. Here we are not trying to predict which class an object falls into, rather we are trying to predict a numerical value. For example: Given a dataset of cars in Table 2.

Age (Years)	Mileage (km)	Horsepower (HP)	Price (Euros)
3	45000	140	21000
3	50000	150	20000
4	60000	160	19500
4	70000	170	19000
5	80000	180	18500
5	90000	190	18000
6	85000	200	17500
6	95000	210	17000
7	90000	220	16500
7	100000	230	16000

Table 2: Car dataset with Age, Mileage, Horsepower, and Price

In this case, we want to predict the price of a newly given car, which is a continuous value, using this dataset. We apply the regression tree algorithm to this dataset and get this regression tree in Figure 3:

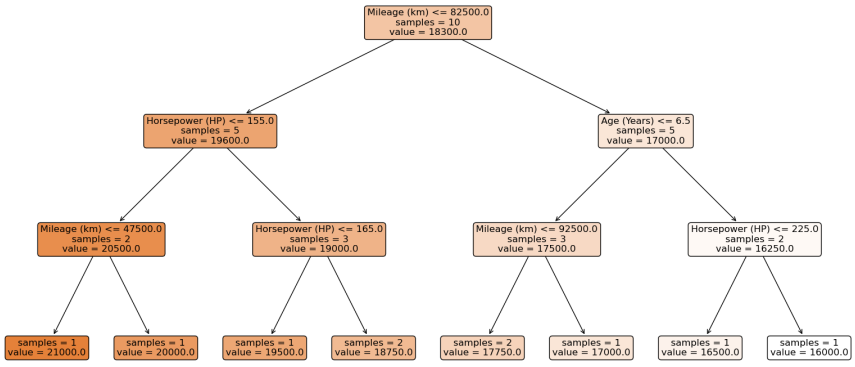


Figure 3: Regression Tree created with scikit.learning and matplotlib

After constructing the regression tree, we can predict the value of a car that is 10 years old, has a mileage of 90000 km, and has 100 HP. This prediction should end up in the second most right leaf with a price of 16500 euros.

2.4 Univariate and Multivariate Decision Trees

2.4.1 Univariate Trees

A univariate decision tree is a decision tree that tests only one feature in a node. That means it divides the dataset into regions by looking at one single feature's values. If it is a binary tree, it will divide the dataset into two regions. A binary tree is a decision tree, which's nodes have two child nodes. That means, that at each node, the dataset is divided into two discrete subsets. Our previous decision tree in Figure 2 is a univariate binary decision tree, since it tests one feature in each node, and each node has 2 successors. There is a function defined for each decision node. If n is the node, we define the function of that node as $f_n(x) : x_i \geq w$ where x is the sample point, x_i is its value of feature i, and w is the threshold. Hence, node n divides the dataset into two subsets $L_n = \{x|x_i \geq w\}$ and $R_n = \{x|x_i \leq w\}$. This process is called a binary split [Alp10]. It is also possible to generalize this into non-binary trees with more splits at a node as seen in Figure 4.

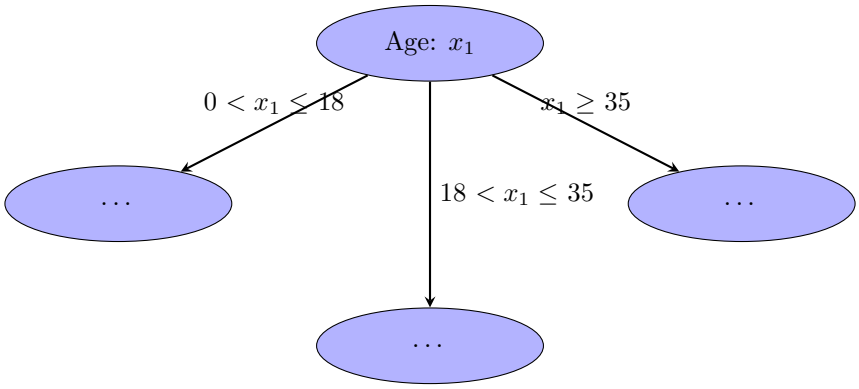


Figure 4: Univariate, non-binary decision tree example. The univariate feature here is Age, where x_1 is the age of a sample point. The root node has 3 successors.

Since we have seen there can exist there exist lots of trees for a given dataset, we want to calculate the minimum tree. That means, the tree with the least nodes, best splits, etc. However, it is NP-complete to calculate a minimum tree[Qui86, Alp10]. That is why, we are working with approaches to determine minimal trees.

2.4.2 Multivariate Trees

More general than univariate trees, in multivariate trees we can test more than one feature in a node. So, it is possible to make a dividing of the dataset which lies on dependencies of multiple features at one single node. This means for each node, there is a multivariate function defined. The splittings are done according to the value intervals of this function.

2.5 Summary

This section provided an overview of decision trees, discussing their basic structure and the types of nodes they contain. Decision trees are used for both classification and regression tasks, where the paths from the root to the leaf nodes represent decision rules. Their primary advantages are simplicity and ease of interpretation, although they are prone to overfitting and instability.

3 Main Concepts and Problems With Decision Trees

As we have seen with examples of decision trees, we will now delve into the "algorithms" that are after all the process. We will discuss how these algorithms work, how decision trees are constructed, how they choose their branching feature, how they calculate their branching value etc.. After that, we will explain what the main problems are about decision trees as a motivation for Random Forest.

3.1 Splitting Rules and Criteria

One of the most important factors in constructing a decision tree based on a given dataset is to decide the splitting (branching) criteria. In our previous example on the Titanic dataset, ours chose "Sex" as the root criterion because it was the best criterion to divide the dataset into two groups since there were 4 women and 6 men. And of course, there is a smart calculation behind this decision. In this chapter, we will focus on metrics such as information gain, entropy, impurity, and Gini index which are used in making these splits.

3.2 Impurity

In the case of a decision tree for classification, namely, a classification tree, the goodness of a split is quantified by an impurity measure. A split is pure if after the split, for all branches, all the instances choosing a branch belong to the same class. [Alp10]

Let m be the node,
N_m is the number of training instances reaching node m. For the root node, it is N.
N_m^i is the number of instances of N_m that belong to class C_i with $\sum_i N_m^i = N_m$.

Given that an instance x reaches node m, the estimate for the probability of class C_i is
$P(x \text{ reaches class } C_i \mid x \text{ reaches node } m) = \frac{N_m^i}{N_m}$ [Alp10]

We call m pure if $P(x \text{ reaches class } C_i \mid x \text{ reaches node } m)$ is whether 0 or 1 for all i. 0 means there is no instance belonging to class C_i, whereby 1 means all instances reached node m belong to class C_i. [Alp10]

3.2.1 Entropy

Entropy (also called Shannon Entropy [Sha48]) is a metric that measures impurity. Impurity means irregularity and randomness of data in a dataset. Therefore, entropy refers to the homogeneity of a dataset. The entropy of a dataset X is labelled as $H(X)$, and can take values between 0 and $log_2 n$ where n is the size of the dataset. Entropy is mainly used in classification trees to optimize splitting. Here is the formula to calculate the entropy of a given dataset:

$$H(X) = -\sum_{i=1}^{n} P(C_i, m) \log_2 P(C_i, m)$$

where $P(C_i, m) = P(\text{instance reaches class } C_i \mid \text{instance reaches node } m)$, n is the number of classes, X is the subset of which we are calculating the entropy.

For example, if we have a dataset M at a given node m with 3 classes (let us call them A, B, C). In the remaining dataset M at node m, there are 2 objects in A, 3 objects in B, 1 object in C. Then $P(A, m) = \frac{2}{6}$, $P(B, m) = \frac{3}{6}$, $P(C, m) = \frac{1}{6}$. Hence the entropy of M is calculated as:

$H(M) = -(\frac{2}{6} \log \frac{2}{6} + \frac{3}{6} \log \frac{3}{6} + \frac{1}{6} \log \frac{1}{6}) \approx 1.459$

More classes mean higher entropy. If we have just one class, the entropy will be 0. Another factor that implies entropy is the distribution of classes. If a class has much more samples than others, the entropy will be smaller. On the other hand, if the classes are balanced, entropy will get higher.

Entropy is used to measure how randomized a subset is at a node when constructing a classification tree. It is mainly used in context with Information Gain which will be discussed later.

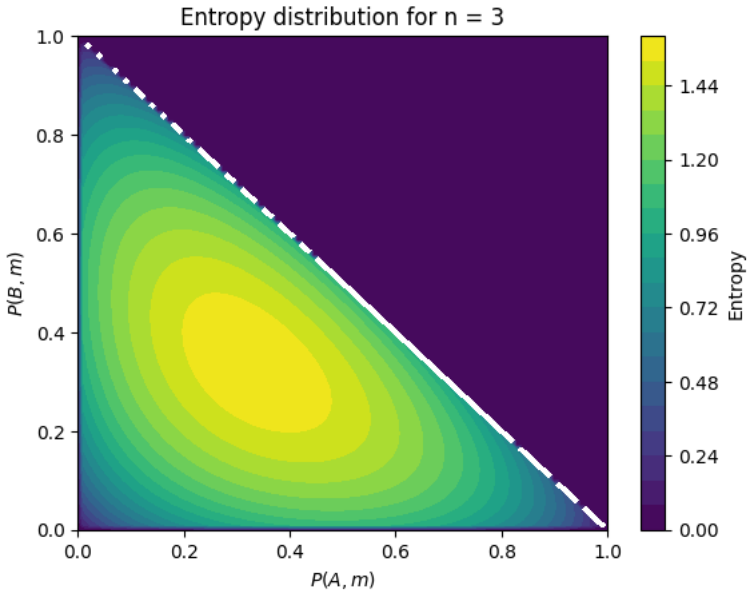


Figure 5: Entropy distribution with 3 classes.

When we look at Figure 5, it can be seen that entropy is at its maximum ($log_2 3 \approx 1.585$) if the probabilities of an instance belonging to a class are equal ($P(A,m) = P(B,m) = P(C,m) = 0.33$).

3.2.2 Gini Index

Another measure of impurity is the Gini index (also called Gini impurity). Similarly to entropy, the Gini index is used to measure the impurity at a node (subset). The Gini index was actually used to measure income inequality [Gin12]. In the context of machine learning, it is used to measure the "diversity" within the dataset.

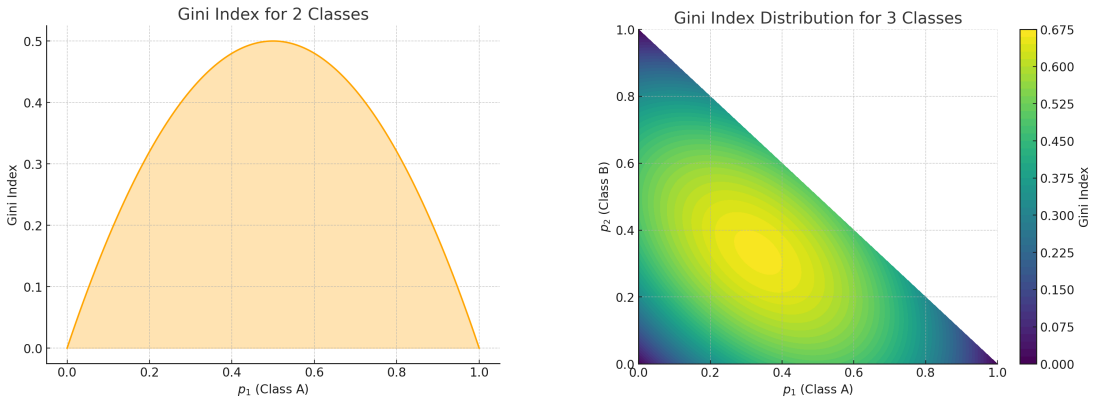
Let X be a dataset with n instances and the classes $C_1, C_2, ..., C_n$. p_i is the probability of an instance belonging to class C_i. Then the Gini impurity of dataset X is defined as:

$\text{Gini}(X) = 1 - \sum_i^n p_i^2$

Let us assume we have a dataset of 100 samples with 3 classes C_1, C_2, C_3 with frequencies 50, 30, and 20. Then, $p_1 = 0.5, p_2 = 0.3, p_3 = 0.2$, $\text{Gini}(X) = 1-(0.5^2+0.3^2+0.2^2) = 1-(0.25+0.09+0.04) = 1 - 0.38 = 0.62$

The Gini index takes values between 0 and 1. Where the Gini index goes toward 0, homogeneity increases (instances are distributed inequally over classes), and where it goes toward 1, it means diversity increases (instances are equally distributed over classes). The maximum value, the gini

index can take is $1 - \frac{1}{n}$ where n is the size of the dataset. It goes towards 1, where n increases ($\lim_{n\to\infty}(1 - \frac{1}{n}) = 1$). For n = 2, Gini($X$) takes values between 0 and 0.5. We can see the Gini distributions for 2 and 3 classes in Figure 6



(a) Gini Index for 2 Classes

(b) Gini Index Distribution for 3 Classes

Figure 6: Gini Index Distributions

The Gini index was adapted into Machine Learning with the development and popularity of decision trees. One of the most important contributions is made by the book "Classification and Regression Trees" in 1984, by Breiman et al.[BFOS84].

3.2.3 Quantification of Information Gain

Information Gain is an important term used in machine learning, especially in decision trees. Information gain is a metric to measure how well a feature (independent variable) divides a dataset into subsets.

Information Gain is defined over entropy. It measures how much the entropy decreases after a split. For instance, we measure the entropy of dataset X before splitting. Then we calculate the average entropy over the subsets after the split based on a feature x_j. Let us call $H(X)$ the entropy of dataset of a dataset X before split, If x_j is a categorical feature, $H(S_v)$ is the entropy of the subset S_v, where S_v is the subset with all instances with $x_j = v$, $Val(x_j)$ is the set of possible values of feature x_j. Then the average entropy over the subsets S_v split on feature x_j is calculated as:

$\sum_{v\in \mathrm{Val}(x_j)} \frac{|S_v|}{|S|} H(S_v)$

Then, information gain of splitting S by feature x_j is defined as the subtraction of the average entropy from the original entropy:

$IG(S, x_j) = H(X) - \sum_{v\in \mathrm{Val}(x_j)} \frac{|S_v|}{|S|} H(S_v)$

If x_j is non-categorical (continuous), we usually divide the dataset into two subsets using a threshold value t. In this case, the information gain of splitting S by feature x_j with threshold value t is defined as:

$IG(S, x_j, t) = H(S) - (\frac{|S_{\leq t}|}{|S|} H(S_{\leq t}) + (\frac{|S_{>t}|}{|S|} H(S_{>t}))$, where $S_{>t}$ is the subset of all instances with $x_j > t$ and $S_{\leq t}$ the subset of all instances with $x_j \leq t$

Let's calculate the Information Gain on an example of a dataset given in Table 3.

Instance	x_1	x_2	x_3	x_4	Class
1	2.5	3.0	1.5	2.0	A
2	1.5	2.5	2.0	1.5	B
3	3.0	4.0	3.5	3.0	C
4	2.0	3.5	1.0	2.5	A
5	3.5	2.0	2.5	3.5	B

Table 3: Dataset X with Features x_1, x_2, x_3, x_4 and Classes A, B, C

$H(X) = -\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{1}{5}\log_2\left(\frac{1}{5}\right)\right) \approx 1.5219$

Now, we divide our dataset into two subsets X_1 and X_2 based on the continuous feature x_1 with threshold value t = 2.5 as seen in Figure 7.

Instance	x_1	x_2	x_3	x_4	Class
1	2.5	3.0	1.5	2.0	A
2	1.5	2.5	2.0	1.5	B
4	2.0	3.5	1.0	2.5	A

(a) Subset $X_1 : x_1 \leq 2.5$

Instance	x_1	x_2	x_3	x_4	Class
3	3.0	4.0	3.5	3.0	C
5	3.5	2.0	2.5	3.5	B

(b) Subset X_2: $x_1 > 2.5$

Figure 7: Subsets after the split

Hence, information gain splitting feature x_1 with threshold 2.5 in dataset X is:

$IG(X, x_1, 2.5) = H(X) - (\frac{|X_1|}{|X|}H(X_1) + (\frac{|X_2|}{|X|}H(X_2)) \approx 1.5219 - 0.95097 \approx 0.57093$

3.3 Decision Tree Algorithms

In this chapter, we will talk about various algorithms such as ID3, C45, and CART. We will analyze their pseudocodes and complexities.

3.3.1 ID3

The ID3 (Iterative Dichotomiser 3) algorithm was introduced by J. Ross Quinlan in 1986 [Qui86]. ID3 is used to create a decision tree from a given dataset by choosing the best local feature to split. Therefore, it is a greedy approach. The best local feature is calculated using information gain. ID3 calculates the information gains of each feature at each node (beginning at the root node with the entire dataset) and chooses the one with the most. Then, it splits the dataset and continues recursively until a leaf node is returned. ID3 stops, if all samples of a node (subset) belong to one single class or if there is no feature left, and that node becomes a leaf node. In that case, that leaf node is labelled with the most common class among instances in the subset. ID3 can not guarantee an optimal solution for all instances, since it uses a greedy approach. ID3 can also overfit data. ID3 is hard to use with continuous data due to the difficulty of finding the optimal threshold value. The pseudocode of the ID3 algorithm is given in Algorithm 1.

Algorithm 1 ID3 Algorithm for Classification Trees

Input: Dataset D, Feature set F
Output: Decision Tree T
if all examples in D have the same class **then**
    return a leaf node with a class label
end if
if F is empty **then**
    return a leaf node with a majority class label in D
end if
$A \leftarrow$ feature in F with highest Information Gain
if A is continuous **then**
    Determine the best split point s for A
    Split D into D_l and D_r based on s
    if D_l or D_r is empty **then**
        return a leaf node with a majority class label in D
    else
        $T \leftarrow$ Create decision node that splits on $A \leq s$ and $A > s$
        $T_l \leftarrow \text{ID3}(D_l, F - \{A\})$
        $T_r \leftarrow \text{ID3}(D_r, F - \{A\})$
        Add T_l and T_r as children of the decision node
    end if
else
    for each value v of A **do**
        $D_v \leftarrow$ subset of D where $A = v$
        if D_v is empty **then**
            Add a leaf node with a majority class label in D
        else
            $T_v \leftarrow \text{ID3}(D_v, F - \{A\})$
            Add branch T_v to T
        end if
    end for
end if
return T

Let us analyze the time complexity of ID3. Entropy calculation of a dataset can be done in $O(n)$, where n is the size of the dataset. Therefore, the calculation of information gain for one single feature is also in $O(n)$. To calculate the best information gain, we have to iterate over each feature. If m is the number of features, we can calculate the information gain (in the worst case) in $O(nm)$ for one single node. We could save calculated entropies to avoid calculating the same entropy again by using dynamic programming. But in the worst case, it does not matter. The constructed tree with maximum branching factor b can have at most $b^0 + b^1 + ... + b^l = \sum_{i=0}^{l} b^i = \frac{1-b^{l+1}}{1-b} = \frac{1-b^{\log_b(n)}}{1-b} = \frac{1-b \cdot n}{1-b} \in O(nb)$ nodes where $b^l = n$ since all nodes in last level are leaves and b is the number of possible values of the feature which has maximum possible values among all features. So, we can calculate the information in $O(nm)$ for one single node, and we have at most $O(bn)$ nodes. Therefore, the algorithm performs all information gain calculations in $O(n^2mb)$. Since all other operations are possible in constant time ($O(1)$), the algorithm runs in $O(n^2mb)$ at worst case.

3.3.2 C4.5

C4.5 algorithm is an improvement of ID3, also introduced by J.Ross Quinlan [Qui93]. The most important improvements can be listed as:

- Handling both continuous and discrete data. C4.5 can find optimal threshold values, whereas ID3 cannot do that. In ID3, usually, the mean value is taken with continuous attributes.
- Handling missing feature values. C4.5 can recognize missing data and is able to perform the splits according to non-missing data.
- After the tree is constructed, C4.5 performs post-pruning and simplifies the tree by removing redundant branches and leaves to avoid overfitting.
- C4.5 measures the accuracy of each node.
- C4.5 uses information gain ratio instead of information gain. The gain ratio is a normalization of information gain, which avoids the extreme advantage of features with more values.

3.3.3 CART

CART (stands for Classification and Regression Trees) is an algorithm introduced in 1984 by Leo Breiman et al. [BFOS84] to construct decision trees. The CART algorithm constructs only binary trees. That means, at each node, the dataset is divided into two subsets. In contrast to ID3 and C4.5, CART can used for both classification and regression problems. In classification problems, the Gini index or entropy can be used to find the best split points. In regression problems, usually, the sum of squared errors, mean squared error(MSE) and mean absolute error are used. The pseudocode of the CART algorithm is given in Algorithm 2.

Algorithm 2 CART Algorithm

Input: Dataset D with N samples, each sample has M features
Output: A decision tree
if stopping criteria is met **then**
    Create a leaf node and assign a label (classification) or value (regression)
    return the leaf node
end if
Select the best split point (θ, j) for a feature j that minimizes the impurity (Gini index for classification, variance for regression)
Split the dataset D into two subsets D_{left} and D_{right} based on the best split point
Create a decision node that splits on feature j and threshold θ
if D_{left} is pure or small enough **then**
    Create a leaf node and assign a label or value
else
    $T_{left} \leftarrow \text{CART}(D_{left})$
end if
if D_{right} is pure or small enough **then**
    Create a leaf node and assign a label or value
else
    $T_{right} \leftarrow \text{CART}(D_{right})$
end if
Attach T_{left} and T_{right} to the decision node
return the decision node

The worst case time complexity of CART is in $O(pN^2 \log N)$ [Lou14], where N denotes the number of samples, p the number of input variables, and K the number of variables randomly drawn at each node.

3.4 Overfitting

Overfitting is a main problem that can occur in decision trees. It causes the model to have a very high performance predicting within the learning set, but a bad performance with unseen

instances. Causes of overfitting can be listed as:

- Model learns every detail in the dataset. This can happen if the dataset is too small
- The constructed decision tree is very deep and at each leaf node there are too few instances

The decision tree in Figure 2 can be an example of an overfit decision tree since there are too few instances at the leaves (almost 1 instance per leaf) and the tree is too big depending on the dataset. There are some techniques to prevent overfitting such as pruning, setting a maximum depth, setting a minimum sample number per leaf, cross-validation, and ensemble methods which we will dive deeper into in the next chapter.

3.5 Underfitting

Underfitting is also a main problem with decision trees. Same as in overfitting, the model has a bad performance with predicting unseen instances, but also with instances from the learning set. Underfitting can happen because of model is too basic to catch the dependencies between the features, or because the features in the dataset do not represent the dependencies well enough. There is a visualization of underfitting in comparison of good fit and overfitting samples in Figure 8. The X-axis represents the feature's values whereas the Y-axis represents the values of the dependent variable.

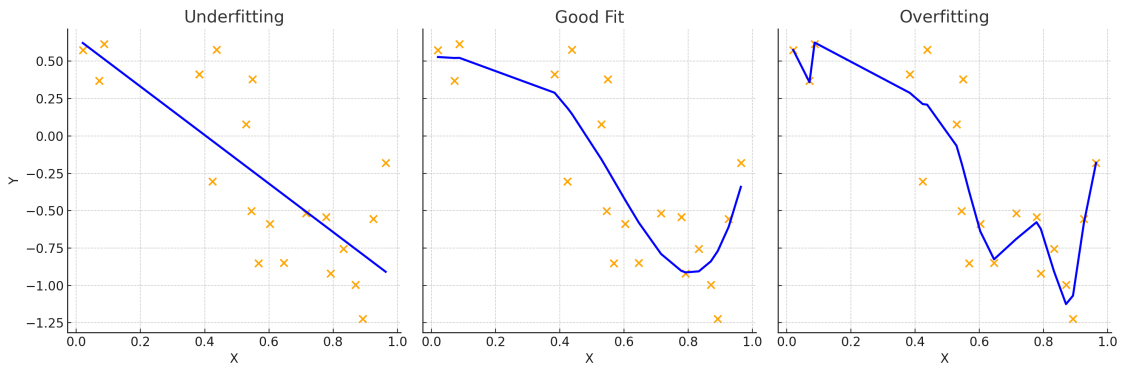


Figure 8: Schema of Underfitting, Good Fit, Overfitting

3.6 Pruning

Pruning is used to reduce the overfitting of decision trees. The main idea is, deleting or merging some branches and nodes of the decision tree. This avoids overfitting and simplifies the tree's complexity. Therefore, the decision tree will have a better performance with unseen data. There are two main types of pruning a decision tree.

3.6.1 Pre-Pruning

This type of pruning is performed during the construction of the tree. The construction of the tree is stopped according to some criteria, this can for example be maximal depth.

3.6.2 Post-Pruning

In contrast to pre-pruning, in post-pruning, we construct the tree as usual. After the tree is completely constructed, we perform pruning on extra-detailed and redundant subtrees. This type

of pruning works better in practice and is also used more frequently. There are different pruning strategies such as accuracy, and cross-validation.

3.7 Summary

Here, we explored various aspects such as the rules and criteria for splitting nodes, including Information Gain, Entropy, and the Gini Index. We also discussed the common problems associated with decision trees like overfitting, underfitting, and the necessity of pruning to maintain model accuracy and simplicity.

4 Ensembles

4.1 Ensemble Learning

Ensemble learning is also called collective learning and multiple classifier systems. In ensemble learning, multiple models are brought together as a board and used to make decisions. By using these basic models together, which do not give very good results on their own, it is aimed to obtain a strong collective learner, a decision maker. In this way, by using multiple individual learners together, higher accuracy rates can be made for classification and regression.

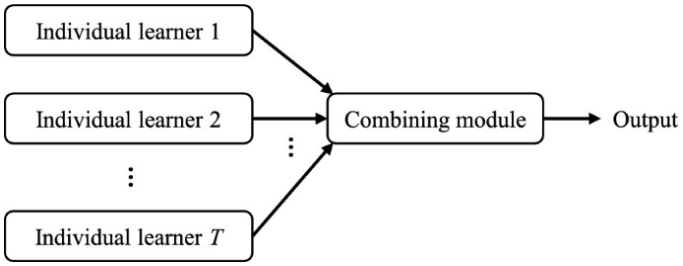


Figure 9: The workflow of ensemble learning [ZZ21]

The workflow of ensemble learning is to first train a group of individual learners and then combine them through some strategy, as shown in Figure 9. If an ensemble consists of the same types, it is said to be homogeneous, e.g., "decision tree ensemble". For homogeneous ensembles, the corresponding learning algorithms are called base learning algorithms and the individual learners are called base learners. In contrast, a heterogeneous ensemble contains different types and different learning algorithms, and there is no single base learning algorithm. Bringing together multiple learners for generalization is certainly stronger than data from just one individual learner, and this is especially true for weak learners. For this reason, base learners are sometimes called weak learners, and the focus of ensemble learning is on these learners [ZZ21].
If we consider the example given in Figure 10, we have three test examples and three individual learners (h_i). If the correct classification is made in the given example, a ✓ mark is placed between that individual learner and the test sample, and if the classification is wrong, an $\times$ mark is placed. It can be seen that the examples and the value given by the classification in Figure 10 (a) give an accuracy of 66.6%, and the ensemble gave a better result and responded to it with 100% accuracy. In the 2nd table, each learner decided the test examples in the same way, and the ensemble did not improve the result. However, when the values given in the 3rd table are taken into consideration, the ensemble makes a worse decision. As can be seen in this given example, in order to be able to say that the ensemble is a good and accurate ensemble, it must contain correct and diverse individual learners.

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✓	✗
h_2	✗	✓	✓
h_3	✓	✗	✓
Ensemble	✓	✓	✓

(a) Ensemble helps.

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✓	✗
h_2	✓	✓	✗
h_3	✓	✓	✗
Ensemble	✓	✓	✗

(b) Ensemble doesn't help.

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✗	✗
h_2	✗	✓	✗
h_3	✗	✗	✓
Ensemble	✗	✗	✗

(c) Ensemble hurts.

Figure 10: An Example for ensemble learning [ZZ21]

4.2 Bootstrap Aggregating (Bagging)

Bootstrap Aggregating, also called bagging, is not a specific algorithm for decision trees, instead is a machine learning ensemble algorithm intended to improve the accuracy of machine learning algorithms used in classification and regression. Bagging, which aims to prevent overfitting and reduce variance, is generally applied in decision tree methods but can be used with any method. The first bagging algorithm was introduced by Breiman in 1996 [Bre96]. This algorithm has brought many advantages since its introduction. One of the advantages of bagging is that it generalizes and improves unstable procedures such as classification and regression trees. Moreover, as Breiman also mentioned in his article [Bre96], it is relatively simple to make an existing method even better thanks to bagging. There are 3 important terms for bagging and these are original, bootstrap, and out-of-bag datasets. The original dataset here represents any collection of information given externally.

The bootstrap dataset is created by randomly selecting objects from the given original dataset. In addition, the size of the bootstrap dataset must be the same as the original dataset. On the other hand, the bootstrap dataset may have duplicate objects and thus differ from the original dataset.

The out-of-bag dataset contains data that is in the original dataset but not in the bootstrap dataset, and also in this case duplicate objects in the bootstrap dataset do not play a role. Since the bagging algorithm selects data randomly, it can select data more than once, so there is a possibility that it may not select some data at all. Therefore, out-of-bag datasets are in an important position for bagging as they are used to test decision trees. As a result, these out-of-bag and bootstrap datasets are created by the algorithm for all decision trees in the forest, as we can see in Figure 11. Moreover, Since the data in the out-of-bag dataset is not applied in that decision tree, an error occurs and this error can be measured with the out-of-bag error. Out-of-bag error is a method of measuring the prediction error of machine learning models utilizing bagging. To calculate the out-of-bag error we will do the following steps:

First, identify all the decision trees that were not trained on the out-of-bag samples. Then, group the trees that share the same out-of-bag samples. These grouped trees will then vote on the classification of their out-of-bag samples, with the majority vote determining the final result. After that, we will compare these majority vote results with the actual true labels. Finally, repeat this process for each set of decision trees with different out-of-bag sample groups.

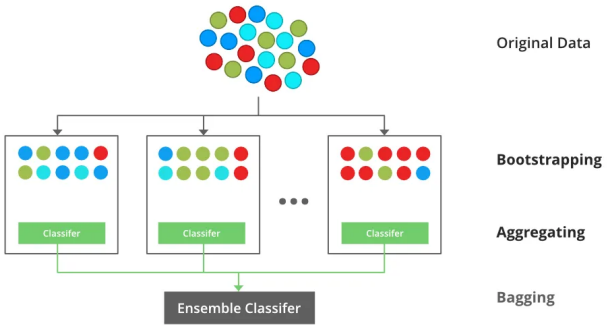


Figure 11: Bootstrap Aggregating(Bagging) [Tan22]

4.2.1 Implementation of Bagging Algorithm

The pseudocode of the bagging algorithm is given below in Algorithm 3 and in this chapter, this code will be explained.

Algorithm 3 Bagging Algorithm

Input:
  Dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$
  Base learning algorithm $\mathcal{L}$
  Number of training rounds T
Procedure:
for $t = 1, 2, \ldots, T$ **do**
  $h_t = \mathcal{L}(D, D_{bs})$
end for
Output: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y).$

First, if we look at the inputs given to us, there is training set D, base learning algorithm $\mathcal{L}$ and Number of training rounds T. To explain these, Training set D consists of pairs of input features x_i and their corresponding labels y_i. This represents the data on which the models will be trained. In addition, the Base learning algorithm $\mathcal{L}$ is the machine learning algorithm used to train each model. For bagging, decision trees are typically used, but any learning algorithm can be chosen. Finally, number of training rounds T specifies the number of iterations or models to train.

The algorithm first iterates T times to create as many different models as possible, where each iteration is a round of training. During each iteration, bootstrap datasets are created from the original dataset D, and the way this is created is completely random and replacement-based. As mentioned above, some data may be used more than once, or not at all. Then, the given base learning algorithm (usually a decision tree) is trained on this bootstrap sample.

After T iterations, we have T times trained models, and the outputs of these models are collected to look at their situation with an input x. Finally, in classification tasks, majority voting is often used with this calculation.

$$H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y)$$

The bagging algorithm brings many advantages. Since this algorithm averages the predictions of multiple models, it provides more stable and reliable predictions and thus reduces overfitting. In addition, this algorithm provides improved accuracy because it trains not just one dataset but multiple datasets and produces an output and prediction accordingly.

Bagging is commonly used in finance, particularly in credit risk assessment. By creating multiple decision trees from different bootstrap samples of the training data, bagging reduces the variance. Each decision tree might evaluate various factors such as credit history, income level, and employment status. The final prediction is then made by aggregating the results from all the trees, leading to a more stable and reliable assessment of the applicant's creditworthiness. This approach helps banks and financial institutions minimize losses.

4.3 Random Subspace Method

This technique called the random space method or attribute bagging, was first introduced by Tin Kam Ho in 1998 [Ho98]. This ensemble algorithm behaves differently from the bagging algorithm in feature selection and its main goal is to select a random subset of features for each base learner. Also, in this algorithm, each base learner is trained on the original training data but using only the selected subset of features. After this training, all predictions are brought together and combined to reach a final prediction of the ensemble.

As with every ensemble algorithm, this algorithm also brings many advantages to the user. Increased diversity or improved performance can be given as examples of these advantages. Compared to a single model trained on all features alone, we obtain a more accurate and robust result with the combination given above and a common final prediction. Also, since we train each learner on a different subset of features, the diversity between the base learners increases with this method, thus improving generalization and reducing overfitting. In addition, this method is useful for high-dimensional datasets where the number of features is very large compared to the number of samples.

Ho's main goal here is to find a way to create multiple decision trees for a forest without creating the same tree over and over again. Ho states a feature space of the size n has a feature subspace of the size 2^n [Ho98]. The dimensionality of the feature subspace enables tree classifiers to independently generalize. In this algorithm, it selects a subspace from the feature subspaces and randomly assigns it to the decision tree, and this assignment process continues until all subspaces are exhausted. In this way, decision trees are trained with the entire training set and will be created differently from each other because they are trained with different feature sets. The following algorithm can be used to build an ensemble of models utilizing the random subspace method:

1. Assume there are N training points and D features in the training dataset.
2. Let L represent the number of individual models within the ensemble.
3. For each model l, select n_l (where $n_l < N$) as the number of input points for model l. Typically, a single value of n_l is used for all individual models.
4. For each model l, generate a training set by selecting d_l features from the D features with replacement and train the model on this set.

After all of that, we are combining the outputs of the L individual models by the majority and it will be the decision of the algorithm. A simple illustration of this ensemble learning algorithm is given in Figure 12.

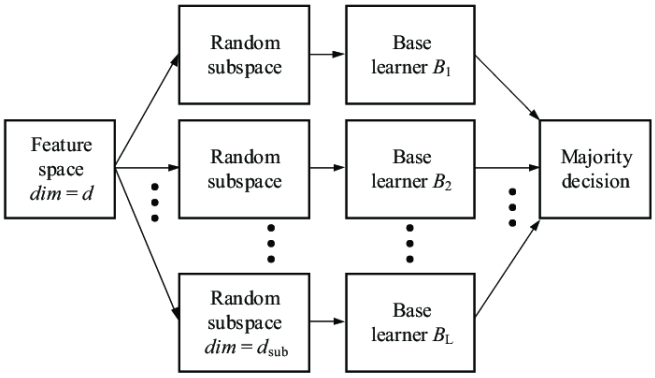


Figure 12: The workflow of Random Subspace Method [res24]

4.4 Random Decision Forests

4.4.1 Introduction to Random Decision Forests

We have mentioned in previous topics that decision trees bring many benefits, but since only one tree is used, they cause problems such as the prediction not being reliable and robust enough or overfitting and bias. To correct these negativities and obtain better and more accurate results, ensemble algorithms and random forest are needed, which is an extension of the bagging method. Also, when multiple decision trees are combined into an ensemble within the random forest algorithm, they produce more accurate results, especially when the individual trees are not correlated with each other.

As mentioned in previous sections, the first algorithm for random forest was written by Tin Kam Ho in 1995 [Ho95] and used the random subspace method. After that, the random forest algorithm, which is developed as an extension of Breiman's bagging algorithm idea, was trademarked by Adele Cutler [CCS12] and Leo Breiman [Bre01], but the proper introduction of random forests was made in a paper by Breiman. As Breiman mentioned in his 2001 article, a random forest is a classification method that comprises a set of tree-structured classifiers $\{h(x, \Theta_k), k = 1, \ldots\}$, where the $\{\Theta_k\}$ are independent and identically distributed random vectors. Each tree in the ensemble votes for the most common class for a given input x [Bre01].

Decision trees for the random forest algorithm have been studied extensively and have been the subject of many articles and experiments over the last 20 years. This well-known algorithm, which is still in use, can easily handle and solve both classification and regression problems thanks to its simplicity, ease of use, and universality. Random forests, which are robust against problems such as overfitting and noise data that can occur in decision trees, provide more accurate and stable predictions by aggregating the outputs of multiple decision trees. In addition, while decision trees consider all possible feature splits, random forests only select a subset of these features, resulting in a low correlation between decision trees. In fact, it builds multiple decision trees using bootstrap samples of the training data and randomly selects a subset of features for each split in the tree.

As we mentioned above, the splitting rules and criteria in decision trees are also an important factor in ensuring that the final prediction of the random forest is accurate and good. Moreover, the algorithms we gave for decision trees can also be used for random forests (C4.5, CART, etc.). Random Forests typically use the same type of decision tree algorithm for all trees in the forest, but it is possible to use different algorithms for constructing the individual trees in a Random Forest.

After random feature selection and then bagging algorithm, many decision trees are generated and a voting process called majority voting occurs. When it comes to prediction, in classification tasks, the final prediction is determined by the most frequently occurring "vote", while in regression tasks, the final prediction is the average of the given "votes".

Although Random Forests have many advantages, they also have some disadvantages worth mentioning. First, random forest algorithms can be slow to process data because they calculate data for each decision tree. In addition, since they process large datasets, it is difficult to store this data and they need more resources. In decision trees, looking at a single tree and interpreting it is simpler than interpreting a random forest and more intuitive for the reader. Although Random Forests provide robust predictions, understanding how the model makes decisions can be challenging due to its ensemble structure.
Random forests are widely used in medical diagnostics, for example, to improve the accuracy of disease prediction. For instance, in the detection of breast cancer, random forests can be employed to analyze patient data, including features such as age, tumour size, and genetic factors. By aggregating the decisions from multiple decision trees, the random forest model can provide a more reliable diagnosis by considering various subsets of data.

4.4.2 Democracy and Voting in Random Forests

In Random Forests, the idea of democracy and voting is so important for boosting the model's reliability and accuracy. There are 2 types of voting: simple and weighted majority voting, respectively. Simple majority voting is generally a more common method for random forest. In this type of voting, all classifiers have equal vote value and equal to 1 and each decision tree uses a single vote. As a result of this voting, the class that gets the most votes becomes the final prediction. Additionally, it is easier to implement compared to weighted majority voting. Here is the basic calculation of simple majority voting [DB19]:

$$\max_{1\leq j\leq k} \sum_{t=1}^{n} d_{t,j}$$

In weighted majority voting, the vote of each decision tree is weighted according to a measure of its reliability. This can be based on the confidence level or accuracy of the tree's prediction. Finally, the vote with the highest weighted prediction becomes the final prediction. [DSLC14]

$$\max_{1\leq j\leq k} \sum_{t=1}^{n} w_t d_{t,j}$$

The weighted majority voting consists of three phases. In the first stage, the decision trees in the training set are trained. Then, in the second stage, the weights of these trees are determined using the validation set. Finally, in the last stage, the weights of the predictions are examined and combined to find the final prediction [DB19]. As can be seen in Table 4, when we look at the results of different forests, we can see that weighted voting gives better results compared to classical, simple majority voting. In this table, random forests were created using the same criteria, which are the Gini index(GDI), twoing, and deviance.

	Majority Vote			Weighted Vote		
	GDI	TWOING	DEVIANCE	GDI	TWOING	DEVIANCE
Breast	**99.335**	99.1416	**99.5708**	**99.514**	99.1416	**99.5708**
Bupa Liver	70.7218	69.7321	**71.4348**	**72.1739**	71.3043	**72.1739**
Habermann	**72.8406**	71.9167	72.5490	72.5490	72.5490	**73.521**
Pima	81.0546	81.9117	**82.0313**	82.0313	81.6406	**82.3**

Table 4: Comparison of Majority Vote and Weighted Vote [DSLC14]

4.5 Summary

This section introduced ensemble methods, with a focus on bagging, subspace sampling, and random decision forests. These techniques help to improve the robustness and accuracy of models by combining multiple trees and leveraging different aspects of the data.

5 Generalization Error, Bias and Variance

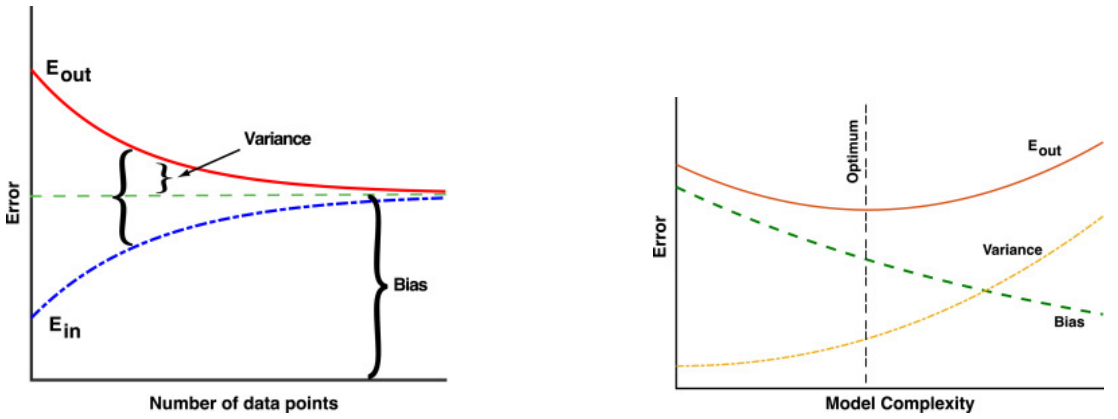
5.1 Generalization Error

Generalization error or out-of-sample error is a metric used to measure the errors of a model with unseen data (test set). The generalization error in random forests is essentially the result of out-of-bag estimation. To minimize generalization error, it is necessary to minimize and avoid overfitting. In regression problems, it is calculated as $\text{GE} = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \hat{y_i})^2$, where y_i is the real value, $\hat{y_i}$ the model's prediction and n the number of instances in test set.

In classification problems, it is calculated as $\text{GE} = \frac{1}{n} \cdot \sum_{i=1}^{n} I(y_i \neq \hat{y_i})$, where $I(y_i \neq \hat{y_i})$ is 1, if $y_i \neq \hat{y_i}$, and 0 if $y_i = \hat{y_i}$.

5.2 Bias-Variance Decomposition

The bias-variance decomposition indicates the accuracy of the predictions given by a model, the complexity of the model, and how accurate its behaviour can be given datasets that have not yet been given and tested. Overall, there is a relationship between the number of adjustable parameters and the ability of a model to better fit the dataset. As the number of parameters increases, the model's fit rate also increases, resulting in fewer errors and lower bias. However, variance in these models also tends to increase in the same way. The aim here is to achieve a balance between bias and variance and to ensure that the model makes the most optimal final prediction. We can clearly see this balance and their actions according to the error in the figures 13 below:



(a) Diagramm of in-sample and out-of-sample error as a function of training dataset size

(b) Bias–Variance decomposition and model complexity

Figure 13: Bias-Variance Balance [MBW+19]

As we mentioned above, bias-variance-decomposition is a method used to detect all errors in the estimates and these problems are divided into 3.

- **Bias error**: Bias error is the error that occurs when the assumptions made by the model are incorrect. High bias leads to the algorithm not fully understanding the relationship between the features and the final predictions, which is the underfitting problem we mentioned earlier.
- **Variance error**: Variance error is the error that shows how much the model's behaviour changes when we give it a dataset that has not been given before. High variance can largely pick up the noise in the training data, which can lead to the overfitting problem, which we mentioned earlier.
- **Irreducible error**: Irreducible error is caused by noise in the data itself that cannot be reduced by any model.

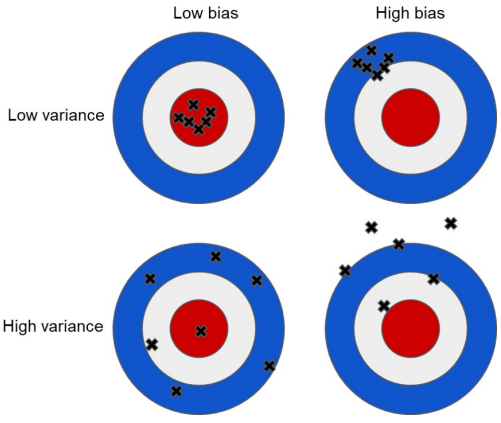


Figure 14: Bias and Variance decomposition [Col]

In Figure 14, the effects of high and low bias and variance on the final estimate are seen. Accordingly, the best effect, that is, the place where the error is least, is in the models with low bias and variance.

Mathematically, this decomposition is expressed as follows [MBW+19]:

$$\text{Bias}^2 = \sum_i \left(f(x_i) - \mathbb{E}_{\mathcal{D}} \left[f(x_i; \hat{\theta}_{\mathcal{D}}) \right] \right)^2,$$

$$\text{Variance} = \sum_i \mathbb{E}_{\mathcal{D}} \left(\left(f(x_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}_{\mathcal{D}} \left[f(x_i; \hat{\theta}_{\mathcal{D}}) \right] \right)^2 \right),$$

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \sum_i \sigma_\epsilon^2,$$

where $f(x_i)$ represents the true function, $f(x_i; \hat{\theta}_{\mathcal{D}})$ represents the prediction made by the model, and σ_ϵ^2 represents the total of irreducible noise. $\mathbb{E}_{\mathcal{D}} \left[f(x_i; \hat{\theta}_{\mathcal{D}}) \right]$ indicates the expected prediction of the model for input x_i when trained on different training datasets $\mathcal{D}$.

In this calculation, the first term is bias and it measures the deviation of our estimator's expectation value from the true value. The second term is variance and it measures how much our estimator is affected by sample effects. If we add these terms with noise, we calculate the out-of-sample error we are looking for.

5.3 Cross-Validation

Cross-validation is another method used to measure how accurate and how error-prone a model is. This method takes different amounts of elements from the dataset instead of taking a fixed amount of elements to test a model in different iterations. This method helps obtain more robust and accurate results. Cross-validation helps reduce the risk of overfitting as a result of these processes and helps us understand how the model will normally perform. There are two types of cross-validation: exhaustive and non-exhaustive. Exhaustive cross-validation methods learn all possible ways to split the original sample into a training set and a validation set. However, non-exhaustive cross-validation does not calculate all possible ways.

5.3.1 Leave-p-out cross-validation

In leave-p-out cross-validation, the dataset is split into two subsets with the cardinality of p and $n - p$. The model is retrained on the remaining data and this process is repeated for all possible combinations of p observations that are removed. So for this Cross-validation method, $\binom{n}{p}$ steps are needed, where n is the total value of the elements of the original dataset [CR08]. This method is exhaustive and provides reliable and accurate performance prediction, but it can be more computationally complex and expensive for large datasets.

5.3.2 Leave-one-out cross-validation

Leave-one-out cross-validation is a special case of the leave-p-out method we just discussed, where p is set to 1. This method requires less computation time than the leave-p-out method since p=1 is taken. Because $\binom{n}{p} = n$ and only n steps are required instead of the $\binom{n}{p}$. However, due to its computational cost, this method is not a preferred method for large samples, and k-fold cross-validation is used more often instead [MSP05].

5.3.3 k-fold cross-validation

In k-fold cross-validation, the original dataset is first divided into k subsets of equal size. Then we do k iterations using one of the k subsets as the validation set and the other $k-1$ as a training set. In each iteration, we use a different subset as the validation set. Therefore, after termination, we will have used each subset once as a validation set. After each iteration, we measure the accuracy.

After the k iterations, we take the average of the accuracies. For example, let us have the dataset $X = \{A, B, C, D, E, F, G, H, I, J\}$ with 10 instances. If we perform a 5-fold cross-validation on this set, we will get the following:

Let us divide the dataset into 5 equal-sized subsets.
$X_1 = \{A, B\}$
$X_2 = \{C, D\}$
$X_3 = \{E, F\}$
$X_4 = \{G, H\}$
$X_5 = \{I, J\}$

Iteration 1:
Training set = $\{C, D, E, F, G, H, I, J\}$
Validation set = $\{A, B\}$
Accuracy = 0.90

Iteration 2:
Training set = $\{A, B, E, F, G, H, I, J\}$
Validation set = $\{C, D\}$
Accuracy = 0.85

Iteration 3:
Training set = $\{A, B, C, D, G, H, I, J\}$
Validation set = $\{E, F\}$
Accuracy = 0.88

Iteration 4:
Training set = $\{A, B, C, D, E, F, I, J\}$
Validation set = $\{G, H\}$
Accuracy = 0.87

Iteration 5:
Training set = $\{A, B, C, D, E, F, G, H\}$
Validation set = $\{I, J\}$
Accuracy = 0.89

Average Accuracy = $\frac{0.90+0.85+0.88+0.87+0.89}{5} = 0.878$

5.4 Summary

The discussion here was centred on the concepts of generalization error, bias-variance trade-off, and cross-validation. The section emphasized how these elements interact and affect the performance of decision trees and random forests.

6 Conclusion

In this report, the basic concepts of decision trees, impurity measures, and decision tree algorithms, which are important topics for random forests, were discussed. We have seen that decision trees are a powerful tool to make predictions in regression and classification problems. Despite the power of decision trees, in most use cases there are some problems and disadvantages such as overfitting. Especially in complex datasets, a decision tree has difficulties in learning the dependence of the features and targets which leads to overfitting. We use random forests for those

cases, introduced by Leo Breiman [Bre01]. We basically create multiple simple decision trees using the same data instead of one complex which would be overfit. Each of these multiple trees uses a different dataset which usually has the same size as the original dataset and is randomly constructed by instances of the original dataset, but can have duplicates (Bagging). We moreover discussed the random subspace method which is an ensemble technique using different subsets of the features for each learner. Random forest is a merge of those two techniques of bagging and the random subspace method. Each of these with bagging and random subspace method created decision trees combine their predictions using some techniques. This provides us with multiple non-overfit decision trees trained with the same original dataset.

Random forests, while powerful, have limitations. They are often difficult to interpret, making it challenging to explain individual predictions in fields like healthcare, finance etc. Additionally, they can be computationally expensive when especially with large datasets. Future research could focus on improving interpretability through better visualization techniques and feature importance analysis. Enhancing computational efficiency and developing methods to handle high-dimensional data more effectively are also important areas.

In conclusion, while decision trees offer a straightforward approach to modeling, random forests provide a more sophisticated and reliable solution by addressing the inherent limitations of single decision trees. The combination of multiple models in a random forest enhances predictive accuracy and robustness, making it a valuable tool in the field of machine learning.

References

[Alp10] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010.

[BFOS84] L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

[Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[CCS12] Adele Cutler, D Richard Cutler, and John R Stevens. Random forests. *Ensemble machine learning: Methods and applications*, pages 157–175, 2012.

[Cha19] Soumo Chatterjee. A deep dive to decision trees. `https://medium.com/analytics-vidhya/a-deep-dive-to-decision-trees-6575e016d656`, 2019. Accessed: 2024-06-15.

[Col] Jason Collins. Heuristics and the bias-variance trade-off. `https://corporate.jasoncollins.blog/heuristics-and-the-bias-variance-trade-off`. Accessed: 2024-06-15.

[CR08] Alain Celisse and Stéphane Robin. Nonparametric density estimation by exact leave-p-out cross-validation. *Computational Statistics & Data Analysis*, 52(5):2350–2368, 2008.

[DB19] Alican Dogan and Derya Birant. A weighted majority voting ensemble approach for classification. In *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6. IEEE, 2019.

[DSLC14] Mostafa El Habib Daho, Nesma Settouti, Mohammed El Amine Lazouni, and Mohammed El Amine Chikh. Weighted vote for trees aggregation in random forest. In *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, pages 438–443. IEEE, 2014.

[DV13] Barry De Ville. Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(6):448–455, 2013.

[Gin12] Corrado Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.]*. Tipogr. di P. Cuppini, 1912.

[Ho95] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[Ho98] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[Kag] Kaggle. Titanic - machine learning from disaster. `https://www.kaggle.com/c/titanic/models`. Accessed: 2024-06-02.

[KS08] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature biotechnology*, 26(9):1011–1013, 2008.

[Lou14] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[LWZ12] Yanli Liu, Yourong Wang, and Jian Zhang. New machine learning algorithm: Random forest. In *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings 3*, pages 246–252. Springer, 2012.

[MBW+19] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[MR05] Oded Maimon and Lior Rokach. *Data mining and knowledge discovery handbook*, volume 2. Springer, 2005.

[MSP05] Annette M Molinaro, Richard Simon, and Ruth M Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. Reprint:Shavlik Dietterich, Readings in ML, 1990.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Elsevier Science, 1993.

[res24] Scheme of ensemble learning: Random subspaces are obtained from feature space dimension. `https://www.researchgate.net/figure/Scheme-of-Ensemble-Learning-Random-subspaces-are-obtained-from-feature-space-dim-and_fig1_317828745`, 2024. Accessed: 2024-06-15.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[Tan22] Hema Anusha Tangellamudi. Bootstrapped aggregation (bagging). `https://medium.com/@hemaanushatangellamudi/bootstrapped-aggregation-bagging-481f4812e3ea`, 2022. Accessed: 2024-06-15.

[ZZ21] Zhi-Hua Zhou and Zhi-Hua Zhou. *Ensemble learning*. Springer, 2021.