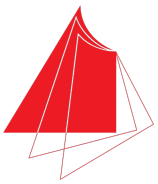

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

University of Applied Sciences Karlsruhe

Random Forest

Tech Report

Timo Blust, 48594
blti1012@hs-karlsruhe.de
B.Sc. Computer Science

Prof. Dr.-Ing. A. Laubenheimer
Winter 2017

Keywords: decision forest, decision tree, machine learning, regression, classification

Contents

Chapter 1

Introduction to machine learning

Nowadays there exist hundreds (or even thousands) of different algorithms that are classified as machine learning algorithms. But what makes an algorithm (or a program) a machine learning algorithm (or program)? Probably the most popular definition of machine learning was provided by Tom Mitchell in 1997: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." [1]. A very popular example of such a program is DeepMind's AlphaGo [2]. AlphaGo is a computer program that learns to play the ancient Chinese board game Go [3]. For the definition above, this means that the task T is playing Go (whether against itself or a human player), the experience E is the data of all previously played games and the performance measure P simply decides if a game is won or lost. If the program is a machine learning program, it has to improve over time (more formal: over the amount of played games). This means the more games the program played, the more likely it is to win its next game.

Every machine learning (ML) algorithm tries to approximate an unknown function

$$f : \mathcal{X} \to \mathcal{Y}$$

The ultimate goal is to find a hypothesis function h that best approximates f.

1.1 Learning types

Machine learning algorithms are divided into four sub-classes:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

1.1.1 Supervised learning

In supervised learning, the learner (that is the training algorithm) tries to approximate a function $f(x) = Y$ from labeled training data. The training data is a set of examples. An example consists of an input x (typically a n-dimensional vector) and its desired output Y (the label) [4]. The accuracy of algorithms that use supervised learning techniques can be measured by the ratio of correct predictions to the total amount of predictions during testing.

A popular example of supervised learning is the task of detecting handwritten digits. The MNIST [5] database contains 70,000 labeled images (28x28 pixel) for training and testing supervised learning algorithms. In this example the input vector x is an (flattened) image and its desired output Y is the digit, that is represented in the image given by x. Popular supervised learning algorithms are support vector machines, linear regression, decision trees and neural networks.

1.1.2 Unsupervised learning

Like in supervised learning, unsupervised learning is the task of approximating a function $f(x) = Y$. However, in unsupervised learning Y (the output or label) is unknown and not given in the training set. Hence, accuracy measurement is also impossible. Algorithms like k-means, generative adversarial networks (a sub-class of neural networks) or anomaly detection are unsupervised [6].

1.1.3 Semi-supervised learning

As the name suggests, semi-supervised learning combines the two methods mentioned previously. Semi-supervised learning tries to accomplish supervised learning tasks with a training set of which the majority of the data points are unlabeled [7]. Typically, labeling data requires expert knowledge and time which can lead to high costs in real world applications. Semi-supervised learning splits in two classes:

- transductive learning: the algorithm tries to find the labels for all unlabeled examples in the training set only,
- inductive learning: the algorithm tries to learn a general rule that can map the entire input set to the output set.

1.1.4 Reinforcement learning

Unlike any of the previously mentioned methods, reinforcement learning (RL) does not require a training or testing data set at all [8]. In RL, the agent (that is the specific RL algorithm) has three major differences to the previously mentioned machine learning techniques:

- it does not require any training or testing data,
- it interacts with its environment (meaning it can influence its own input data),
- it is a reward based system (meaning it can be rewarded/punished for good/bad decisions).

Learning is achieved by applying the agent to its environment. The task of the agent is then to decide which action (from the set of all possible actions) should be executed at the current circumstances (the state of the agent itself, as well as the state of the environment) in order to maximize the total future reward. After this action has been executed, the environment forwards the new state and the current reward back to the agent. This agent-environment interaction loops until a goal state is reached [9]. Games are usually good environments for RL agents. An agent is trained to play a certain game. The game reports its state (for example: how many enemies are there and where are they, are there any obstacles around the player, ...) and the current reward (for example: the player's score) to the agent. Now the agent has to choose an action for the next step/iteration of the loop (for example: move left/right/top/bottom, jump, hide, ...). This action is then applied to the environment. The goal state is reached, if the game ends (for example: all enemies are defeated, all coins are collected, time is over, ...).

1.2 Classification and regression

Supervised learning tasks can be divided even further into the categories *classification* and *regression*. The difference between the both classes is the kind of prediction they make [10].

In *classification* the output (prediction) is categorical. The MNIST [5] data set, for example, addresses a classification problem. The classifier (that is the function that is used to classify or label the data) takes an image as an input and returns the most likely category (digits from 0 to 9) of that image. If one passes a random image (that does not contain a digit at all) as an input, like a photo of a mountain, the classifier will still categorize the image as some digit (whether this makes sense or not).

In *regression* the output (prediction) is continuous. Predicting the market price of a house, for example, is a regression problem. The regressor (that is the function that produces the output value) takes some sort of input data (like price history and data about the property market) and predicts the house's price in a year. The price could, theoretically, be any number and is therefore continuous.

Chapter 2

Decision trees

The basic elements of random forests are decision trees (DTs) (like in nature: a collection of trees is called forest). "A tree is a collection of nodes and edges organized in a hierarchical structure [...]. Nodes are divided into internal (or split) nodes and terminal (or leaf) nodes." [11, p.6]. *Decision* trees are used to make decisions about certain criteria.

In a decision tree, each split node represents a decision (or test) and each edge the outcome of the previous decision. Figure 2.1b shows an example decision tree. This tree tries to classify the shape of a given geometry (from a binary image, for example). The decision making starts at the top (root) node and continues level-wise (top-down) until it reaches a leaf node.

How does a decision tree know which decision belongs to the root node and which decisions have to follow which outcomes (from previous decisions)?

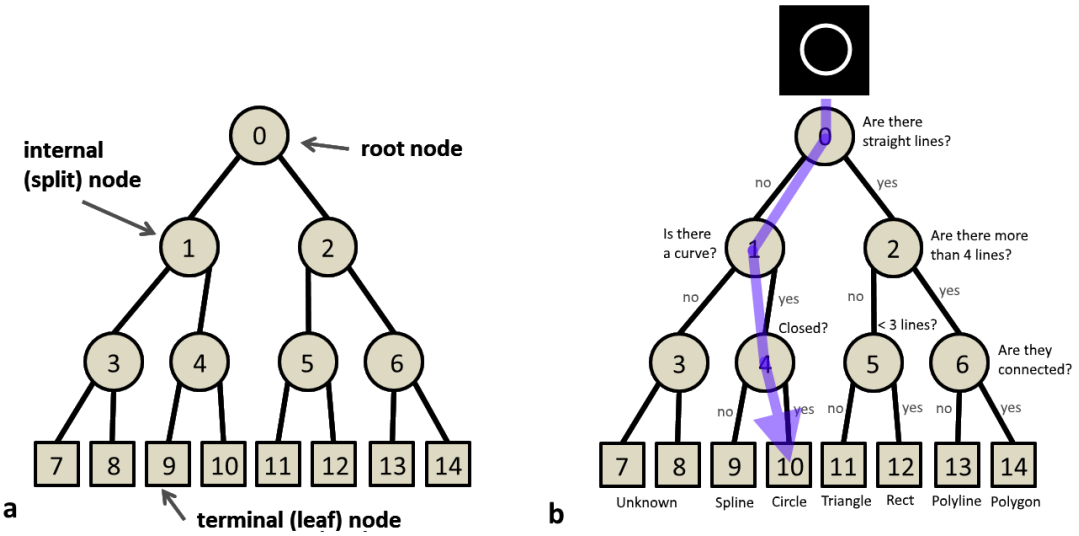

Figure 2.1: **Binary decision tree. (a)** Trees are hierarchical structures consisting of nodes and edges. In contrast to graphs, trees do not have loops. **(b)** Example decision tree that detects primitive geometries in an image.

Source: [11, p.6]

2.1 Example: toy production line

For an easier understanding of decision trees an example environment is introduced.

Imagine a company that produces toys (like action figures). At the end of the production process, a large conveyor belt has to transport all the different toys to their place in the warehouse. The company did not tag the toys when they were produced (with a bar code or similar) and now the automatic storage system cannot distinguish the toys from each other. To resolve this problem, some sort of classification system is attached to the conveyor belt. This classification system is

Figure 2.2: **Toy production line.** The toys produced by the company in the example.

Sources:
`https://images-na.ssl-images-amazon.com/images/I/71gmRJIhsTL._SL1500_.jpg`
`https://lumiere-a.akamaihd.net/v1/images/file_5a756e76.jpeg?width=1200&region=0%2C0%2C2000%2C2000`
`https://cdn.bmstores.co.uk/images/hpcProductImage/imgFull/294814-Batman-Action-Figure-front-20-inches1.jpg`
`https://smedia.webcollage.net/rwvfp/wc/cp/23821006/module/hasbrous/_cp/products/1453393627898/tab-8a4603b5-371b-4783-b3e9-da0c5c74b6bd/1f59651f-5d1d-41cf-a1c0-f5ceb275fef8.jpg.w960.jpg`
`https://target.scene7.com/is/image/Target/51328706?wid=520&hei=520&fmt=pjpeg`
`https://lumiere-a.akamaihd.net/v1/images/image_d616bd0a.jpeg?width=1200&region=0%2C0%2C2000%2C2000`
`https://images-na.ssl-images-amazon.com/images/I/71cmLZFzM2L._SL1500_.jpg`
`https://images-na.ssl-images-amazon.com/images/I/71JSPfGWxoL._SL1500_.jpg`

now responsible for telling the storage system of which type each toy is, so that the storage system can store the toys at the right places in the warehouse. The production line provides the eight different toys given in fig. 2.2.

A top of the art computer vision system is integrated in the classification system and feeds it with different features (gained from multiple cameras that take pictures of each toy on the conveyor belt). The computed features are:

- body shape $x_1 \in \{humanoid, unknown\}$
- main color $x_2 \in \{red, blue, green, brown, gray, yellow, purple, white, black\}$
- accessory 1 $x_3 \in \{cape, hammer, shield, mask, net, nothing\}$
- accessory 2 $x_4 \in \{cape, hammer, shield, mask, net, nothing\}$
- height $x_5 = [1, 10]$ (cm)

The classification system has then to decide (based on the given features) of which type each toy is. A decision tree is used as the decision maker. Valid decisions Y are *Superman, Thor, Batman, Iron Man, Spiderman, Groot, Captain America* and *Hulk*

Before the DT can be used, it has to be trained with a set of training data. Because a decision tree is a supervised learning model, the training set must be labeled (see section 1.1.1). An example in a training set is denoted as

$$\mathbf{s} = (\mathbf{x}, Y) = \{x_1, x_2, ..., x_d, Y\}$$

body shape x_1	main color x_2	accessory 1 x_3	accessory 2 x_4	height x_5	type Y
humanoid	red	net	mask	9	Spiderman
humanoid	blue	cape	nothing	10	Superman
humanoid	gray	cape	hammer	9	Thor
humanoid	gray	mask	cape	10	Batman
humanoid	red	mask	nothing	10	Iron Man
humanoid	brown	nothing	nothing	6	Groot
humanoid	blue	net	mask	10	Spiderman
humanoid	blue	mask	nothing	9	Spiderman

Table 2.1: **Sample training set.** A possible (part of a) training set for the decision tree in the toy company example. x_1 to x_4 are discrete attributes while x_5 is continuous.

where d is the dimensionality of the input vector $\mathbf{x}$ (5 in the toy company example) and Y the desired label (or class) of an input vector. Each feature (or attribute) in the input vector $(x_1, ..., x_d)$ can either be discrete or continuous, depending on the training algorithm. In the toy company example there are four discrete $(x_1, ..., x_4)$ and one continuous (x_5) attributes.

For instance, a sample training set (or at least a few entries of it), that could be used to build the tree in fig. 2.1b is given in table 2.1

2.2 Information theory

In order to understand decision tree learning, one has to understand the meaning of entropy and information gain.

2.2.1 Entropy

Entropy is a measure of impurity of a distribution of a random variable. In the case of training a ML model, like a decision tree, the random variable refers to a feature (column) in the training set. The *has curves* feature in table 2.1 for instance, can take one of two values (*true* or *false*) and contains 7 samples. The distribution of this feature is 3 times *true* and 4 times *false*, which means that the variable is fairly even distributed. An even distribution yields a high entropy which indicates that the distribution is impure.

For a discrete random variable $X \in \{x_1, ..., x_n\}$ entropy is defined as

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

$P(x_i) = [0, 1]$ is the probability of X being x_i in a randomly selected sample. The logarithmic part of the equation applies a weight to the probability, depending on the probability itself. Because $\lim_{x \to 0} \log_b(x) = -\inf$ for $x \in \mathbb{R}^+$ and $\log_b(1) = 0$ for any base b, the weight of values with high probability is smaller than the weight of values with low probability. As a result, sparse distributed distribution (only a few values with high probability) yield lower entropies than high distributed distributions. Figure 2.3 illustrates the entropy for different distributions. The entropy of a totally pure distribution (all data points in the distribution have the same value) is 0.

2.2.2 Gini impurity

A similar alternative to entropy is the gini impurity which is defined as

$$\text{Gini}(X) = 1 - \sum_{i=1}^{n} P(x_i)^2$$

The behavior of both, entropy and gini impurity, is identical in the matter of indicating impurity. Both functions fall if the distribution maximizes its purity and rise if the distribution maximizes its impurity. However, there can be a computational advantage of gini due to the replacement of the log function with a simple multiplication [12].

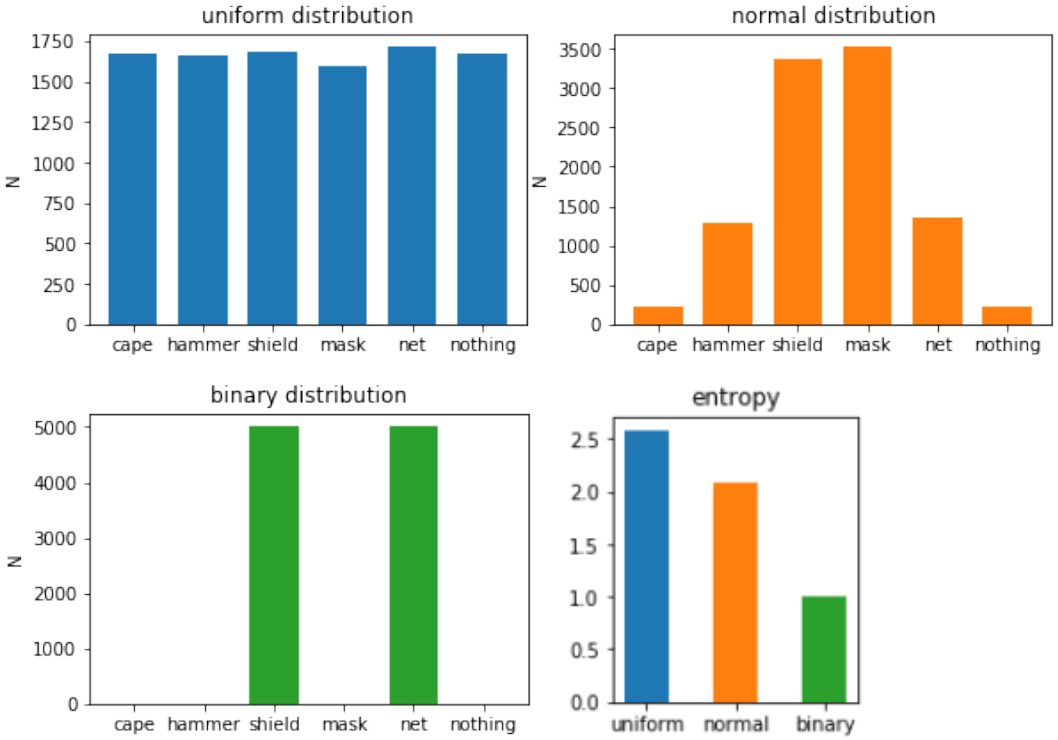

Figure 2.3: **Distributions and their entropies.** Three different distributions (uniform, normal and binary on a set with 10000 samples) of the *accessory* attribute in the toy company example. The bottom right chart shows the entropy for each distribution. The less distributed the set is, the lower is the entropy.

2.2.3 Information gain

Information gain measures how valuable a split of a labeled data set (a set of examples) by a given attribute is (more on splitting data later). The higher the information gain, the better (more pure) the split. Let S be the data set and A the attribute to split on, information gain IG is computed as

$$IG(S, A) = H(S) - \sum_{k \in vals(A)} \frac{|S_k|}{|S|} H(S_k)$$

where

- $H(S)$ (the entropy of a set of examples) refers to the entropy of S reduced to its labels. For instance, let S be the training set shown in table 2.1. $H(S)$ now refers to the entropy over the *type* column.
- $vals(A)$ is the set of all values attribute A can take.
- S_k is the subset of S for which the attribute A is k ($S_k = \{x \in S : x_A = k\}$).

2.2.4 Gain ratio

Some algorithms use gain ratio over information gain to compute the best split (the reasons are explained later). Gain ratio is computed as

$$GR(S, A) = \frac{IG(S, A)}{IV(S, A)}$$

where the intrinsic value (IV, sometimes referred to as *SplitInfo*) is computed as

$$IV(S, A) = -\sum_{k \in \text{vals}(A)} \frac{|S_k|}{|S|} \log_2 \frac{|S_k|}{|S|}$$

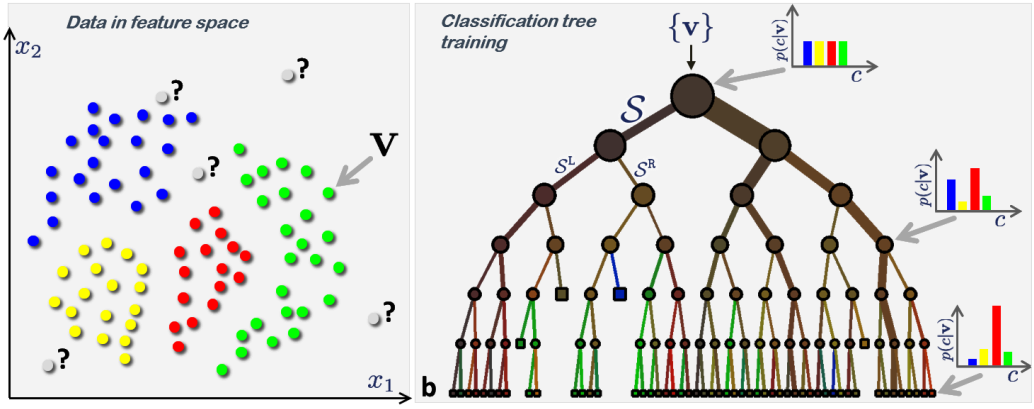

Figure 2.4: **Tree training on 2D data.** The plot on the left hand side shows the training data set with two dimensional feature space. Each color denotes a different class. The gray points are test points (unclassified). The classes are uniformly distributed, meaning each class has the same number of data points. The root node of the tree covers the whole feature space (which can be seen by the histogram at the top on the right hand side). The deeper the tree goes, the more increases the purity of the different classes in different nodes.

Source: [11, p.22]

2.3 Building a tree

In general, decision trees are built iteratively, starting at the root node. A tree is trained by finding a split criterion for each node in the tree (except the leaf nodes). A split criterion is a function that splits the training set into subsets based on one or more attribute values. It can be seen as a question with a finite number of answers. How the split criterion is computed depends on the tree algorithm used. However, the overall goal of splitting is to get pure subsets, meaning that some subset $S_n \subset S$ contains only data points with the same label. After a split criterion for a certain node is computed, child nodes are created according to the possible outcomes of the split. For instance, a possible split criterion of the training set in table 2.1 is splitting by *main color*. The outcomes of that split would be *red, blue, gray* and *brown*. A new child node is created for each outcome. Each child node then obtains a subset of the parent's example set. This subset is also split by the same split criterion. The whole process is repeated for each node until a termination criterion is fulfilled (the set to split on is pure or a threshold in tree depth is reached, for instance). If the termination criterion is fulfilled, a leaf (or terminal-) node is created. This leaf node then represents one possible overall outcome of the decision tree. Depending on the type of tree (classification or regression), the leaf node predicts either a label (classification) or a real value (regression). Figure 2.4 illustrates, how splitting increases the purity of the sets.

Leaf nodes in classification trees always predict the most occurring label from the subset of that node. As shown in fig. 2.4 the remaining subset of the leaf node at the very bottom right contains almost only red labels. This node will always predict *red*. However, leaf nodes in regression trees will always predict the mean of the remaining subset.

This work covers C4.5 and CART, two major DT algorithms. The general working process from above applies to both of these algorithms. However there are some differences between them which are shown in the next section.

2.3.1 C4.5

C4.5, developed by R. Quinlan in 1993 is the successor of ID3 (R. Quinlan, 1986). It can handle discrete, as well as continuous attributes and is only applicable to classification problems. While ID3 relies on information gain to derive a split criterion, C4.5 uses gain ratio. It is shown that information gain tends to prefer attributes with more outcomes, because it generates smaller subsets which tend to have lower entropy [13, p.4]. Gain ratio was developed to counter this behavior by taking the subset's size into account (see section 2.2.4).

Discrete attributes

For discrete attributes, the gain ratio is computed for every possible split. The split with the highest gain ratio is taken. After a discrete attribute is used for a split, that attribute can be removed from the new subsets. Splitting by a discrete attribute results in subsets that are pure regarding this attribute. This means that the information gain for this attribute falls down to 0 after the split.

Continuous attributes

Continuous attributes are handled by searching for the optimal threshold value among all values in the training set. A continuous attribute A is always split into two subsets by finding a threshold value t from the training set that splits the set into $S_1 \cup S_2 = S$ so that $S_1 = \{x \in S : x_A \leq t\}$ and $S_2 = \{x \in S | x_A > t\}$ where $t \in S_A$ (S_A refers to all values of attribute A in set S). Unlike discrete attributes, continuous attributes can occur multiple times in a branch of a tree. For instance, assume a continuous attribute *age*. A node splits the age at 20 to split teenagers and adults while a child node can split the age once more at 60 to split off seniors.

2.3.2 CART

First mentioned in 1984 CART (or *classification and regression tree*) is a binary tree [14]. A binary tree is a tree whose internal nodes always have two child nodes. Non-binary splits tend to fragment the training set too quickly leaving insufficient data for the next level in the tree [15, p.311]. Unlike ID3 or C4.5, CART uses gini impurity instead of entropy.

Discrete attributes

Because CART does only binary splits, discrete attributes have to be handled differently than in C4.5. For instance, lets define an attribute $color \in \{red, green, blue\}$. Instead of splitting the training set into three chunks, CART splits by a subset of the attributes' values. The subset with the highest gain ratio is then selected as split criterion. Possible splits over *color* are

- $\{red, green\}$ and $\{blue\}$
- $\{red, blue\}$ and $\{green\}$
- $\{blue, green\}$ and $\{red\}$

Continuous attributes

Splitting continuous attributes works the same way as in C4.5 (see section 2.3.1).

Regression

Regression problems are problems where the target variable is not a class but a continuous value. Because continuous variables (like $x \in \mathbb{R}$) have an infinite number of possible values (because their associated set is infinite), neither entropy nor gini impurity are useful as they would discretize the target variable. Instead variance can be used to compute the maximum information gain. Variance is calculated as

$$\sigma^2(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{X})^2$$

where N is the size of the training set and $\overline{X}$ the mean of the target variable X over the training set. In the case of DT learning, variance can be computed with the formula above (although it differs from the actual definition of variance), because it iterates over a sample set (the training set) [16, eq.4]. Information gain can be adopted to

$$IG(S, A) = \sigma^2(S) - \sum_{k \in \{L,R\}} \frac{|S_k|}{|S|} \sigma^2(S_k)$$

where entropy is replaced with variance and k indicates either the left or right child node (because CART always builds a binary tree).

Another difference to classification trees is the predictor in the leaf nodes. Using the value with the highest probability in the remaining set at the leaf node makes no sense because the result is continuous. Instead, the mean of the remaining set (over the target variable) is used.

2.4 Performance measurement

To measure the performance of a DT (in terms of prediction accuracy) *cross-validation* is one of the most common techniques. In the simplest way the original data set is split into two partitions (*train* and *test*). The first partition is then used as training set for the algorithm. Once the model is trained, the second partition is used to verify it. The model (in this case) is a DT. The verification step computes the accuracy of a model by these scheme:

1. Feed each data point from the testing set to the model and check if the final prediction matches the label (or value) of that data point.
2. Divide the number of correct predictions by the total number of predictions (the size of the testing set).

As stated in section 1.1.2, cross-validation is only applicable to supervised learning tasks, because it requires labeled data points.

2.5 Overfitting

A common problem of DTs is overfitting. Overfitting occurs when the training set is getting too small. Because the training set shrinks with increasing number of levels, the more levels a tree has, the more likely it is to overfit. Overfitting means, that the rules constructed in a tree are too specific and suited to the training set. This causes a very high accuracy ($\sim 100\%$) in the training set but it will perform poorly on unseen data.

Because overfitting is not a huge problem in random forests (the reason is explained later), techniques to avoid overfitting are not discussed in detail here. There are three ways to avoid overfitting:

- Depth limit: limit the maximum level of a tree.
- Minimum information gain: define a threshold for information gain. If the information gain falls below this threshold create a leaf node instead of splitting further.
- Pruning: Construct a fully fledged tree. After construction walk the tree up from the leaf node to the root node and examine each node if its rule actually improves the accuracy. If not, replace the sub-tree (up to the examined node) with a leaf node predicting the most probable value.

Chapter 3

Random forests

Random forests, also known as decision forests, fall under the category of ensemble learning algorithms. It is shown that random forests outperform decision trees (in terms of accuracy) [17]. But what is random forest, what makes it more powerful than decision tree and why is that?

To answer these questions, the following chapter examines the most important aspects of random forests.

3.1 Ensemble learning

Zhi-Hua Zhou, professor at the Nanjing University in China, defines ensemble learning in [18] as:

> Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn one hypothesis from training data, ensemble methods try to construct a set of hypotheses and combine them to use.

A *learner* refers to some sort of learning algorithm, for example C4.5 or CART (but also non-DT algorithms like neural networks). Solving the same problem means approximating the same unknown function f (as explained in chapter 1) with multiple algorithms.

An ensemble is set of learners (so called *base learners*). The advantage of ensemble learning is its ability to generalize way better than its base learners. Generalization is the most central concept in ML. It describes the ability of an algorithm to abstract rules and patterns from data, so that these rules and patterns also apply to unseen data.

Base learners are often referred to as *weak learners*, as their ability to predict the correct result is only slightly better than random guessing. Ensemble learning combines a set of weak learners to a *strong learner* which can make very accurate predictions. However, base learners are not necessarily weak learners. A well trained decision tree, for example, can also predict with high accuracy.

Ensemble learning usually uses one algorithm as base learning algorithm (like C4.5 or CART) to produce homogeneous base learners. However, there exist several methods that use different base learning algorithms and thus producing heterogeneous base learners [18].

An advantage of ensemble learning algorithms is the higher robustness to overfitting. An overfitted base learner will not harm an ensemble as much, because there exist other base learners which can compensate the overfitting.

3.2 Construction

In general, random forests (as well as other ensembles) are constructed in two steps. First, construct a number of base learners (DTs in this case). Second, combine those by some scheme to produce an overall output. In order for random forests to generalize well, accuracy and diversity of base learners are the key factors. Accuracy can simply be measured by *cross-validation*. Diversity in base learners, on the other hand, is not measurable so there is no way of telling how diverse a learner is. However it is possible to induce diversity in the training set through so called *meta-algorithms* bagging, boosting or stacking [18].

A global parameter for random forest is the number of base learners T it contains. It is also referred to as *forest size*.

3.2.1 Bagging

Bagging, abbreviation for *bootstrap aggregating*, is a parallel construction technique. A bootstrap sample is generated by sampling the original training set with replacement (meaning that the same example can occur multiple times). The size of each sample is the same size as the original set. A base learner is then trained on each of the subsets. After all learners are trained, they are combined by majority voting (classification) or averaging (regression) [18].

> **Majority voting** combines a set of base learners to a single strong learner. Given an input vector $\mathbf{x}$, each of the combined learners predicts its outcome for $\mathbf{x}$. As an overall outcome (the final prediction) the most predicted class (over all base learners) is selected. A more general (and formal) definition of (weighted) majority voting for classifiers that produce probability estimates is
>
> $$C(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} w_t P_t(y|\mathbf{x})$$
>
> where
>
> - $\mathbf{x}$ is the input data point,
> - $\mathcal{Y}$ is the set of the target variable (the label or class),
> - T is the number of learners (trees),
> - w_t is the weight of learner t (can be omitted or set to 1 for non-weighted majority voting) and
> - $P_t(y|\mathbf{x})$ denotes the probability of predicting y for input $\mathbf{x}$ using learner t.

Bagging is (by intuition of the author) the most used randomness model for decision forests. The pseudo-code of the bagging algorithm is shown in algorithm 1.

Algorithm 1: Bagging in classification

Input : Training set $D \leftarrow \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$
Base learning algorithm L
Number of trees T

```
for t ← 1 to T do
    D_t ← Bootstrap(D)    % generate bootstrap sample
    h_t ← L(D_t)          % train the base learner h_t
end
```

Output: $H(\mathbf{x}) \leftarrow C(\mathbf{x})$ % majority vote over all base learners

3.2.2 Boosting

Boosting is a sequential construction technique. It is rather a family than a single algorithm. For an illustration, AdaBoost, the most popular boosting algorithm is shown. The core feature of AdaBoost is its use of weights for each example in the training set.

1. Let $t = 1$ and $D = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$ be the training set.
2. Let $\mathbf{w_t} = \{w_1, ..., w_n\}$ be a weight vector for base learner h_t over all examples in D.
3. Train base learner h_t from D and $\mathbf{w_t}$.
4. Measure the error of h_t (by testing).

5. Let α_t be the weight of the learner h_t, determined by its error.
6. Let $\mathbf{w_{t+1}} = \mathbf{w_t}$ and increase the weight of every incorrectly predicted example in $\mathbf{w_{t+1}}$
7. If $t < T$ increment t and goto step 3.
8. Return the majority voting over all learners $\{h_1, ..., h_T\}$ weighted by $\{\alpha_1, ..., \alpha_T\}$.

In the enumeration above, a support for weights in the base learning algorithm is implicated. However, if a base learner does not support weights by itself, the training set D can be sampled according to the weight of each example [18].

Boosting is usually not used in random forests, because of there iterative construction behavior of one tree relying on the previous generation. However, [19] introduces boosting to random forest.

3.2.3 Stacking

Stacking is a partially parallel construction technique. The algorithm has three steps. First, train T first-level individual learners from the same training set but with different learning algorithms. Second, combine those first-level learners in a second-level learner. Train that learner on a training set generated by the first-level learners. The pseudo-code is shown in algorithm 2 [18].

Algorithm 2: Stacking

Input : Training set $D \leftarrow \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$
First-level learning algorithms $L_{11}, ..., L_{1T}$
Second-level learning algorithm L_2

```
for t ← 1 to T do
    h_t ← L_1t(D_t)        % train first-level learner h_t with its associated algorithm L_1t
end
D' ← ∅                     % define second-level training set
for i ← 1 to n do
    for t ← 1 to T do
        z_it ← h_t(x_i)    % use h_t to classify example x_i
    end
    D' ← D' ∪ {((z_i1, ..., z_iT), y_i)}    % append the new example to D'
end
h' ← L_2(D')               % train the second-level learner h'
```

Output: $H(\mathbf{x}) \leftarrow h'(h_1(\mathbf{x}), ..., h_T(\mathbf{x}))$

Stacking is usually not used in random forest, because it already produces a strong learner. However, one could argue that each base learner in a random forest is a stacked learner by itself, resulting in a forest of stacked learners. Analyzing multi-level ensembles is not subject of this work and therefore skipped.

3.2.4 Randomized node optimization

In all the examples shown previously the dimensionality of the input vector (the number of features or attributes) is very low (5, at its maximum). As shown in section 2.3, the introduces DT algorithms evaluate all available features from the training set. This is not a huge problem for examples of low cardinality. But what happens if the dimensionality of the feature space increases and grows to infinity (in the worst case). In such cases randomized node optimization (RNO) comes to the rescue. RNO reduces the dimensionality d of the input vector $\mathbf{x}$ at each node in a tree by randomly selecting d' features, so that $d' << d$. Therefore the learner does not have to examine the information gain of all features but only the features of the reduced input vector $\mathbf{x}'$ [11, p.12]

3.3 Classification forests

Classification forests combine a set of T classification learners to predict a class $y \in \mathcal{Y}$ from an input vector $\mathbf{x} \in \mathcal{X}^n \subseteq \mathbb{R}^n$.

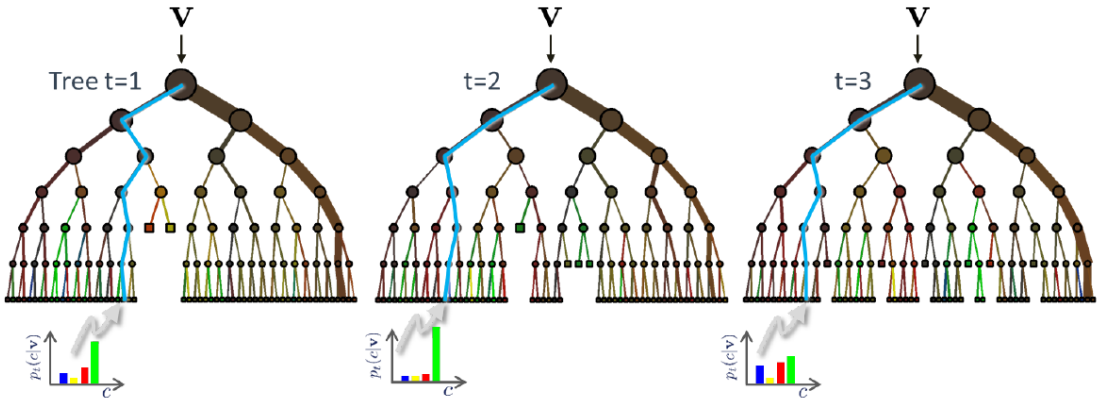

Figure 3.1: **Forest classification.** Predictions from different trees for the same input data point $\mathbf{v}$. Each leaf node in each tree has a different probability distribution $P(c|\mathbf{v})$. The overall outcome of a forest prediction is usually the majority vote.

Source: [11, p.24]

The prediction model. Usually, the final prediction of a classification forest is the majority vote (see section 3.2.1). However, in some cases the output of the forest is not the predicted class, but the probability estimation P itself. In that case, the total probability estimation of a forest is computed by averaging over all learners' probability estimations [11, p.24]

$$P(y|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(y|\mathbf{x})$$

Figure 3.1 illustrates an example forest with three trees.

Randomness. Randomness between learners can be injected by one or more of the techniques mentioned in section 3.2 (most likely a combination of bagging and RNO). Randomness is very important in decision forests as it should lead to a higher diversity within the forest. A higher diversity results in better predictions of unseen data.

The effect of the forest size on generalization. Generalization is a core concept in all ML tasks and therefore a very important factor when designing prediction models. Good generalization means, that a trained model can handle previously unseen data as good as the training data. A very important aspect is noise. How good can the model handle outliers? Figure 3.2 shows a comparison between forests of different sizes. The training set is given by two 2D Gaussian distributions of which each belongs to a class. The base learner of each forest is an axis-aligned weak learner (C4.5, for example). As fig. 3.2c_1 shows, a forest of only one tree produces overconfident predictions. The transition between the classes is very sudden. This is very unintuitive, because a small change in a test point near that transition could cause a change in the prediction from fully confident class A to fully confident class B. A more intuitive way is a smoother transition between the classes, because a test point in the middle of both distributions should result in an uncertain prediction. By increasing the forest size, the transition becomes smoother and smother. This results in more intuitive predictions, because the probability of a test point belonging to one of the classes decreases with distance to the center of mass of the training points.

3.4 Regression forests

Like regression trees, regression forests are used to predict a continuous multi-variate label $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$. The goal of the regression forest is to estimate a probability density function $p(\mathbf{y}|\mathbf{x})$. As before, $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ is the multi-variate input.

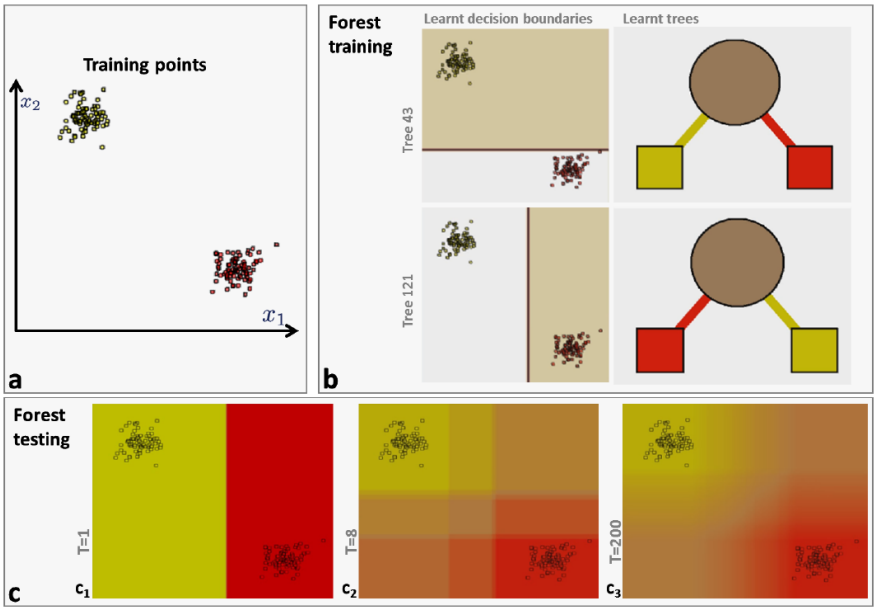

Figure 3.2: **Effect of forest sizes.** **(a)** Training set with two continuous features (x_1 and x_2) belonging to two classes (*red* and *yellow*). **(b)** Different trees produce different leaf predictors. **(c)** By increasing the forest size T, the transitions between the classes are getting smoother.

Source: [11, p.26]

The advantage of regression forests over regression trees is that it splits complex nonlinear problems into smaller sub-problems (linear ones, for instance) which can be handled more easily by simpler models [11, p.50].

Figure 3.3 shows three examples of different predictors in the leaf nodes of a weak learner (like CART). **(a)** shows a constant predictors which return the average over their training subsets (see section 2.3.2). **(b)** shows predictors that perform a polynomial and a linear regression over their training subsets respectively. **(c)** shows predictors that perform a linear regression over their training subsets and return the corresponding probability distribution.

The prediction model. The prediction model of regression forests is very similar to them of classification trees, except that it uses probability density distribution instead of probability

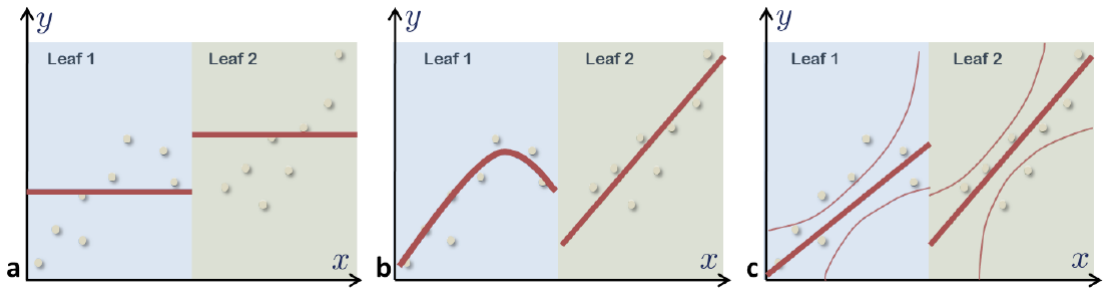

Figure 3.3: **Example predictor models.** Different possible predictor models. **(a)** Constant. **(b)** Polynomial and linear. **(c)** Probabilistic-linear. The latter returns the conditional distribution $p(y|\mathbf{x})$.

Source: [11, p.50]

distribution:

$$p(y|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} p_t(y|\mathbf{x})$$

Randomness. Like in classification, randomness is injected by (most likely) bagging and RNO.

3.5 Other decision forests

3.5.1 Density forests

Unlike classification or regression forests, density forests operate on unlabeled data which makes it an unsupervised learning task. The goal of density forests as defined in [11, p.70] is:

> given a set of unlabeled observations we wish to estimate the probability density function from which such data has been generated.

The base learners of a density forest are clustering trees.

3.5.2 Manifold forests

Manifold forests deal with the reduction of high-dimensional to a much lower dimensionality, while preserving relative distances between data points. Real world data is often described by a very large number of dimensions. However, a closer look at the data often reveals a much lower dimensional underlying distribution. By finding this underlying manifold and "unfolding" it, a maybe easier way to interpret that data can be found.

Manifold forests are described in [11, p.97] as:

> Given a set of k unlabeled observations $\{v_1, ..., v_i, ..., v_k\}$ with $v_i \in \mathbb{R}^d$ we wish to find a smooth mapping $f : \mathbb{R}^d \to \mathbb{R}^{d'}, f(v_i) = v_i'$ such that $d' << d$ and that preserves the observations' relative geodesic distances.

3.6 Real world example

A famous real world example for classification forests is the Microsoft Kinect sensor in fig. 3.4. The task is defined in [11, p.44] as:

> Given a depth image such as the one shown in fig. 3.4a we wish to say which body part each pixel belongs to.

The Kinect sensor has to distinguish between 31 different body part classes. The unit of computation is a pixel $\mathbf{p} \in \mathbb{R}^2$ with corresponding feature vector $\mathbf{x}(\mathbf{p}) \in \mathbb{R}^d$. The feature vector $\mathbf{x}$ is a collection of depth differences between a point $\mathbf{p}$ and an set of offsets $\mathbf{r}$. Because the set of possible offsets is infinite, the dimensionality of $\mathbf{x}$ is also infinite. Figure 3.5 illustrates the classification of a single frame. For further details see [11, p.44].

3.7 Conclusion

This work started with a small introduction to machine learning concepts. Furthermore, decision trees (the base element of decision forests) have been examined extensively. The concepts of building a decision tree has been discussed. The differences between classification and regression trees has been stated and popular algorithms of both classes has been explained. Furthermore, decision forests has been covered by discussing several ensemble learning techniques. A brief overview of several meta-algorithms were given and the concepts of classification and regression forests were examined.

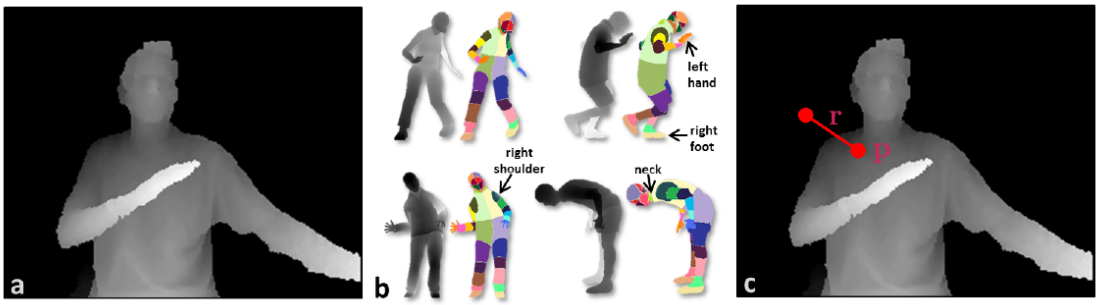

Figure 3.4: **Classification forests in Microsoft Kinect sensor.** **(a)** An input frame as acquired by the Kinect depth camera. **(b)** Synthetically generated ground-truth labeling of 31 different body parts. **(c)** One of the many features of a "reference" point $\mathbf{p}$. Given $\mathbf{p}$ computing the feature amounts to looking up the depth at a "probe" position $\mathbf{p} + \mathbf{r}$ and comparing it with the depth of $\mathbf{p}$.

Source: [11, p.44]

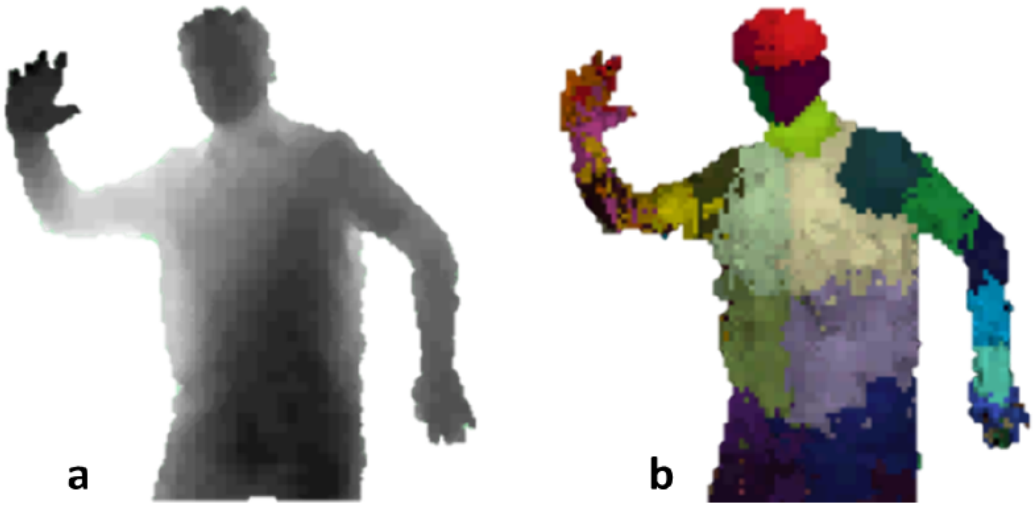

Figure 3.5: **Classification forests in Microsoft Kinect sensor.** **(a)** An input depth frame with background removed. **(b)** The body part classification posterior. Different colors correspond to different body parts, out of 31 different classes.

Source: [11, p.45]

Acronyms

CART classification and regression tree

DT decision tree

ML machine learning

RL reinforcement learning

RNO randomized node optimization

Bibliography

[1] T. M. Mitchell, *Machine Learning*. March 1997.

[2] "Alphago." https://deepmind.com/research/alphago/.

[3] Wikipedia, "Go (game) — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Go_(game)&oldid=808897978.

[4] Wikipedia, "Supervised learning — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Supervised_learning&oldid=809538176.

[5] C. B. Yann LeCun, Corinna Cortes, "The mnist database." http://yann.lecun.com/exdb/mnist/.

[6] Wikipedia, "Unsupervised learning — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Unsupervised_learning&oldid=806744572.

[7] Wikipedia, "Semi-supervised learning — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Semi-supervised_learning&oldid=811431152.

[8] Wikipedia, "Reinforcement learning — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=810477823.

[9] D. Silver, "Rl course by david silver - lecture 1: Introduction to reinforcement learning," 2015. https://www.youtube.com/watch?v=2pWv7GOvuf0.

[10] E. Cai, "Machine learning lesson of the day – supervised learning: Classification and regression," 2014. https://chemicalstatistician.wordpress.com/2014/01/05/machine-learning-lesson-of-the-day-classification-and-regression/.

[11] A. Criminisi, E. Konukoglu, and J. Shotton, "Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning," tech. rep., October 2011. https://www.microsoft.com/en-us/research/publication/decision-forests-for-classification-regression-density-estimation-manifold-learning-and-semi-supervised-learning/.

[12] Wikipedia, "Computational complexity of mathematical operations — wikipedia, the free encyclopedia," 2017. https://en.wikipedia.org/w/index.php?title=Computational_complexity_of_mathematical_operations&oldid=808200458.

[13] E. H. Jr., "Information gain versus gain ratio: A study of split method biases," tech. rep. https://www.mitre.org/sites/default/files/pdf/harris_biases.pdf.

[14] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis, 1984.

[15] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning*. Springer, 2009.

[16] Wolfram-MathWorld, "Variance." http://mathworld.wolfram.com/Variance.html.

[17] S. Cinaroglu, "Comparison of performance of decision tree algorithms and random forest: An application on oecd countries health expenditures," tech. rep., March 2016. http://www.ijcaonline.org/research/volume138/number1/cinaroglu-2016-ijca-908704.pdf.

[18] Z.-H. Zhou, "Ensemble learning," tech. rep., 2009. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/springerEBR09.pdf.

[19] Y. Mishina, M. Tsuchiya, and H. Fujiyoshi, "Boosted random forest," tech. rep., 2009. http://www.vision.cs.chubu.ac.jp/MPRG/C_group/C058_mishina2014.pdf.