Combining Decision Tree Predictions under Consideration of Uncertainty

Bachelor-Thesis von Florian Peter Busch aus Dieburg
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Moritz Kulessa

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Combining Decision Tree Predictions under Consideration of Uncertainty

Vorgelegte Bachelor-Thesis von Florian Peter Busch aus Dieburg

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Moritz Kulessa

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Florian Peter Busch, die vorliegende Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Florian Peter Busch, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: Unterschrift / Signature

_____ _____

Contents

List of Figures

List of Tables

Abstract

The random forest algorithm is a popular classification method in machine learning. Thereby, the prediction of this ensemble model is made by combining decision tree predictions within the forest. Those predictions are made based on the instance numbers and distributions of the respective leafs. Therefore, if those leafs are small (i. e. there are only a few number of instances in that leaf), the prediction might be uncertain and false. This seems to be increasingly problematic for smaller leafs when using the probability as the prediction methods. The purpose of this bachelor thesis is examining existing and introducing new prediction methods which can be used instead of probability and which consider the uncertainty of the leafs based on the number of instances used for the prediction. The evaluation of those methods showed that probability gives bad estimations on single decision trees. However, none of the tested methods managed to achieve significantly better results than the probability in random forest classification. The experiments in this thesis indicate that random forest are not as affected by the uncertainty of small leafs as single decision trees, although it was shown that large trees (i. e. small leafs) on average perform better than smaller trees, when used in random forests.

1 Introduction

Machine learning is a large field in computer science which has many real world applications. One common application for machine learning is classification. Classification describes the process of assigning labels to instances using the information given by an available dataset. There are a bunch of different algorithms and models capable of classification. However, different algorithms usually have different advantages and disadvantages. Due to the increasing popularity and use of machine learning in different fields, it has become important for machine learning techniques to not only achieve results as good as possible but also to be fast, efficient, and, preferably, also humanly understandable. One such machine learning algorithm which is not only capable of achieving a high accuracy in classification tasks but also runs relatively fast is decision tree learning. Using decision trees, you can follow the decision path of the tree to be able to understand why the respective class is predicted. Since decision trees group instances (data points) according to properties of the features (information available about a instance), you can always see which similar instances are used to get to the prediction. If you want to make precise decisions for tasks where there are many, possibly small regions, the size of the decision tree and therefore the number of possible regions (leafs) used to make predictions increases. While this usually helps to distinguish between more types of instances, it can also cause the regions to be very small, in other words, there only a few instances on which the predictions are based on. Because you can almost never make sure that your dataset is free from noise and inaccuracies, making predictions on only a small number of instances might not always be the best solution. If, for example, a prediction is based on only one instance, predicting the class of that instance might be the best prediction you can make but you can not be very certain about that prediction being right. On the other hand, if there are 1000 instances which agree on the same class, it is safe to say that the prediction of that class for any new instance is very likely true.

There exist multiple ways of dealing with this kind of uncertainty. It is possible to build a smaller tree so that every region on which a decision is based on has enough instances and you can be more certain about a decision. On the other hand, this can also reduce the accuracy of the respective model because the model might generalize to the data too much. Another method which, among others, aims to reduce uncertainty and error of single models is ensemble learning. Ensemble learning describes the process of taking a number of models, often of the same type but usually trained differently in at least one aspect, and making a final prediction based on the decision of each model. Thereby, it is often possible to reduce the bias, i. e. the error of a machine learning model resulting from the model itself[1], because the other models are not making the same mistakes. Therefore, at least in theory, the correct decisions more often than not outweigh the false decisions and the predictive accuracy of the ensemble is improved compared to the single model. One such ensemble learning technique commonly used for decision tree learning is random forests. In random forests, decision trees are built using a subset of the available data[2]. Moreover, additional randomness is included in the tree building process itself. Therefore, random forests can combine the decisions of sufficiently different decision trees to usually achieve better classification results than single decision trees. However, even if incorrect predictions of single decision trees within random forests can be outweighed by the ensemble learning process, it nevertheless seems desirable to reduce the weight of such predictions for the ensemble to make even less errors in those cases.

The basic approach of making predictions based on decision trees and within random forests is by calculating the probability of the number of instances per class for each region. Thereby, a small number of instances will in many cases not represent the true, unknown underlying probability. The goal of this bachelor thesis is finding ways to reduce or model the amount of uncertainty, unlike the probability prediction, in order to gain more appropriate predictions. Thereby, aggregated predictions too are supposed

[1] This is in comparison to the variance which is the error resulting from noise or other flaws and inaccuracies of the data.

[2] Some instances might be used multiple times, this process is called bagging.

to be more accurate because it seems to be logical that predictions based on less instances should not be weighted equally to predictions based on a large number of instances. Put another way, this thesis aims to find and evaluate alternative prediction methods which, based on a region in a decision tree, get as input a number of instances with the respective class and the instance numbers itself and return some kind of *prediction value*.
To keep the extent of this thesis reasonable, only binary classification[3] is considered. The prediction methods of this thesis are divided into two families. A *(single) decision tree prediction* describe any prediction which can be applied on a single decision tree. This means that when comparing different prediction values of different decision trees you get a ranking or maybe even some kind of probability estimation of the respective class. In comparison to single decision tree prediction, the other family of prediction methods is *aggregating predictions*. Those describe methods which can only be used in an ensemble because they need multiple classifiers to come up with a final aggregated prediction. One of the more simple examples is taking the average over a number of predictions but there are also more complex method you can think of.
The structure of this thesis is as follows. In Chapter 2, I introduce the general machine learning foundations necessary to understand the following chapters of this thesis. Moreover, background on probability distributions and belief functions is given. All single prediction methods, i. e. prediction methods which can be applied on a single decision tree, are introduced in Chapter 4. Prediction methods which make predictions based on multiple decision trees are described in Chapter 5. Thereby, some aggregation methods in Chapter 5 can be used to combine outputs from methods of Chapter 4. Afterwards, Chapter 6 explains which datasets are used to test the methods and states the setup of the experiments. Chapter 7 shows and interprets the results of those experiments. Final remarks and possibility for future research are given in Chapter 8.

[3] Classification, where there are only two possible labels to choose from. For simplicity, those can always be called *positive* and *negative*.

2 Foundations

This chapter covers the foundations necessary to understand this thesis. First, a general introduction to machine learning in regards to the term "learning" is given (2.1). Afterwards, the basic properties of classification (2.2), evaluation (2.3) and the foundations of error and uncertainty (2.4) are presented. The following sections (2.5, 2.6 and 2.7) in this chapter are necessary foundations for one or more prediction strategies proposed in subsequent chapters. Lastly, information about decision trees (2.8), ensemble learning (2.9) and random forests (2.10) is given.

2.1 Machine Learning

In machine learning, instead of strictly following a predefined behavior as specified by the programmer, the behavior of application software is learned by the machine learning algorithm itself. What "learning" in this context means is well described by the following quote, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997). Thereby, the machine learning algorithm is used to train a machine learning model by learning from the experience in the specified task. But because the algorithm does not simply know which learned behavior is "good" or "bad", a performance measure is necessary to tell the algorithm which behavior is beneficial for the specified task. The idea is that the more experience is available, the higher the performance achieved by the resulting machine learning model, which was generated using the machine learning algorithm, is. Therefore, it is very important to specify a suitable performance measure for the respective task.

2.2 Classification

Machine learning is a big field with multiple different tasks. One such task, which is covered in this thesis, is classification. Machine learning classifiers are built on a (preferably large) data set, which consists of features and classes[1] as experience for the training process. When there are only two classes to be predicted, the process is called binary classification. Besides labeled data where each entry is assigned a class, there is also unlabeled data where each entry only consists of features and the respective class is unknown. Here, the goal of classifiers is to recognize patterns in the labeled data, which therefore is used as training data, to then apply the so learned model to the unlabeled data in order to predict the class of new, unlabeled data. There are many different ways to train a classifier, so a good performance measure is needed to efficiently and successfully complete the training process. However, it is important to evaluate the performance of a classifier based on its performance on new, unseen data instead of the training data. Otherwise, the classifier might just root learn every instance of the training data to achieve perfect predictions on the training set while not scoring great results on new data[2] which would be more useful. But because you can not tell how well a classifier performs on data where the class is unknown, parts of the available labeled data are often used to determine the performance of the classifier. This part of the data is called test data. Then, the accuracy metric can be used to assess the performance of a classifier:

$$\text{accuracy} = \frac{\text{number of correctly predicted instances}}{\text{total number of predictions}}$$

[1] Also "labels" or "class-labels".

[2] This behavior of learning to closely to the training data and thereby achieving worse results on new data is called overfitting. The opposite, not learning enough from the training data, is called underfitting.

Keep in mind that an accuracy of 50% for binary classification is not better than randomly guessing the label for each instance[3]. Therefore an accuracy as high as possible (close to 1) is desirable. However, since data itself can be incomplete or contain too much noise, it can be impossible to achieve perfect accuracy (accuracy = 1) for classifiers based on the given data sets. Which accuracy should be considered "good" or "bad" must then be decided depending on the available training data as well as the application task.

2.3 Evaluation

Another, sometimes more useful tool to measure the performance of a classifier is receiver operating characteristic (ROC) curves. Here, not the predicted classes but the predicted values are considered (for example the probability of the positive class). Now, you can plot those values with regard to the true classes to find out more about the behavior of your classifier on the data. Figure 2.1 shows an example of a ROC curve. There, you notice that a trade-off between the true positive rate (instances which are classified as positive and are actually positive) and the false positive rate (instances which are classified as positive but are actually negative) is shown. Now, only using the base classifier on which the ROC curve is built, you can choose any point of the curve as a threshold for classification. Going back to the probability example, this means that you not only can predict the positive class if the probability is above 50% (negative class otherwise) but you could also predict the positive class if the probability is above 30%. In that case, you would classify more instances as negative, thereby reducing the true positive rate and improving the false negative rate. Another example is classifying every instance as positive. Here, the true positive rate would be 1 but the false positive rate would be 1 as well[4]. All in all, the output values of a classifier are better (regardless of whether a high true positive rate or a low false positive rate is more important) whenever the ROC curve is more close to the top left corner (ideal point, zero error) than to the bottom right corner (worst point, full error[5]). In other words, the area under the curve (AUC) can be used to assess the quality of a classifier. The advantage of the AUC compared to the ROC curve is simply that it makes comparing multiple curves much easier (higher AUC indicates a better classifier performance).

Another helpful tool for the evaluation of classifier performance is cross-validation (Kohavi et al., 1995). When applying cross-validation (CV), or more generally speaking k-fold cross-validation, the data (training data and test data) is split into k equally large portions. Then, $k-1$ portions are used to train the model and 1 portion is used to test the model (for example by calculating the accuracy). This process is repeated k times, so that there has been tests on every portion of the data. For example, if you use k=2 (2-fold CV), you build the model on one half of the data and test it on the other half. Afterwards, you build a new model on the former test data (which hereby turns into the training data) and test it on the former training data (which now is the test data). The average of both results is used as the result of the evaluation.

2.4 Error and Uncertainty

When a model does not perform as well as it was hoped for or even if a model performs completely fine, you still would like to know why the model performs as well as it does. While this question in general is

[3] Furthermore, always predicting one class can actually result in accuracies above 50% if the class ratio of the dataset is not balanced. Therefore, you should always look at accuracy values differently given the problem.

[4] In this case, the classifier would not be very useful because the positive class is predicted all the time.

[5] In fact, you can argue that in case of the bottom right corner you only have to switch the prediction of the positive and negative class to gain perfect performance. This is not wrong and as a result you can also argue that the blue linear line in the middle (Figure 2.1) is the worst possible ROC curve. However, correctly implemented classifiers usually have a performance between random and perfect (not worse than random). Therefore, the original argumentation also makes sense.

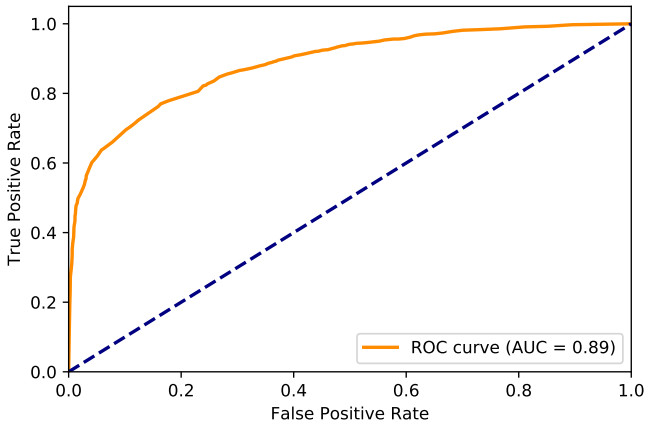


Figure 2.1: Here, an example of a ROC curve is shown (the classifier is a decision tree). The example uses a dataset (magic telescope) and a prediction method ($\text{Prior}_{1,1}$) which are also part of the experiments later. However, the purpose here is only demonstrating how ROC curves look like.

quite complex and difficult to answer, there are also some easier ways to describe or categorize the lack of performance of specific models. First of all, *error* in machine learning is used to describe the mistakes the machine learning model makes. One way to be more specific about the error of a machine learning model is by taking a look at bias and variance. The bias describes the part of the error which is a result of the machine learning model and is high if the model underfits the training data. Contrary to the bias, the variance is used to describe the part of the error which results from the training data itself. Variance in the training set can, for example, be a result of measurement errors or inaccuracies. The variance of a machine learning model is high if that model overfits the training data and therefore also the noise contained in it. Because you want the error in machine learning models to be as low as possible, you want to decrease the sum of bias and variance. Thereby, you have to deal with the bias-variance trade-off. This trade-off describes that models with a low bias often have a high variance and models with a low variance often have a high bias. This makes sense when thinking about the learning process. If the machine learning algorithm only learns very basic properties of the data, the model will probably have a high bias because not enough information of the training data is used but a low variance because noise in the data will probably be mostly or completely ignored by the model. Now, when utilizing more information about the data (maybe by taking more features into account), a more complex model is built. Therefore, this model will represent the data better and therewith have a lower bias. On the other hand, the variance will be more significant because noise in the data will probably not be ignored as much as before. For these reasons, the goal of such machine learning algorithms is to find the best possible trade-off between bias and variance.

Another important aspect related to machine learning mistakes is the concept of uncertainty. For example, imagine two different machine learning classifiers, which are given the task to classify a new instance. For this example, we assume that both classifiers not only return a prediction (the predicted class), but also the probability for the respective class. Now imagine that the first classifier C_1 predicts class a with a probability of 99% and the second classifier C_2 predicts another class b with a probability of 55%. It is clear, that (without any further information) the prediction of C_1 is to be preferred and that it is overall more likely for the new instance to be part of the class a. This example should show that certainty and uncertainty about predictions in machine learning can be very important. Furthermore, you can also define different kinds of uncertainty in order to be able to make even better statements about the quality of a prediction.

One way of doing so is distinguishing between aleatoric and epistemic uncertainty (Hüllermeier, unpublished). Aleatoric uncertainty is randomness which is inherent to the problem or region and can not

easily and most of the time not at all be reduced. Epistemic uncertainty on the other hand refers to a lack of knowledge. Usually, it is possible to reduce the epistemic uncertainty of a problem by gathering more data. Let us consider the coin toss example. If you know the probability of the coin (for example, it is perfectly fair), than there is zero epistemic uncertainty in making a prediction about the outcome. However, there is a high aleatoric uncertainty because both results are possible (equally likely for a fair coin[6]). One the other hand, if you get a new coin and you want to find out if it is fair or what the probability of either side is, then there also is a degree of epistemic uncertainty. Now, you can reduce the epistemic uncertainty incrementally by tossing that coin repeatedly until you can be relatively sure about the probability of getting either result (heads or tails). For decision trees, that translates to smaller leafs having a higher epistemic uncertainty than larger leafs and leafs with a class ratio around 50% having a larger aleatoric uncertainty than leafs with a class ratio around 0% or 100%.

2.5 Probability Distributions

Probability is an important topic in machine learning. Because in most cases it is not possible to achieve perfect machine learning predictions, it is necessary to make statements about the certainty and uncertainty of predictions. This is possible by applying probability theory. In decision tree learning, probability is often used to determine the quality of the prediction and it also can be used to evaluate the performance of a decision tree (you can interpret the accuracy as the probability that the classification of any new instance is correct).
In probability theory, probability distributions are used to describe the probability of an event in an experiment. There are two kinds of probability distributions, discrete and continuous probability distributions. In discrete and continuous probability distributions, the probability mass function (pmf) or the probability density function (pdf), respectively, can be used to determine the probability that an outcome of the modeled experiment takes a specific value or is in a specified range considering the respective distribution. A pmf in discrete probability distributions thereby assigns a probability to every possible outcome (finite) while the pdf in continuous distributions can be used to determine the probability that an outcome lies in an interval[7]. Discrete probability distributions are used when there is a finite number of possible events. Thereby, each event is assigned a probability. For example, the binomial distribution is built based on an event, which can happen, or *be a success* with a probability of *prob*. The distribution is built over a number of tries t. The intuition behind the binomial distribution is, that every trail is independent from each other and is a success with a probability of *prob*. Therefore, to achieve k successes in t tries, which can happen in $\binom{t}{k}$ different ways, there needs to be k successes and $t-k$ failures, which can be calculated with $\text{prob}^k(1-\text{prob})^t$. Hence, the pmf of the binomial distribution is

$$\text{Binom}(k,t,\text{prob}) = \binom{t}{k}\text{prob}^k(1-\text{prob})^t \tag{2.1}$$

So, there can be $s = 0, 1, ..., t$ possible number of successes whereby the number of successes which are close to the probability ($\frac{s}{t} \approx \text{prob}$) are most likely. An example of a binomial probabiliy distribution is shown in Figure 2.2a.
Continuous probability distributions are not based on a finite number of events but on continuous values. When using continuous probability distributions, common use cases are “How likely is the value x smaller than b?” or “How likely is it, that the value x is in between a and b?”. A continuous probability distribution, which will be important for this thesis, is the beta distribution. In the beta distribution, there are two parameters α and β which determine the shape of the beta distribution. The beta distribution can be used to estimate probabilities when the actual probability is unknown. The prior knowledge

[6] In which case the aleatoric uncertainty is maximal.
[7] With an infinite amount of possible outcomes (due to the continuity), every single outcome has a probability of zero. Only intergrating over intervals results in probabilities.

used to estimate the probability is represented by α and β. The higher α and β, the more precisely an actual probability value can be estimated. The pdf on the beta distribution is

$$\text{Beta}(x, \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \tag{2.2}$$

B both in the beta as well as in the beta-binomial distribution denotes the beta function (Abramowitz and Stegun, 1965).
The beta distribution is used in the beta-binomial distribution. Instead of a probability *prob*, the beta-binomial distribution takes two parameters α and β[8] (alongside t) to determine the probability distribution. Thereby, the beta-binomial distribution takes a random probability from the beta distribution. So for an increasing α and β with $\frac{\alpha}{\alpha+\beta} \longrightarrow$ prob, the beta-binomial distribution converges to the binomial distribution. For small values α and β though, the distribution is significantly less peaked. The pmf of the beta-binomial distribution is as follows:

$$\text{Beta-Binom}(k, t, \alpha, \beta) = \binom{t}{k} \frac{B(k+\alpha, t-k+\beta)}{B(\alpha, \beta)} \tag{2.3}$$

Figure 2.2 shows a binomial as well as multiple beta-binomial distributions. You can see, that, even though prob $= 0.6 = \frac{3}{3+2} = \frac{\alpha_1}{\alpha_1+\beta_1}$, the beta-binomial distribution is much steeper than the binomial distribution. This changes noticeably for $\alpha_2 = 6$, $\beta_2 = 4$ and $\alpha_3 = 30$, $\beta_3 = 20$ ($\frac{\alpha}{\alpha+\beta} = 0.6$ for every α, β pair).

2.6 Belief Functions

This part gives the background on belief functions. While most of the theory surrounding belief functions seems to be based on "A mathematical theory of evidence"(Shafer, 1976), we will only explain the basic information about this topic. The following introduction of belief functions is therefore oriented on guided by the information on belief functions given in Quost et al. (2011) and Denœux (2006).
The so called frame of discernment Ω is a finite set which consists of all possible states of belief. The mass function $m : 2^\Omega \rightarrow [0, 1]$ assigns a mass to every subset of Ω. It must hold that $\sum_{A \subseteq \Omega} m(A) = 1$. The mass of any subset A is not necessarily larger than zero but every $A \subseteq \Omega$ with $m(A) > 0$ is called a focal set of m. Mass functions are normalized if $\emptyset$ is not a focal set. If there are no more than two focal sets, one of them being Ω, a mass function is called simple ($m(A) = 1 - w$ and $m(\Omega) = w$ for $A \subset \Omega$ and $w \in [0, 1]$).
Such mass functions m can also be represented by their plausibility and commonality functions which are, respectively, defined as follows:

$$pl(A) = \sum_{B \cap A \neq \emptyset} m(B), \forall A \subseteq \Omega, \tag{2.4}$$

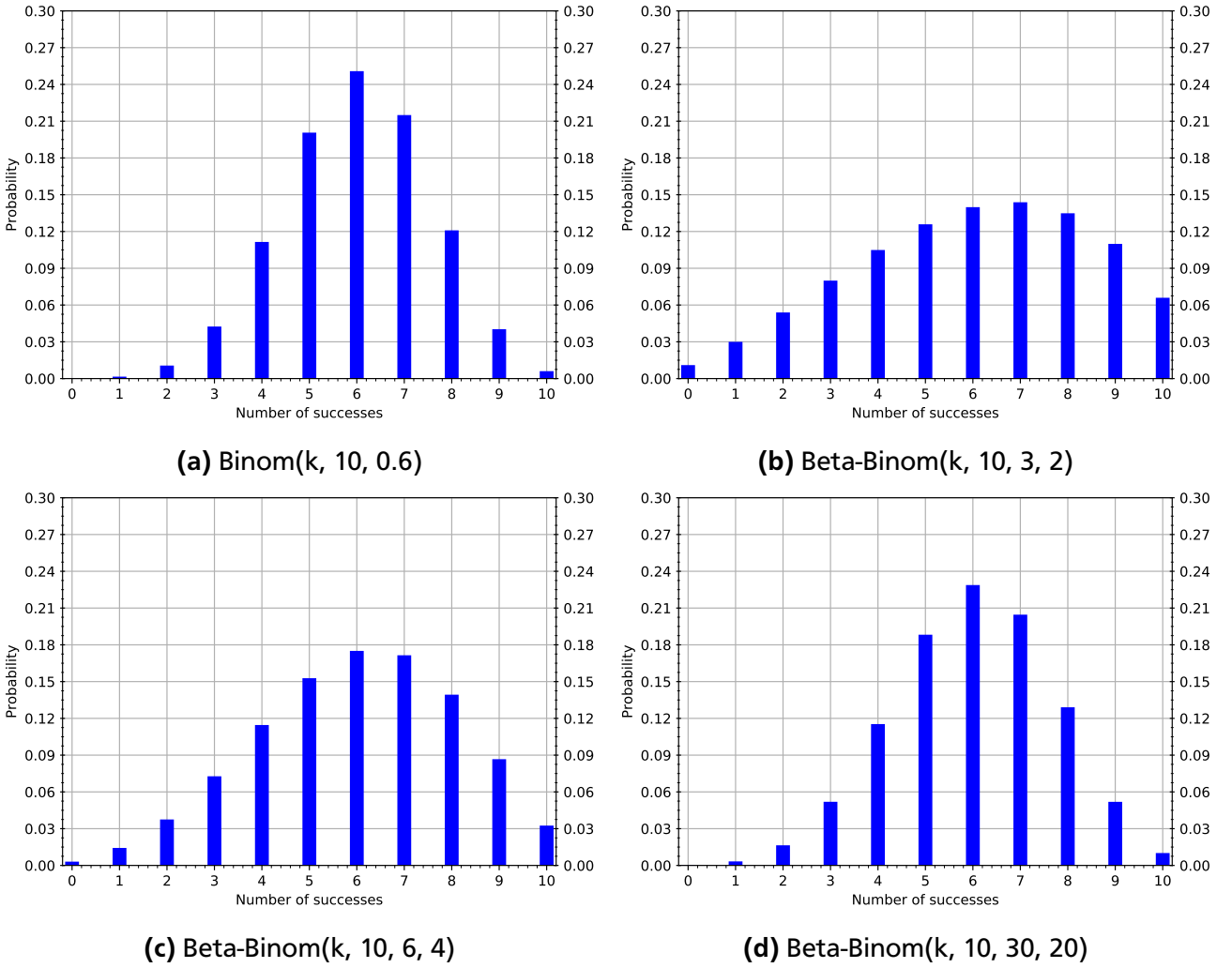
$$q(A) = \sum_{B \supseteq A} m(B), \forall A \subseteq \Omega. \tag{2.5}$$

Combining two mass functions is possible by using Dempster's rule of combination $\textcircled{\cap}$:

$$(m_1 \textcircled{\cap} m_2)(A) = \sum_{B \cap C = A} m_1(B) m_2(C), \forall A \subseteq \Omega. \tag{2.6}$$

An important disadvantage of Dempster's rule of combination for this thesis is that it requires m_1 and m_2 to be independent and to not share parts of evidence. That also means that classifiers which are trained

[8] Sometimes a and b is used instead of α and β.



(a) Binom(k, 10, 0.6)

(b) Beta-Binom(k, 10, 3, 2)

(c) Beta-Binom(k, 10, 6, 4)

(d) Beta-Binom(k, 10, 30, 20)

Figure 2.2: Here, examples of a binomial and different beta-binomial distributions are shown.

on overlapping datasets are not completely independent, even if they were trained using different algorithms. Therefore, Dempster's rule of combination does not seem well suited to combine such classifiers. Moreover, decision trees in random forests also share parts of evidence (because data points are used multiple times in multiple decision trees) and therby Dempster's rule of combination is probably not a good choice for combining decision trees in random forests.
For this reason, the cautious rule of combinations (cautious rule) was introduced (Denœux, 2006). The idea behind the cautious rule is based on the *Least Commitment Principle* (Smets, 1993). This principle states that, given two plausibility functions pl_1 and pl_2, pl_2 is less committed than pl_1 if

$$pl_1(A) < pl_2(A), \forall A \subseteq \Omega.$$

Following the intuition behind a plausibility function $pl(A)$ as "the maximum amount of potential specific support that could be given to A" (Smets, 1993)[9], the *Least Commitment Principle* states that, when considering two belief function, the least committed one is the most appropriate. Without going into any more detail about the derivation (you can refer to the quoted papers for that purpose), the idea behind the cautious rule is finding the mass function m_{12} as a combination of m_1 and m_2 (therefore being more

[9] For a better understanding: The plausibility functions contains the sum of all masses which share at least one element of Ω, so one possible state. Therefore, any mass not contained in $pl(A)$ does not share anything with A. So, the plausibility functions give as much support to A as reasonably possible.

informative than either original one). Applying the *Least Commitment Principle*, you choose the mass function as the solution which is the least committed one out of the set of possible mass functions.
The calculation can be done by calculating the weight functions of m_1 and m_2 using

$$w(A) = \prod_{B \supseteq A} q(B)^{-1^{|B|-|A|+1}}, \tag{2.7}$$

and combining those weight functions. Here, $\wedge$ denotes the minimum operator.

$$w_{12}(A) = w_1(A) \wedge w_2(A), \forall A \subset \Omega \tag{2.8}$$

Using this equation, the cautious rule of m_1 and m_2 is noted $m_1 \textcircled{\wedge} m_2$ and given by

$$m_{12} = m_1 \textcircled{\wedge} m_2 = \bigcirc\!\!\!\!\!\cap_{A \subset \Omega} A^{w_1(A) \wedge w_2(A)}. \tag{2.9}$$

2.7 Exploration and Exploitation

Another topic which at first glance does not seem very relevant for this paper is exploration and exploitation. Often explained by the example of the multi-armed bandit problem, the core of this area is making the best decision in an environment where you want to maximize some kind of score (reward) but you lack knowledge about the environment. The multi-armed bandit problem can be seen as a problem, where there are several slot machines and you want to choose the slot machine which maximizes your profit. However, you do not know the chances of the slot machines. If one machine gave more profit than another one after trying out both once, this does not guarantee that the second one is worse than the first one because the performance is influenced by randomness. Now, the problem is making the best decision when choosing which slot machine to use. Thereby, two factors need to be considered. Exploitation describes the fact that slot machines which performed well in previous tries are more likely to be profitable than those which did not. However, you do not want to exploit a machine which in fact is worse but only was more profitable because of randomness which is why you also want to exploit the other slot machines. Therefore, the *upper confidence bound* algorithm (UCB) was introduced which aims to optimize the trade-off between exploration and exploitation (Auer et al., 2002).

2.8 Decision Trees

One common machine learning technique is decision tree learning. Thereby, a tree structure is used to predict the class of a new instance by following the path according to this instance. Using the training data, decision trees are built top down by choosing a feature on which the current node (starting at the root) is split. At the beginning, all instances can be seen as contained in the root node. When splitting a node on a feature, the node gains two or more child nodes, one for every possibility of the split. The instances of the node which has been split are then divided accordingly and assigned to the associated child nodes. This procedure can be repeated multiple times for every child node until either the resulting child only contains instances of one class, in which case the prediction for this node can not be improved further, there are no more feature to be split on, or another stop criterion is reached. After that, predicting new instances is possible by following the decision path of the tree considering the features of the instance to be classified. When reaching a node, a decision about the prediction has to be made. The most common method is using the probability taken from the instances in that node. The class with the highest probability is then predicted.
There is a certain terminology for trees and decision trees in particular, which will be important for

understanding later parts of this thesis. Every node of a tree, which has no successor (no child nodes) is called a leaf or a leaf node. The depth of the tree describes the maximum number of nodes in any decision path starting from the root node going to a leaf. Thereby, the root node has a depth of zero, children of the root node have a depth of one, their children a depth of two and so on. In order to assign a depth to every node in the tree, this procedure can be repeated until all leaf nodes are reached. For example, Figure 2.3 shows a simple decision tree with a maximum depth of three. The decision tree in this example can be used to determine whether an animal is a bird, a cat, a dog, a dolphin, an elephant, a fish or a spider. When classifying a new instance, the decision tree is traversed from the root node (orange) to a leaf node (blue). At each node, starting with the root node, the test of the respective node is performed and then the corresponding edge is followed to the next node. When a leaf node is reached, a prediction is made based on the information in that leaf. Keep in mind that usually the prediction in the leafs is based on the instances in that leaf. This is not shown in the figure here.
Pruning of a decision tree describes the process of simplifying the tree in order to avoid overfitting on the training data. Among others, two common ways of pruning decision trees is restricting the growth of the tree in the tree building process by specifying a minimum numbers of instances which have to be contained in a node in order to construct a split resulting in that node or by specifying a maximum depth of the tree. However, deciding how to set those and other decision tree parameters, can be a difficult and time consuming task when looking for the best[10] decision tree.

In this thesis, the focus is on binary classification and therewith on binary decision trees. An example

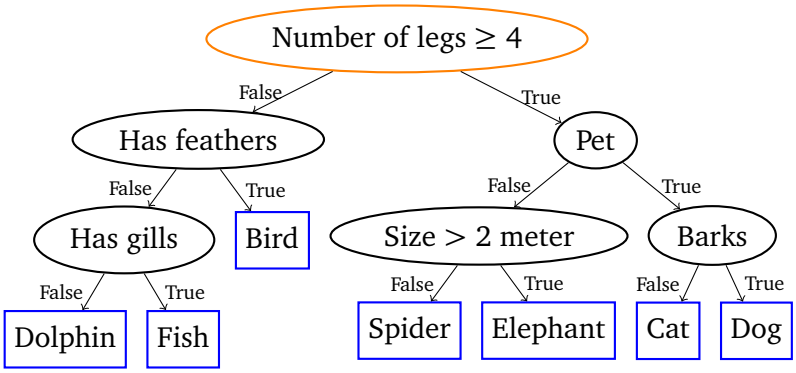


Figure 2.3: Simple example of a decision tree.

of a binary decision tree is shown in Figure 2.4a. Instead of classifying between different animals, this decision tree only decides if an instance is a mammal or not. Because we know that every pet in our example (not in reality) is a mammal, the *Barks* node of the decision tree in Figure 2.3 can be omitted. Note, that, since there are only two possible labels, different leafs will predict the same class.[11] Here, the classes of binary classification will be seen as positive and negative (for simplicity) and the colors green and red, respectively, will be used to represent these classes. Besides Figure 2.4a, Figure 2.4b shows how a pruned decision tree[12] might look like. In this example, the decision tree was built on a training set containing one instance of every animal of the previous example[13]. One pruning strategy which could have resulted in that tree (instead of the unpruned tree of Figure 2.4a) is restricting the maximum depth to 1. In this example, the instances of the training set (here, there are only seven train instances, which is far less than should be used in an actual machine learning model) are split among all leafs of the decision tree. The instances in the respective leaf then determine the prediction made

[10] Again, it should be noted that the "best" classifier or decision tree can vary even within the same task depending on the evaluation technique.

[11] That is not unusual but it is stated here explicitly so that the reader does not get the wrong idea from the first example, where this is not the case.

[12] Alternatively, this tree can be seen as the decision tree in Figure 2.4a but still in creation after constructing the first split.

[13] Which is one bird, one cat, one dog, one dolphin, one elephant, one fish and one spider, so a total of seven training instances.

for that leaf. In this example, a probability of 33% in favor of the positive class would be predicted for the first (left) leaf, a probability of 75% in favor of the positive class for the second (right) leaf. Because the class distributions in the leafs are not distinct, this decision tree does not have an accuracy of 100% on the training set but it might generalize better to unseen data. For example, hamsters are classifiyed correctly by the smaller tree but not by the first one. This also shows how important it is to have a large amount of representative data in order to train a good model.

Furthermore, it should be said that there are a couple of important decision tree algorithms which can be

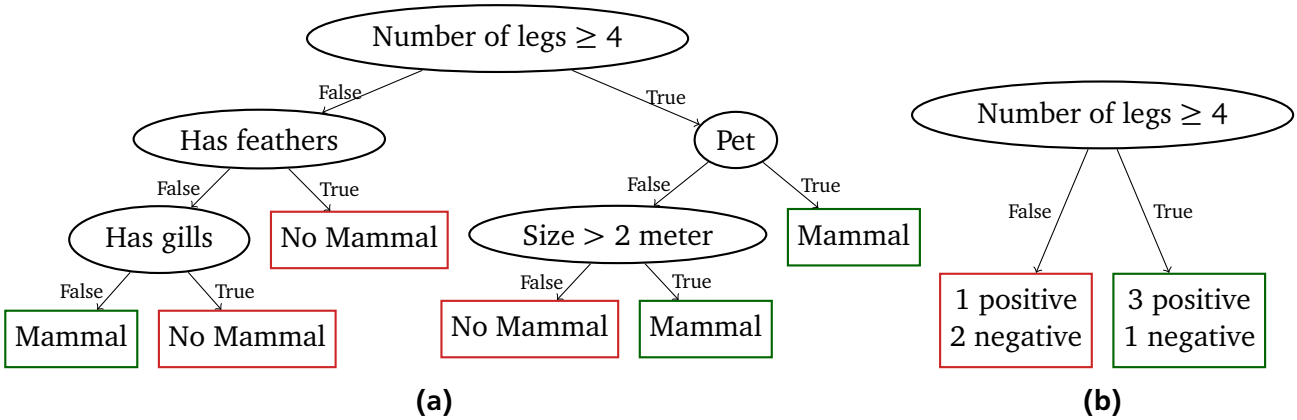


Figure 2.4: Figure 2.4a shows an example of an unpruned binary decision tree. How a pruned decision tree for the same data could look like is shown in Figure 2.4b.

used for machine learning classification. An introduction to what are probably the most popular decision tree algorithms, ID3, C4.5, and CART is given in Gupta et al. (2017). In addition to listing advantages and disadvantages of these algorithms, this paper also contains information about the attribute selection measures entropy, information gain, gain ratio, and gini index which, when building the decision tree, can be used to determine the best split of the data in a decision tree node. The decision tree algorithms and attribute selection measures are not well explained within this thesis because this thesis aims to explore how a prediction of a new instance in a decision tree can be improved in order to achieve a better overall prediction in random forests. Therefore, for this thesis, we assume that the decision trees and the decision tree structure are fixed and we do not try to improve the decision tree learning but the decision tree classification (prediction) process. For that, comprehensive knowledge about the decision tree building process is not necessary.

2.9 Ensemble Learning

In machine learning, particularly in classification, we want to achieve the best possible accuracy. Therefore, many methods exist which can help to improve the performance (i. e. accuracy) of a model when using machine learning algorithms. One such technique is ensemble learning. "Ensemble learning refers to learning a weighted combination of base models of the form

$$f(y|x,\pi) = \sum_{m \in \mathcal{M}} w_m f_m(y|x)$$

where the w_m are tunable parameters." (Murphy, 2012) Here, $\mathcal{M}$ ist the set of classifiers in the ensemble, $f_m(y|x)$ is the prediction of the instance x for class y made by the classifier m and π is a parameter vector. Less formally, ensemble learning describes the process of using the outputs of multiple machine learning models (for example classifiers) in order to aggregate those to one final prediction. The idea is, that using multiple models which all by themselves have different errors in different regions of the data, makes it so that those error regions of one specific model will be *overshadowed* by the aggregated

prediction of the ensemble (the set of models used for ensemble learning is called ensemble).
For example, a very trivial but nevertheless quite effective way of applying ensemble learning is by averaging the predictions of the ensemble. When looking at the definition given by Murphy above, the average would correspond to the case of $w_m = \frac{1}{|\mathcal{M}|} \forall m$. Another possibility is using the best prediction of the ensemble. In that case, the weight for the best prediction w_m^* is $w_m^* = 1$ and the weight for every other prediction is $w_m = 0, \forall m \neq m^*$.
When applying ensemble learning, some things should be considered. First of all, ensemble learning always comes at the cost of runtime and storage requirement. If you use n models, you need n times more runtime[14] and storage than if you are simply using one model. Therefore, you want the ensemble to be worth the effort. But if we now image using ensemble learning on top of our machine learning model to achieve better results, we might have the problem that our models are very similar or even identical. Especially in the latter case we have an extra cost but no improvement in performance. That should make clear that we would like the models to be different in some ways because otherwise ensemble learning is superfluous. How you achieve that can vary for different machine learning algorithms. Some might even contain a significant amount of randomness in which case it could be enough to use different random seeds to get models with varying performance in different regions of the data. However, there also exist different methods which are not as model specific. If you have enough data, you might consider splitting the data in n parts, and training n different models on one part of the data each, thereby getting n different models. The idea is that, if there is enough data available, the different models will each capture a large part of the data and make errors in different regions. Therefore, ensemble learning will improve the overall performance. However, you want to train your model on a large amount of data. Having enough data can be a problem in any case but this problem is amplified when only using $\frac{1}{n}$ of the training data.
Another, more data efficient way of training your models is bagging (Breiman, 1996). Bagging[15] describes the procedure of using different parts of the same data set to train a set of different models on that datasets. Thereby, every model draws data from the dataset "at random, but with replacement" (Breiman, 1996). Therefore, any datapoint might be in the training set for any classifier either once, not at all, or even multiple times. Breiman showed that bagging is a useful method which can help to increase the accuracy when using machine learning.

2.10 Random Forests

Random forests are an ensemble learning technique based on decision trees. Basically, random forest use a number of decision trees in order to derive an overall prediction based on the single decision tree predictions. As stated before in 2.9, the models within an ensemble should be different in some ways in order for the ensemble to add value to the aggregated prediction. Random forests achieve that in two ways. Firstly, bagging is applied to train different decision trees on different parts of the data. Secondly, the trees are built in a more random way than usual by selection some features at random. This inserts more randomness in the single decision tree building process which makes the random forest ensemble more effective. As a result, random forests are a powerful machine learning method which can achieve good performances in machine learning and particularly improve upon single decision trees (Breiman, 2001).

[14] One advantage of ensemble learning is that it is possible to to increase the runtime as much because you can parallelize the classifier in the ensemble.
[15] Short for "bootstrap aggregating", for more details see Breiman (1996).

3 Related work

The problem in Ceolin et al. (2011) is that there is a high uncertainty in a decision when the population the decision is based on is too small. When deriving a probability from a small number of samples, you should not assume that the resulting probability is representative of the actual probability, "Flipping a coin twice, obtaining a heads and a tails, does not guarantee that the coin is fair, yet." (Ceolin et al., 2011). What is done in this paper is deriving the probability from the beta distribution where the parameter α and β are taken from the samples but additionally increased by one. This way, a higher sample count is needed in order to approximate the directly derived probability. This can also be seen as using the Beta-Binomial distribution and comparing it to the Binomial distribution.
The *Laplace estimation* or *Laplace correction* was suggested by Provost and Domingos to be the best way of getting probability estimates of decision trees. Thereby, instead of using the calculated probability $\frac{p}{N}$, where p is the number of instances of the respective class and N is the total number of instances of that leaf, the *Laplace correction* provides a smoothed estimation by predicting a score of $\frac{p+1}{N+C}$ (C being the number of total classes). It was also stated that larger trees give better result for probability estimation and that the *Laplace correction* works well on those trees. Furthermore, using bagging with decision trees for probability estimation never worsened but potentially improved the results of probability estimation.
Another paper goes into detail as to why random forests perform well (Wyner et al., 2017). Here, random forest are compared to AdaBoost, a very successful machine learning algorithm where an improvement is achieved over multiple iterations. Thereby, AdaBoost adapts the model to improve accuracy on those instances which were classified incorrectly by the previous iterations (Freund et al., 1999). Moreover, based on empirical observations, Freund et al. stated that AdaBoost does not overfit even for many iterations. Wyner et al. argue that random forest behave similarly. Therefore, they introduced the informal notion of *spiked-smooth*. Thereby, *spiked* describes the behavior of classifiers adapting to small regions like noise points and *smooth* refers to this behavior being achieved by averaging. In other words, a *spiked-smooth* decision surface can be a result of averaging multiple classifiers which perfectly fit their respective data. Thereby, *spiked-smooth* classifiers fit noise of the data but only locally in very small regions. Therefore, overfitting should not be a large problem because the extreme localization of noise points makes it so similar data points are not classified as the noise points but as the region around them. All in all, AdaBoost as well as random forest are, according to Wyner et al., not (or at least only slightly) affected by overfitting.
Considering probability estimation, Možina et al. argue that rule learning algorithms (therefore including decision trees) are to optimistic when building models. If you were to find out the probability of a region (for a single rule), you could determine those by calculating $\frac{s}{n}$ where s is the number of positive examples and n is the number of all examples. However, it is argued that rule learning tends to be overly optimistic on that ratio. Since rule learning aims to find regions with a distinct (0% or 100%) probability, it might happen that rules are chosen in a way that pure regions are discovered even though those regions are not actually representing the computable probability $\frac{s}{n}$. However, those rules can be a result of random patterns in the data and not of the actual underlying probability. For this reason, Možina et al. introduced an algorithm to determine $\widetilde{s}$ which describes the expected value of s. Using $\widetilde{s}$, it is possible to calculate a probability estimation of a region which is not as optimistic. They conclude with mentioning that such corrections might be beneficial for probability estimation on some machine learning algorithms which behave similar as rule learning such as decision tree learning.

4 Decision Tree Prediction Methods

This chapter covers prediction methods which can be used in place of what is usually the probability. As is the goal of this thesis, the proposed prediction methods aim to take the number of instances used for the prediction into account. Therefore, for this chapter we define p as the number of positive instances in the leaf and n as the number of negative instances in the leaf. So, to make a prediction of the class of the instance i, this instance i will be assigned to a leaf in the decision tree. Based on the number of positive instances in that leaf (p) and the number of negative instances in that leaf (n), the prediction of i is made.
Please note, that the formulas in the different approaches calculate the prediction value in favor of the positive class. The prediction value for the negative class can be calculated by simply swapping p and n in the respective formula.
First, the probability approach is explained in more detail (4.1), then a prediction method which strongly takes the number of instances into account is presented (4.2). The following prediction methods are based on different ideas, namely prior information (4.3), UCB (4.4), belief functions (4.5) and probability distributions (4.6).

4.1 Probability

Using the probability is what is usually done when making predictions from decision trees. Since the goal of this thesis is improving upon this probability approach, it is worth shortly explaining how probability is applied in decision trees in order to make appropriate comparisons afterwards. The probability is calculated as follows:

$$\text{Probability}(p,n) = \frac{p}{p+n} \tag{4.1}$$

As stated previously, you can see that the probability stays constant if the number of positive and negative instances increases proportionally. The behavior of the probability function can also be seen more precisely in Figure 4.1. Here, as well as in the figures of the following sections, there are two types of graphs. The first type of graph only shows one function, which is plotted based on the number of instances p and n which are increased in a constant relation $\frac{p}{n}$ whereby p and n are non-negative integers[1]. So for those type of graphs, we start with the smallest values p and n which fulfill the relation and plot the corresponding value. Then we plot $2 \cdot p$ and $2 \cdot n$ with the corresponding value and we continue like that until $x \cdot (p+n) > 100$ because we only plot the graph until 100 instances. We stop at 100 because with an increasing amount of instances, the uncertainty (at least the epistemic part) will decrease. Therefore, for the motivation of this thesis, it is more important to achieve good results for lower number of instances. The other type of graph uses the relation of p and n itself, so basically the percentage of positive instances. Using this class ratio (basically the underlying probability), instances are generated. This results in more or less varying graphs because for different graphs different values are generated. For example, given a class ratio of $\frac{3}{4}$ (0.75%), the first ten instances might all be positive, therefore not really representing the underlying probability. On the other hand, this behavior is the same in actual datasets. Due to noise or just randomness in general, the instance representation does, most of the time, not approximate the actual probability of this region well. To get a better understanding of this, the graphs of this type are built on 1000 different instance sets based on the class ratio. For every graph shown in this chapter, an underlying probability of 75% is used. Every graph is shown (light blue).

[1] While some methods support $p = n = 0$, other methods like the probability are not defined on $p = n = 0$. Since we are never dealing with empty leafs, $p + n > 0$ always holds and that case is not a problem. Whenever it is undefined, no value for that case is calculated and therefore no value for that case is shown in the graph.

The red line in those graphs shows the average of all graphs, which should therefore be similar to the corresponding graph of the previous type. The green lines show the 0.1 and 0.9 quantile and the purple lines show the 0.25 and the 0.75 quantiles. With the help of those colored graphs, it is easier to make statements about the behavior of the respective methods.

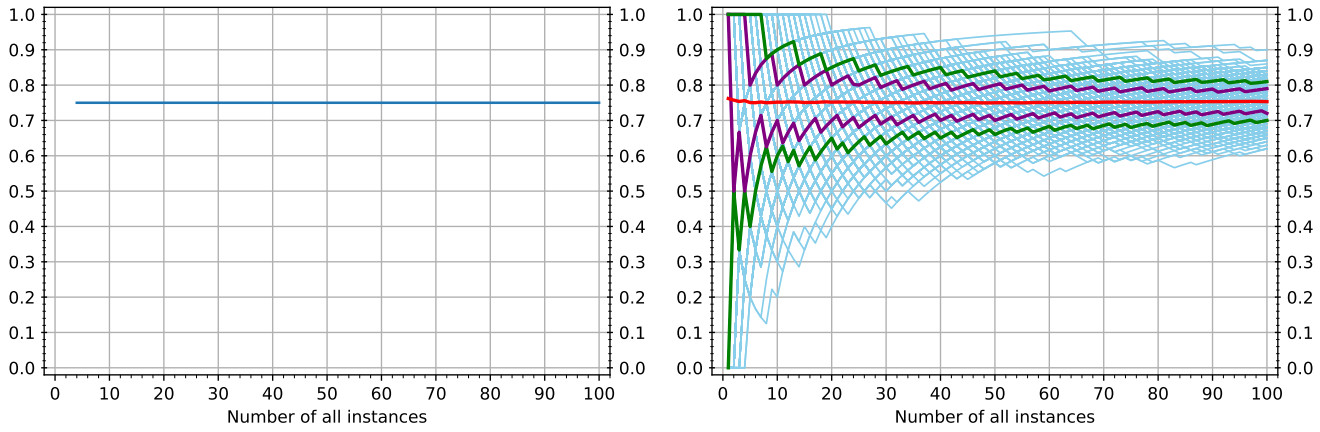


Figure 4.1: Probability. The very simple behavior of this graph makes sense, since the graph assumes a constant probability. The right graph uses randomly generated values according to the probability of 0.75. It can be seen, that especially for small number of instances, the confidence using probability can be significantly to high or to low. For high instances numbers, the results seem reasonable.

4.2 Linear Prediction

One way to directly take the number of instances into account is to use the difference between the number of instances of the positive and of the negative class.

$$\text{Linear}(p,n) = p - n \tag{4.2}$$

Since the difference between the positive and the negative instances determines the strength of the resulting value, more instances with the same class ratio are valued higher than less instances. While this fulfills the motivation of this thesis, one might argue that it is not a good property of this approach that a leaf $l_1 = (p_1, n_1) = (5, 0)$ results in the same value as another leaf $l_2 = (p_2, n_2) = (505, 500)$. You could say that l_1 is in favor of the positive class while l_2 is pretty much even. This is a result of the inability of this approach to distinguish between the uncertainty of the prediction based on the number of instances (epistemic uncertainty) and the uncertainty which is a result of the underlying data distribution (aleatoric uncertainty). In the previous example, it seems likely that a prediction resulting in l_2 has about equal chances of being positive or negative, so you can be pretty sure that even the actual probability of instances in that leaf is about 50%[2]. For l_1 on the other hand, if you had to make a prediction, you would predict the positive class because we have only seen a couple of positive but not one negative instance. Still, the small number of instances on which this prediction is based also means that the certainty of this prediction is not very high.
Another important notion is that, compared to most of the other prediction methods, linear prediction does not return a value in [0,1] but any natural number depending of the number of instances. This might be worth keeping in mind when looking at aggregation strategies.

[2] We are looking at a leaf with 1005 instances, so it is safe to assume that the difference between the actual probability (unknown) and the calculated probability ($\frac{505}{1005} \approx 50\%$) is not particularly large.

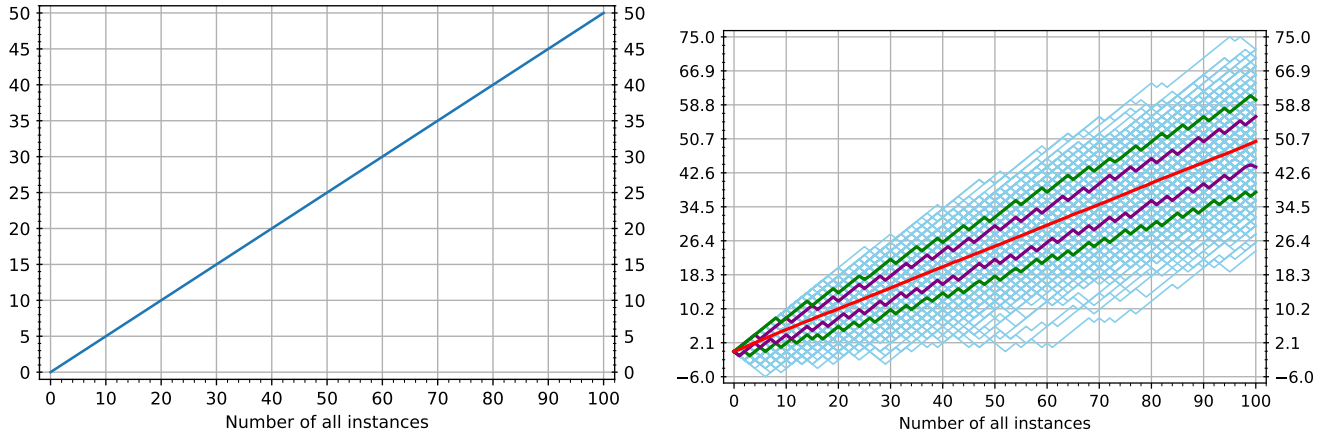


Figure 4.2: Linear. As per definition, the linear-prediction graph increases linearly with the number of instances (more precisely, the difference $p - n$). An important property of this approach is, that a higher number of instances results in a higher prediction value which is easy to see on both plots. However, the variance actually increases with more instances.

4.3 Prior Based Prediction

As described before, the beta binomial distribution takes prior knowledge into account. Inspired by this procedure, you can think of a prediction method utilizing prior knowledge. So instead of deriving a prediction from (p, n) only, you might be able to reach more accurate predictions using $\frac{p+p_{\text{prior}}}{p+p_{\text{prior}}+n+n_{\text{prior}}}$. Obviously, the question remains how p_{prior} and n_{prior} should be set. To allow for an intuitive understanding of the method names, we will call those methods *Prior* prediction methods. In this thesis, two suggestions for choosing the prior values are made. The $\text{Prior}_{1,1}$ prediction[3] uses $p_{\text{prior}} = 1$ and $n_{\text{prior}} = 1$, resulting in

$$\text{Prior}_{1,1}(p, n) = \frac{p+1}{p+1+n+1} = \frac{p+1}{p+n+2} \tag{4.3}$$

In comparison to probability, this approach only slightly changes the resulting value for large p and n. For small values, the difference is more significant. As a result, this formula achieves the motivation of this thesis in a rather simple way. The more instances are available for the prediction, the closer the predicted value comes to the probability value of $\text{Probability}(p, n) = \frac{p}{p+n}$. For example, for one positive and no negative instance $\text{Probability}(1, 0) = 1$ but $\text{Prior}_{1,1}(1, 0) = \frac{2}{3}$ whereas for 100 positive and zero negative instances, $\text{Probability}(100, 0) = 1$ and $\text{Prior}_{1,1}(100, 0) = \frac{101}{102} \approx 1$. You can see this behavior and thereby the difference to the probability prediction in Figure 4.3

The second prior based formula suggestion uses the instances of the prior node of the decision tree path as prior information.

$$\text{Prior}_{\text{prev-node}}(p, n) = \frac{p + p_{\text{prior}}}{p + p_{\text{prior}} + n + n_{\text{prior}}} \tag{4.4}$$

where p_{prior} and n_{prior} are the number of positive and negative instances, respectively, which are taken from the predecessor of the leaf node. Note that this strategy is similar to using the previous node with the probability function, only with the difference that the instances which also reach the leaf are weighted twice ($p <= p_{\text{prior}}$ and $n <= n_{\text{prior}}$). This method can be seen as a compromise between taking the probability of the leaf node and taking the probability of the previous node (reducing the depth of

[3] $\text{Prior}_{1,1}$ can also be reffered to as the *Laplace correction* (Provost and Domingos, 2000) and is not a new method. However, I will stick to the name $\text{Prior}_{1,1}$ because of the connection to the other *Prior* method in this chapter.

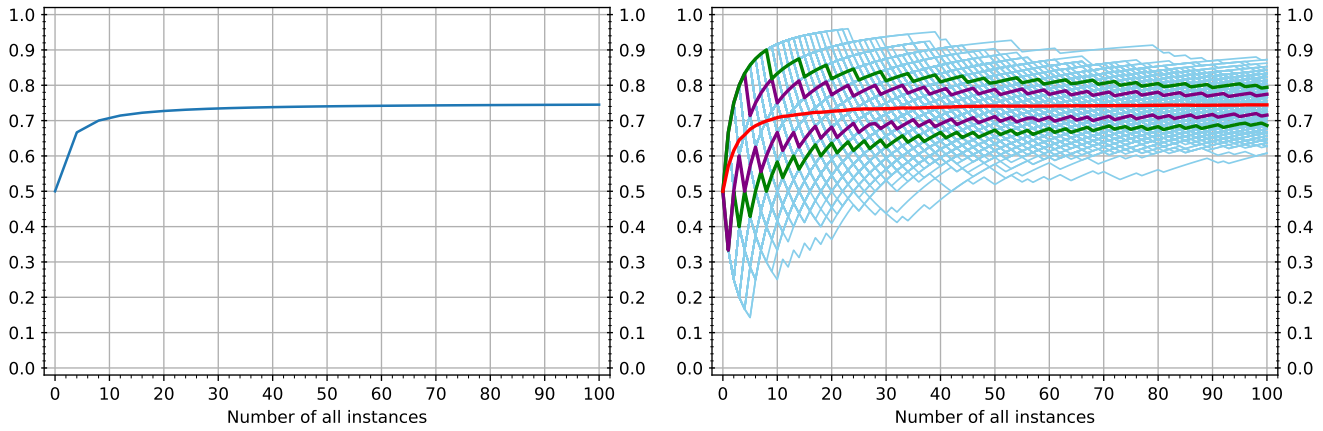


Figure 4.3: $\text{Prior}_{1,1}$. Here, the prediction starts at 0.5 but converges fast to the underlying probability of 75%. While this is very similar to the probability prediction, for small number of instances the spread of the graph is significantly smaller.

the tree by one for the respective prediction). How the $\text{Prior}_{\text{prev-node}}$ prediction method might look like can be seen in Figure 4.4.

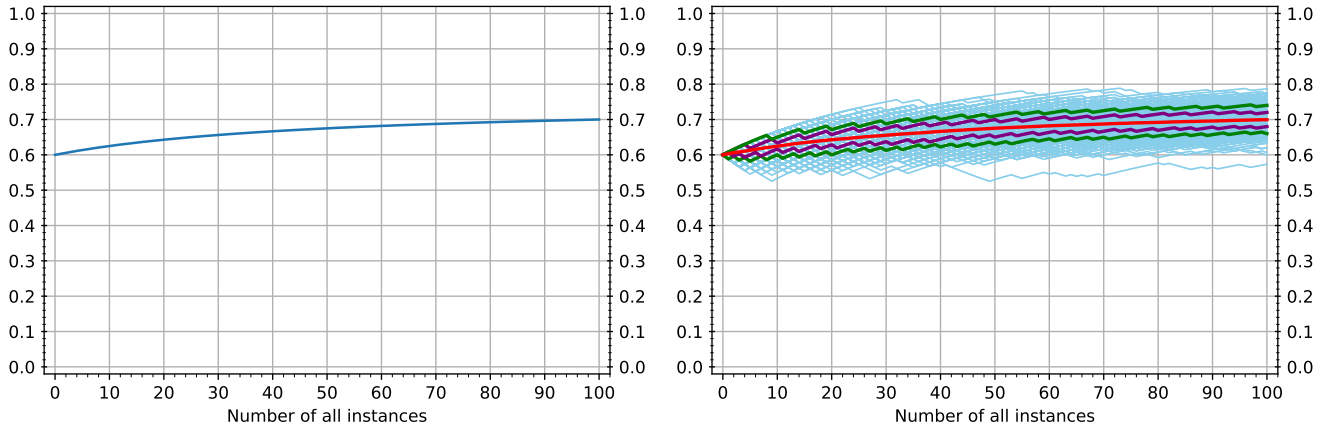


Figure 4.4: $\text{Prior}_{\text{prev-node}}$. Here, $p_{\text{prior}} = 30$ and $n_{\text{prior}} = 20$. Keep in mind that the previous node ($p_{\text{prior}} = 30$ and $n_{\text{prior}} = 20$) in reality can not be smaller than the leaf node. This is not taken into account here, because $p_{\text{prior}} = 30$ and $n_{\text{prior}} = 20$ are constant. Moreover, previous leafs can be very different so this graphs should be treated with caution.

4.4 UCB Inspired Prediction

The idea behind UCB was presented before. Here, this adaptation of the UCB formula is following the idea of exploration and exploitation in order to get a method which lets us make predictions based on decision tree leafs. In comparison to the multi-armed bandit problem though, we can not increase our knowledge of the instance distribution in the leafs, so the terms of exploration and exploitation have to be interpreted slightly differently.
Exploitation describes the probability of one class compared to the other class. So if we want to be secure about the prediction, the probability should be as close to 1 or 0 for the positive or negative class as possible. Looking at multi-armed bandits, a high exploitation value indicates that this decision had

good results up to this point so it might be a good idea to pursue that decision. For the decision tree prediction, the reasoning can be understood similarly. Training instances which reached a leaf showed that is it more or less likely for instances of that sort to be classified as the specific class when they follow the same decision path. On the other hand, exploration can be seen as an uncertainty about the decision. The idea is that leafs with a high number of instances can be seen as if they had frequently been explored in the past. Therefore, in this decision tree based approach, we say that the exploration is small when a prediction is based on many instances and vice versa. The resulting formula looks as follows:

$$\text{UCB-DT}(p,n) = \text{Probability}_{\oplus}(p,n) + \alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} \tag{4.5}$$

where n_{all} is the overall number of instances used in the training process and α can be set at will. Now, we could choose an α value to determine the quality of the leaf. But instead of doing so, we suggest calculating the α value for a specified scenario and using that value as the degree of confidence of this approach. The idea is that for $p > n$ we have two exploitation values, $\text{Probability}_{\oplus}(p,n) \in [0.5,1]$ and $\text{Probability}_{\ominus}(p,n) = 1 - \text{Probability}_{\oplus}(p,n) \in [0,0.5]$. We now want to calculate the α value for which $\text{Probability}_{\oplus}(p,n) - \alpha \cdot \sqrt{\frac{\ln(n_{\text{all}})}{p+n}} = \text{Probability}_{\ominus}(p,n) + \alpha \cdot \sqrt{\frac{\ln(n_{\text{all}})}{p+n}}$. If $n > p$, the same formula applies, but with switched classes of positive and negative. Note that for $p = n$ the value of α is zero. With this information you can calculate α:

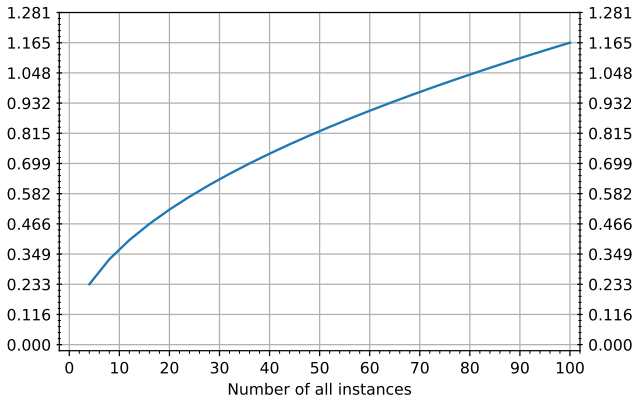
$$\begin{aligned}
\text{Probability}_{\oplus}(p,n) - \alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} &= \text{Probability}_{\ominus}(p,n) + \alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} \\
\text{Probability}_{\oplus}(p,n) &= \text{Probability}_{\ominus}(p,n) + 2\alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} \\
\text{Probability}_{\oplus}(p,n) - \text{Probability}_{\ominus}(p,n) &= 2\alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} \\
\frac{\text{Probability}_{\oplus}(p,n) - \text{Probability}_{\ominus}(p,n)}{2} &= \alpha\sqrt{\frac{\ln(n_{\text{all}})}{p+n}} \\
\frac{\text{Probability}_{\oplus}(p,n) - \text{Probability}_{\ominus}(p,n)}{2\sqrt{\frac{\ln(n_{\text{all}})}{p+n}}} &= \alpha
\end{aligned}$$

$$\alpha = \frac{\text{Probability}_{\oplus}(p,n) - \text{Probability}_{\ominus}(p,n)}{2\sqrt{\frac{\ln(n_{\text{all}})}{p+n}}} \tag{4.6}$$

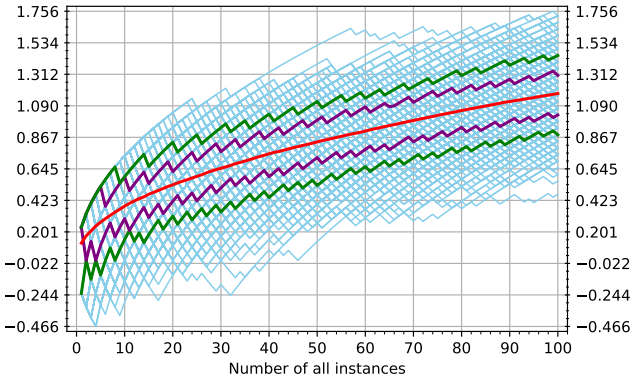
The larger α, the higher the confidence in the prediction. Since α only represents the confidence in the prediction, it still needs to be combined with the actual prediction. The prediction then is

$$\text{UCB-DT}(p,n) = \begin{cases} \alpha, & \text{if } p > n \\ -\alpha, & \text{if } p < n \\ 0, & \text{if } p = n \end{cases} \tag{4.7}$$

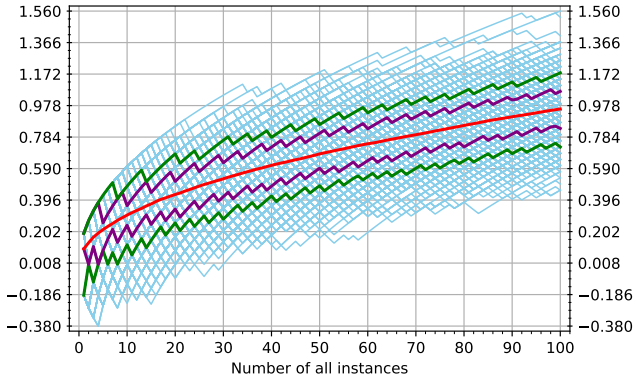
What the formula looks like in use can be seen in Figure 4.5.



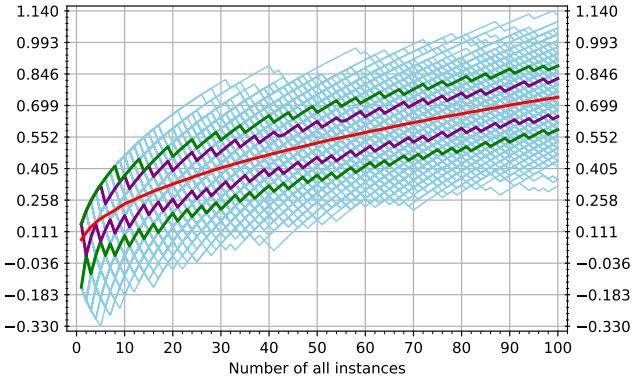
(a) $n_{\text{all}} = 100$



(b) $n_{\text{all}} = 100$



(c) $n_{\text{all}} = 1000$



(d) $n_{\text{all}} = 100000$

Figure 4.5: UCB. Here, graphs of different values of n_{all} are shown. The prediction value increases for more instances, but slower than linearly. It can also be seen, that the third and fourth graph look like the second one, but decreased by a constant factor. This might indicate, that the value of n_{all} does not matter when comparing prediction values, as long as it stays equal for all predictions.

4.5 Plausibility

A paper by Hüllermeier is based around the distinction of aleatoric and epistemic uncertainty. By combining the set based approach around belief functions and probabilistic arguing, another prediction method can be created.

We want to find out the plausibility π for a class or model. First of all, the plausibility describing how likely it is that any classifier $h \in \mathscr{H}$ fits the data can be described as

$$\pi_{\mathscr{H}}(h) = \frac{L(h)}{\sup_{h' \in \mathscr{H}} L(h')} = \frac{L(h)}{L(h^{\text{ml}})}, \tag{4.8}$$

where $h^{\text{ml}} \in \mathscr{H}$ refers to the maximum likelihood estimation calculated on the data of the problem. For any classifier h, we can determine the degree of support of an instance x_0 for any class in binary classification (for example the positive class $\oplus$) by

$$\pi(\oplus | h, x_0) = max(2h(x_0) - 1, 0) \tag{4.9}$$

We can see that we only get support if $h(x_0) > 0.5$ which makes sense because if that is not true, the respective other class is more likely. Now, we can get the plausibility of the positive class considering an instance x_0 using the formula

$$\pi(\oplus, x_0) = \sup_{h \in \mathscr{H}} \min(\pi_{\mathscr{H}}(h), \pi(\oplus | h, x_0)) \tag{4.10}$$

The respective formula for the negative class ($\ominus$) can be defined analogously.
Following this idea, Hüllermeier introduces a formula which can be used for machine learning based on instances of two classes. For decision trees, we can again see p as the number of positive instances of a leaf and n as the number of negative instances of that leaf. Hereby, we get the formulas for π

$$\begin{aligned}\pi(\oplus) &= \sup_{\alpha\in[0,1]} \min\left(\frac{\alpha^p(1-\alpha)^n}{(\frac{p}{p+n})^p(\frac{n}{p+n})^n}, \max(2\alpha-1,0)\right)\\ \pi(\ominus) &= \sup_{\alpha\in[0,1]} \min\left(\frac{\alpha^p(1-\alpha)^n}{(\frac{p}{p+n})^p(\frac{n}{p+n})^n}, \max(1-2\alpha,0)\right)\end{aligned} \tag{4.11}$$

Coming back to the concept of aleatoric and epistemic uncertainty, we can now derive those values.

$$u_e = \min(\pi(\oplus), \pi(\ominus)) \tag{4.12}$$

$$u_a = \min(1-\pi(\oplus), 1-\pi(\ominus)) \quad = 1-\max(\pi(\oplus), \pi(\ominus)) \tag{4.13}$$

That makes it possible to derive the preferences of the positive and the negative class.

$$p_\oplus = \begin{cases} 1-(u_a+u_e), & \text{if } \pi(\oplus) > \pi(\ominus) \\ \frac{1-(u_a+u_e)}{2}, & \text{if} \pi(\oplus) = \pi(\ominus) \\ 0, & \text{else } (\pi(\oplus) < \pi(\ominus)) \end{cases} \tag{4.14}$$

We can now also derive the preference for the negative class

$$p_\ominus = 1-(p_\oplus + u_a + u_e) \tag{4.15}$$

and see that

$$p_\oplus + p_\ominus + u_a + u_e = 1 \tag{4.16}$$

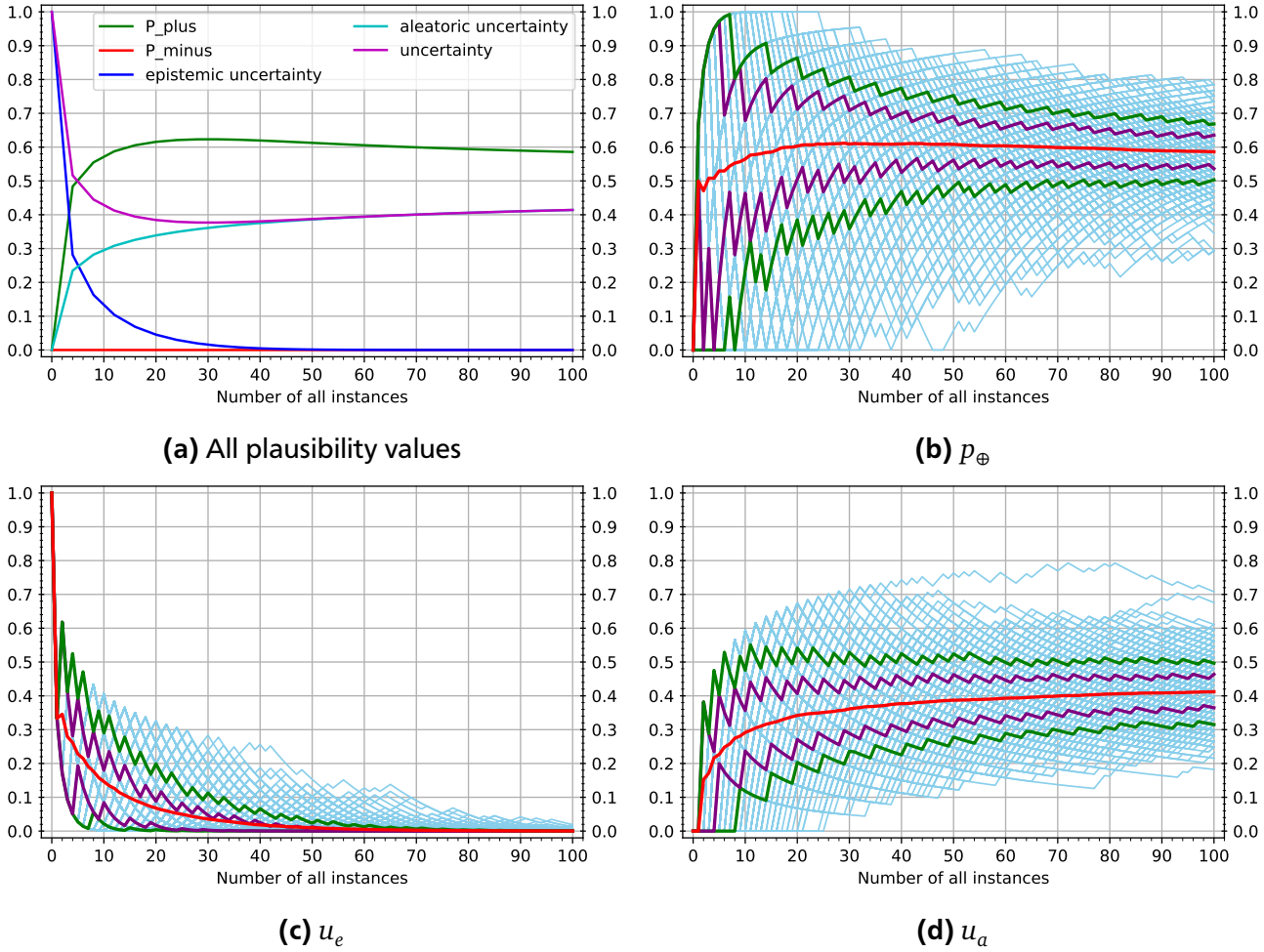
Thereby we have one measure for the positive and the negative class each and we can also determine the aleatoric and the epistemic part of this leaf (p, n). While $p_\oplus$ is not considered a plausibility itself, we will call this approach the *Plausibility* prediction because it uses the concept of plausibility. Since there is no other method using plausibilities, there is no danger of confusion and this name can be chosen. This leads us to our formula for the prediction

$$\text{Plausibility}(p,n) = p_\oplus \tag{4.17}$$

where $p_\oplus$ is calculated as explained above. How this whole *Plausibility* approach (Hüllermeier, unpublished) can behave is visualized in Figure 4.6.

4.6 Confidence Bounds

Another way for assessing the confidence in a prediction based on instance counts is taking a look at probability distributions. The idea is to build a probability distribution on the leaf, which is used to predict the instance. Therfore, the number of positive and negative instances in that leaf are available to be used when building the distribution. Now, you can build a probability distribution for the other class by simply swapping the values of p and n. Doing so will create two probability distributions, one for every class, which can now be used to find out how likely it is that one class should be preferred (and by how much).
Following this idea, one way to measure the similarity between the distribution of both classes is by



(a) All plausibility values **(b)** $p_{\oplus}$

(c) u_e **(d)** u_a

Figure 4.6: Plausibility prediction. The upper left graph (4.6a) shows how the components of the probability approach behave (uncertainty is the sum of aleatoric and epistemic uncertainty). The upper right graph (4.6b) shows how $p_{\oplus}$ settles down around 0.6, the bottom left (4.6c) shows how the epistemic uncertainty decreases and the bottom right graph (4.6d) shows how the aleatoric uncertainty first increases and than settles around 0.4.

taking the shared are of both distributions. For example, in discrete distributions this can be achieved by taking the minimum value of every possible number of successes. The resulting area must be within the interval $[0, 1]$ because the sum of the area of one single distribution must be 1 and the area can not increase by taking the minimum. Therefore, a small area close to 0 indicates that almost 100% of all instances can be strictly assigned to one class, while an area of 1 (both distributions are the same) indicates that both classes are equally likely.
An alternative to using the area is using the intersection of the distributions in the middle[4]. The intuition here is that the higher the intersection point is, the more similar are both distributions. If the intersection is at the bottom close to 0, then the similarity of both distributions is small so the prediction of one class can be done with more confidence. Furthermore, the interpretation of the height is more difficult than for the area. What you can say is, that, starting from 0, a higher height (intersection point) indicates a weaker prediction. But unlike the area, the height is not necessarily within $[0, 1]$ because discrete probability distributions usually have small values for a high number of tries n. So, to increase the comparability of different heights, you could normalize the heights by dividing it with the maxmimum of the probability distributions. This way, all heights are in $[0, 1]$ and a height of 1 occurs whenever both dis-

[4] There is always an intersection in the middle because the distributions are symmetric.

tributions are equal. While computational effort is not the focus of this thesis, note that the calculation of the height can (in most cases) be done significantly more efficiently than the calculation of the entire area.
The suggested distributions to use this approach on are the binomial and the beta-binomial distribution. Here, the parameters of the binomial distribution are prob $= \frac{p}{n}$ and $t = p + n$. For the beta-binomial distribution, we set $\alpha = p + 1$, $\beta = n + 1$ and $t = p + n$. In order to draw the distribution of the other class, we set prob $= \frac{n}{p}$, $\beta = p + 1$ and $\alpha = n + 1$ for the respective parameter and leave t the same. Now the area or height can be calculated. Note that for the binomial distribution, if $p = 0$ or $n = 0$[5], the distribution is maximal (1) at 0 or 1, respectively and therefore the method does not distinguish between cases where one instance count is 0 (for example $p = 1$, $n = 0$ and $p = 100$, $n = 0$ result in the same values). This is an important weakness of the binomial distribution in this approach.
An example showing how this approach works can be seen in Figure 4.7.
Since we can choose to use either the area, the height or the normalized height, all of which return val-

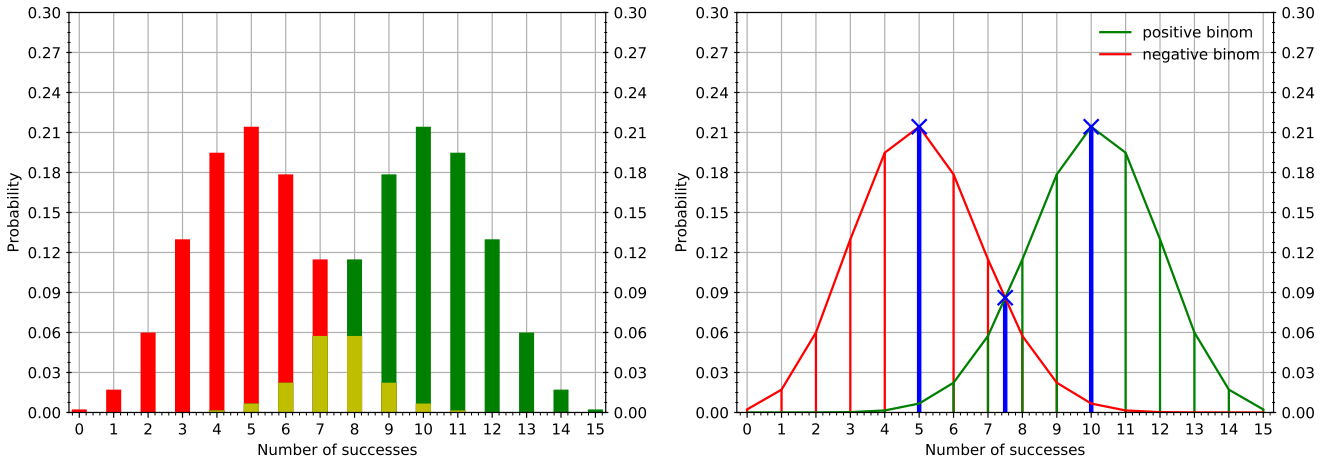


Figure 4.7: Both graphs shown here are built using Binom(k, 15, $\frac{2}{3}$.). The first graph shows how the area between the binomial distribution of the positive and the negative class is calculated. The area is drawn in yellow, showing the intersection of the area of the distributions. The second graphs shows, considering the same distribution functions, how the height is computed. If the point in the middle is not part of the discrete distribution like here (if n is even, taking the height of the distribution at $n/2$ is the correct value), then the height can be interpolated between the two nearest points (blue, in the middle). Normalization of the height can be done by dividing by the maximum height of the distribution (blue, here at 5 and 10 number of successes). In this example, the values are (rounded): area $= 0.18$, height $= 0.09$, maximum_value $= 0.21$ and normed_height $= \frac{\text{height}}{\text{maximum\_value}} = 0.4$.

ues within $[0, 1]$, we denote those values as *confidence_measure*. That means, that a *confidence_measure* can either be the area, height or normalized height of this approach. So how is this measure supposed to be used for prediction? We introduce two possible formulas here. Because we know that a high *confidence_measure* indicates a weak prediction, an easy way to convert that into an prediction values is given by

$$\text{CB}_1(\text{confidence\_measure}) = \begin{cases} 1 - \text{confidence\_measure}, & \text{if pos} \geq \text{neg} \\ -(1 - \text{confidence\_measure}), & \text{else} \end{cases} \tag{4.18}$$

CB_1 makes a statement about much the prediction is in favor of one of the two classes. However, it does not take the probability into account. Imagine a high number of tries t, a probability close to 0.5

[5] $p = 0$ and $n = 0$ is impossible in the decision tree setting.

(for example prob = 0.6) and the binomial distribution. Here, the confidence measure will be close to 0 (because the high value of t will results in a pointed distribution) and therefore the CB_1 will predict something close to 1. So the probability, in this case the aleatoric uncertainty, has almost no weight which might turn out to be an important weakness of this prediction method. Still note that probability is not ignored completely, because for prob = 0.6, it takes a significantly higher t to reach a confidence measure close to 0 compared to prob = 0.9.
Nevertheless, we want to introduce another prediction method which takes the underlying probability into account with more weight.

$$CB_2(\text{confidence\_measure}) = 0.5 + (1 - \text{confidence\_measure})(\text{prob} - 0.5) \tag{4.19}$$

Again, *prob* is the probability derived from the number of positive and negative instances ($\text{prob} = \frac{p}{p+n}$). CB_2 always returns value in $[0, 1]$ but the weaker the prediction is, the closer the predicted value is to 0.5, which indicates no preference for any of the two classes. With the maximum confidence in the prediction, the probability will be returned.
For the rest of this thesis, this confidence bounds idea can be referred to as $CB(p, n)$. At any point where the variants are important, it will be described which variants are talked about. One variant of this idea can be seen in Figure 4.8.

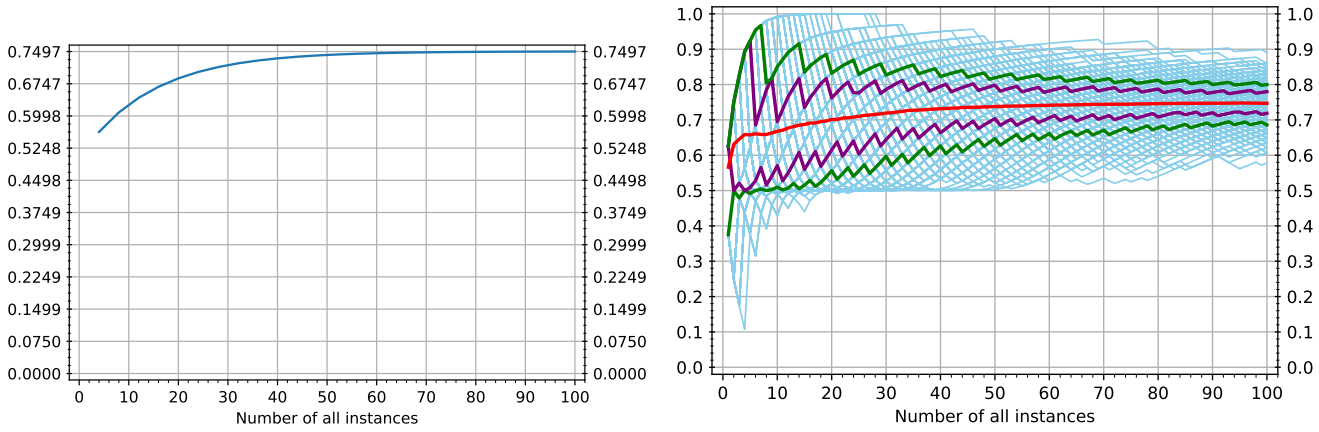


Figure 4.8: Confidence Bounds. Shown here is an example for the confidence bounds prediction. Here, the normalized height for the beta-binomial distribution is used and the second prediction transformation (CB_2) is applied.

5 Aggregation Methods

We consider the predictions of the classifiers (decision trees) within the random forest as the prediction vales x, where x_i is the output value of the ith decision tree. There are N decision trees in a random forest, therefore it always holds that $i \leq N$ or $i \in [1, 2, 3, ..., N]$.

5.1 Simple Aggregation Functions

First of all, there are a couple of rather simple aggregation functions which can be used. The sections here will briefly introduce the average, maximum, median and voting aggregation methods which can be used to combine classification prediction. You can find also find further information in Ponti Jr (2011).

5.1.1 Average

One frequently used combination method is taking the average of your predictions to reach a final prediction. This basically means that every prediction in the ensemble has the same weight assigned to it.

$$\text{average}(x) = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{5.1}$$

You can argue that we do not want the predictions to have the same weight because the different predictions are based on a different number of leafs, i. e. have a varying amount of uncertainty. However, applying the prediction methods of the previous chapter, the idea is that the single prediction values of the individual classifiers already include a level of uncertainty, therefore the average can still make sense. Nevertheless, using the probability for the prediction in the decision trees and the average as the aggregation methods, we indeed can get results which are in contrast to the goal of this thesis.

5.1.2 Maximum

Another popular aggregation method is taking the maximum of our predictions. For example, when aggregating prediction values x with $x_i \in [0, 1]$ where $x_i > 0.5$ indicates a preference for the positive and $x_i < 0.5$ a preference for the negative class, the maximum prediction means taking the single prediction $x_{\max}$ as the final prediction. In this case, it has to be true that $|x_{\max} - 0.5| \geq |x_i - 0.5|$ for all i. Note, that the max prediction is not useful when used on top of probability predictions. Since every pure leaf results in a probability of 0% or 100%, those pure leafs will always have the maximal best prediction. This is also true for a prediction based on one positive and zero negative instances. Anyways, this disadvantage does not hold for the other prediction methods introduced in the previous chapter which leads to the assumption that the max prediction can work better for those.

5.1.3 Median

The median works very similarly to the average. The difference is that, instead of averaging all results, all results get sorted and the result in the middle gets chosen as the final prediction. If the number of predictions which need to get aggregated is not even, the average of the two prediction in the middle will be chosen. All in all, median and average give very similar or even identical results. However, the median prediction is less prone to outliers. Imagine the simple example of three prediction values

$x = (0, 1, 0)^T$. Thereby we have an average of average$(x) = \frac{1}{3}$ but the median predicts the middle value of the sorted tuple $(0, \mathbf{0}, 1)$ which is 0. We can see that this is equivalent to the median of the tuple $(0, 0, 0)$ thereby ignoring the outlier of the first tuple, 1. All in all, I expect average and median to be very similar for the aggregation of prediction in random forest but it is possible that the differences of one of those methods are beneficial to the respective other one in the random forest setting.

5.1.4 Voting

Voting describes the aggregation method for classification in which every classifier in the ensemble votes for a class. In the case of binary classification, that means either voting 0 for the first, or 1 for the second class[1]. Then, the final classification of the ensemble is the class with the most amount of votes.
If you assign, possibly varying, weights to the predictions of each classifier, this aggregation is called weighted voting. If all weights are equal (i.e. you are not using weights), it is called unweighted voting. Note, that if we were to use weighted voting on our prediction methods, the degree of certainty of those methods will be disregarded which results in a loss of information. Since most of the proposed prediction methods would predict the same class on one decision tree[2], disregarding the actual value of the prediction renders the prediction methods useless because the voting results will also be the same.
On the other hand, for unweighted voting we would need to find a way to assign weights to the predictions. Therefore, it seems reasonable to assign weight based on the certainty of the prediction. However, we already did this when considering prediction methods. Therefore, applying unweighted voting on top of that seems redundant. For example, the linear prediction method has a higher value if the (difference of the positive and negative) number of instances is high. All in all, in order to not increase the amount of methods too much, this thesis will not consider unweighted voting aggregation, because of the reasoning that similar weighting is implicitly present in the prediction methods of single decision trees.

5.2 Generalized Mixture Functions

A more advanced method to aggregate classifier results are Generalized Mixture (GM) functions. The idea of GM functions is defining "dynamic weights at the member outputs, making the combination process more efficient" (Costa et al., 2018). Then,

$$H_\Theta(x_1, ..., x_n) = \begin{cases} x_1, & \text{if } x_1 = ... = x_n \\ \frac{1}{n-1} \sum_{i=1}^{n} \left(x_i - \frac{x_i |x_i - \Theta(x_1,...,x_n)|}{\sum_{j=1}^{n} |x_j - \Theta(x_1,...,x_n)|} \right), & \text{otherwise} \end{cases} \tag{5.2}$$

is a GM function where $\Theta : [0, 1]^n \to [0, 1]$ is a function which determines how the GM function works. A very detailed derivation and study of the properties of GM functions can be found in Farias et al. (2016). Most important for the scope of this thesis is that GM functions actually behave quite similar to the respective Θ function. Thereby, some Θ functions which are proposed in the paper by Costa et al. are the average (arithmetic mean), the median and the maximum. This results in the three GM functions H_{avg}, H_{med} and $H_{\max}$. We can get some understanding just by looking at the formulas. It is trivial but also definitely desired, that the prediction of the classifiers is passed through the aggregation function if the values of all predictions are equal. Otherwise, a weight is assigned to every value x_i. Let us say $w(\Theta, x_1, ..., x_n) = \frac{|x_i - \Theta(x_1,...,x_n)|}{\sum_{j=1}^{n} |x_j - \Theta(x_1,...,x_n)|}$. Because of the sum of absolute values in the denominator, we know

[1] It is possible to implement voting also considering the case where no decision is made for a classifier, so that the respective classifier abstains from voting.

[2] The only prediction method introduced in Chapter 4 which can change the predicted class of one single decision tree is the $\text{Prior}_{\text{prev-node}}$ prediction, for every other method the predicted class is equivalent to the predicted class of the probability, only the value of the prediction is differently high.

that $|x_i - \Theta(x_1, ..., x_n)| \leq \sum_{j=1}^{n} |x_j - \Theta(x_1, ..., x_n)|$ and therefore $0 \leq w(\Theta, x_1, ..., x_n) \leq 1$. Now, looking at the formula

$$H_{\text{avg}}(x_1, ..., x_n) = \begin{cases} x_1, & \text{if } x_1 = ... = x_n \\ \frac{1}{n-1} \sum_{i=1}^{n} (x_i - x_i w(\Theta, x_1, ..., x_n)), & \text{otherwise} \end{cases} \tag{5.3}$$

we can see that, within the outer sum $\sum_{i=1}^{n}$, we sum up $x_i(1-w(\Theta, x_1, ..., x_n))$. Therefore, every instance x_i adds between 0 and x_i to the sum, depending on the weight w (or rather $1-w$). Moreover, you can see that x_i has a higher impact on the aggregation if it is close to the value of the underlying aggregation function Θ[3] because in that case $w(\Theta, x_1, ..., x_n)$ approaches zero and x_i gets almost its full weight for the aggregation.
All in all, GM functions try to improve upon their underlying aggregation function Θ by assigning less weights to predictions x_i which are further away from the results of the aggregation function $\Theta(x_1, ..., x_n)$. In their paper, Costa et al. stated that GM functions achieved better results than their respective Θ functions alone when applied with a large number of classifiers in the ensemble.

5.3 Instance Pooling

Considering the goal of this thesis to avoid that small leafs with a high uncertainty have a large impact on our final aggregated prediction, a very straight forward way of dealing with this problem is not even making predictions in the leafs. Instead, the idea of instance pooling is to return the number of positive and negative instances for the respective leaf of every decision tree and aftterwards sum those instances up (pooling). This results in a final number of positive and negative instances, $p_{all} = \sum_{i=1}^{N} p_i$ and $n_{all} = \sum_{i=1}^{N} n_i$, respectively. I previously argued that the probability is an useful mechanism to reach a prediction if the number of instances its based on is significantly high. Here, this is the case, so we can now get to the final prediction by calculating $\text{Probability}(p_{all}, n_{all}) = \frac{p_{all}}{p_{all}+n_{all}}$ as the probability for the positive class.
Also note, that the predictions made by instance pooling are equal to the average of the linear-prediction. However, the AUC score of both approaches can be different and instance pooling might be better when looking at those. Furthermore, this also means that you can see the result of the pooling approach as a confidence in the prediction (we get a probability in $[0, 1]$ for the predicted class). Without further changes, this is not possible for the linear approach and therefore I would argue that instance pooling is more helpful when assessing the worth of the final prediction.

5.4 Beta Distribution Aggregation

The goal of the beta distribution is to estimate the actual probability, in our case based on the number of positive and negative instances in the leaf. A possible way to apply probability distributions to leafs in a decision tree in order to get a prediction value was introduced in 4.6. Contrary to that approach, this section proposes another way of using distributions, particularly the beta distribution, to aggregate multiple leafs to reach a final prediction within a random forest.
First, the beta distribution is calculated for every leaf. This is done in the same way as described before. Afterwards, instead of trying to extract one single value representative for the prediction of the respective decision tree, we keep the distributions and sum them up. That means you have one distribution as an input from every decision tree classifier. The sum of those distributions now forms another distribution. How the aggregation can look like can be seen in 5.1. Any such aggregated distribution however does not trivially lead to a final prediction. One possibility is by making the prediction based on the maximum (the x-value of the graph, basically the *argmax*) of the distribution. Thereby you get a value within $[0, 1]$

[3] More precisely, if this distance (numerator of w) is significantly larger than the sum of all distances (denominator of w).

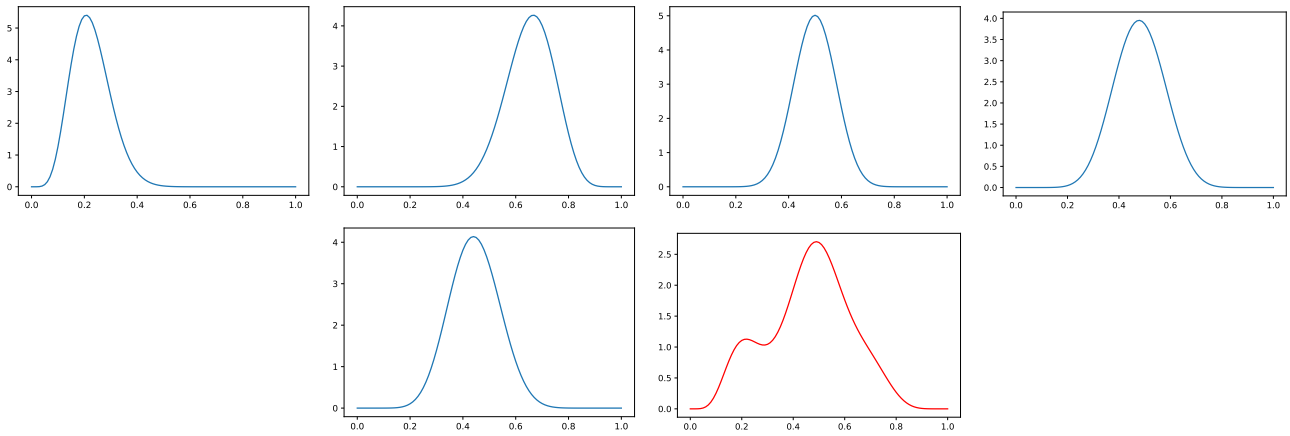


Figure 5.1: The five blue graphs show five different graphs which have been created by plotting the beta distribution on one leaf each. The average of those five distributions is shown by the red graph.

which can be interpreted as the most likely probability and therewith can be chose as the aggregated prediction. Another possibility is by splitting the distribution in two halves at 0.5. If the sum of the area under the graph in $[0, 0.5)$ is higher than the are under the graph in $(0.5, 1]$, the prediction can be made in favor of the negative class. Otherwise, we can predict the positive class. In my opinion, intuitively the maximum approach does make more sense, because the beta distribution aims to find the most likely probability and the maximum of the aggregated distributions hence should be the most likely probability. However, decision trees tend to construct pure leafs if possible. But if you have a leaf where there is only one kind of instance and the number of instance of that type gets higher, the beta distribution of that single leaf gets a very high density at 0 or 1. As a result, single pure leafs can, even with only a couple of instances, influence the maxima strongly, because of those outlier at the interval boundaries. This possible disadvantage does not have a nearly as strong weight when using the prediction based on the area.

5.5 Belief Function Aggregation

Besides the aggregation of the *Plausibility* prediction, we can also think about other aggregation methods based on belief functions which can be used in random forests.
A very simple idea is utilizing the information given by this approach by weighting the predictions by their epistemic uncertainty. This leads us to the *Epistemic-Weight* aggregation.

$$\text{Epistemic-Weight} = \frac{1}{N}\sum_{i=1}^{N} p_{\oplus i} \cdot (1 - u_{e_i}) \tag{5.4}$$

Another idea is trying out Dempster's rule of combination and the cautious rule. The remaining question here is how we get mass functions from the available values ($p_\oplus$, $p_\ominus$, u_a, u_e). We define the mass functions as follows[4]:

$$\begin{aligned} m(\emptyset) &= 0 \\ m(\{+\}) &= p_\oplus \\ m(\{-\}) &= p_\ominus \\ m(\{+,-\}) &= u_e + u_a \end{aligned} \tag{5.5}$$

Now, we can apply Demster's rule of combination and the cautious rule.

[4] It is not ruled out that different transformations into mass functions can also make sense. However, the transformation shown here seemed the most logical and will be used for this thesis.

6 Experiments

This chapter describes how the experimenst were implemented and executed. Thereby, a brief introduction of the framework used, scikit-learn, is given. Afterwards, the dataset and some of its properties are listed and lastly, the setup of the experiments is described.

6.1 Framework

There are a couple of possible ways to implement decision trees and random forests, so it is important to specify which method is used for an experiment. In this thesis, scikit-learn (Pedregosa et al., 2011) is used. Scikit-learn is a software library which provides many machine learning algorithms, including an implementation of decison trees and random forests. I will now give a short overview about those implementations and how you can use them.
Scikit-learn implements, "an optimised version of the CART algorithm"[1]. This also means that regression and classification is supported but with the focus on binary classification in this thesis, only classification trees are used here. However, decision trees in scikit-learn do not support categorical variables which should be kept in mind when looking at the datasets. There are several possible parameters which can be set, among them the splitting criterion (*gini* or *entropy*), the maximum depth of the tree (*max_depth*), a number of required minimum samples in a leaf (*min_samples*), and a random state. The latter can be used to force deterministic behavior of the decision tree building process (like which split is applied when there are two equally good splits for a node). The goal of this thesis is not improving the decision tree building process. Therefore, we stick with the trees created by scikit-learn but we alter the prediction values of the trees by using different prediction methods instead of the probability (see chapter 4).
The scikit-learn implementation of random forests is built on top of decision trees. Thereby, a predetermined number of decision trees is created and the predictions of those trees are averaged in order to get the final prediction. As mentioned previously, it is necessary to use different (random) decision trees within the forest for the random forest ensemble to achieve better results. That is why those decision trees are built from bootstrap samples (bagging) drawn from the training set. Additionally, you can set how many features are considered when splitting a node in the decision tree building process. Using this *max_features* parameter[2], we can make sure that trees are built more varied because different features are used to construct splits in different trees. By default, *max_features* is set to the root of the number of features in the dataset (max_features $= \sqrt{\text{n\_features}}$)[3]. Apart from that, you can also change the same parameters of the decision trees in the same way you can do for single decision trees.
For random forests, we also do not change the way the models are created (except for setting different parameters) but how the prediction is made. Therefore, we apply different prediction methods to the decision trees of the forest and we aggregate those predictions not only by using average but also by applying different aggregation methods introduced in the previous chapter (5).
Because scikit is easier to use with numerical features, only numerical features are used in these experiments.

[1] `https://scikit-learn.org/stable/modules/tree.html`, accessed 12.07.2019
[2] You can also set this parameter in single decision trees, however it usually does not make sense to restrict the considered number of features when only making prediction based on a single decision tree. The default value for *max_features* in scikit-learn decision trees is *None*, i. e. all features are considered.
[3] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, accessed 12.07.2019

6.2 Datasets

In this thesis, 15 binary datasets were used to evaluate the previously introduced methods. Because we want to analyze the effectiveness of the methods across the board, different datasets with different instance and features sizes from different sources were chosen. The "Breast Cancer Wisconsin" dataset is available through scikit-learn. Almost half of the datasets, namely the "Banknote Authentication", "Blood Transfusion Service Center" (Yeh et al., 2009), "Climate Model Simulation Crashes" (Lucas et al., 2013), "Magic Telescope", "MiniBooNE particle identification", "QSAR Biodegradation" (Mansouri et al., 2013) and the "Skin Segmentation" (Bhatt and Dhall) datasets are imported from the UCI machine learning repository (Dua and Graff, 2017). A couple of the datasets used here are arff datasets downloaded online (arf), those are the "Diabetes", "Ionosphere", "Sonar" and "Spambase" datasets. The last three datasets were downloaded using the OpenML (Vanschoren et al., 2013) import function from scikit-learn. These datasets are the "Scene recognition" (Boutell et al., 2004), "SenseIT Vehicle" (Duarte and Hu, 2004) and the "Webdata" (Zeng et al., 2008) datasets. Table 6.1 shows an overview of the size of the datasets by listing the number of instances, the number of features and the class ratio of each dataset. Moreover, for the remainder of this thesis, I will refer to the datasets using abbreviations because the actual name can be rather long in some cases and the meaning of the datasets is not very important here. Therefore, the abbreviations listed in the table are used in order to easily distinguish the different datasets.
Before continuing, there are a couple of things worth noting about the datasets used. First of all, it can be expected that the methods tried in this bachelor thesis might behave differently depending on the size and the properties of the dataset. Especially when looking at different decision tree configurations (*max_depth*, *min_samples*) the tree for different sized datasets will look significantly different. For example, using a large number of *min_samples* can make trees of small datasets very small (up to one node, the root node, only) while trees of the largest datasets are still quite big. We should also keep in mind, that many datasets are not balanced very well (class ratio is far from 50%) which might play a role when looking at how well certain methods behave on certain datasets and why.

Name of the dataset	Abbreviation	instances	features	Class ratio
Banknote authentication	banknote	1372	4	0.44
Blood Transfusion Service Center	transfusion	748	4	0.24
Breast Cancer Wisconsin	breast_cancer	569	30	0.63
Climate Model Simulation Crashes	climate	540	20	0.91
Diabetes	diabetes	768	8	0.35
Ionosphere	ionosphere	351	34	0.64
MAGIC Telescope	telescope	19020	10	0.35
MiniBooNE particle identification	particle	130064	50	0.72
QSAR biodegradation	biodeg	1055	41	0.34
Scene recognition	scene	2407	299	0.18
SensIT Vehicle	vehicle	98528	100	0.50
Skin Segmentation	skin	245057	3	0.79
Sonar	sonar	208	60	0.47
Spambase	spambase	4601	57	0.39
Webdata	webdata	36974	123	0.24

Table 6.1: Overview over the datasets used. Here, you can see the name of the dataset, the abbreviation that will be used in the following parts of this thesis, the number of instances and the number of features of the respective dataset, as well as the class ratio, i. e. how well balanced the dataset is in regard to class distribution. Keep in mind that the number of features listed here do not include the class label.

6.3 Setup

Two major experiment setups were used in this thesis. First, decision tree prediction methods (Chapter 4) were tested on single decision trees. Afterwards, a similar experiment was conducted which tested prediction and aggregation methods (Chapter 4 and 5) on random forest. While the latter is more applicable to the goal of this thesis and therefore more important, we can still get some insight from looking at methods on single decision trees. As previously mentioned, all experiments use the scikit-learn implementation of decision trees and random forests, thereby adding additional prediction and aggregation methods.

6.3.1 Decision Tree Experiment Setup

For the first experiment, comparing accuracy scores is not very helpful because all but one prediction method predict the same class. The difference only lies in the actual scores resulting from the respective leaf used for the prediction. Therefore, the area under curve (AUC) is calculated to evaluate the quality of the results. This makes sense because when aggregating trees we want that the more certain, better predictions are valued higher than uncertain predictions. Since AUC considers the prediction value and not only the prediction itself, we expect to achieve representative results this way.
For every dataset, several decision tree parameter combinations are tested. We set the *min_samples* parameter of the scikit-learn DecisionTreeClassifier to $2^n, n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and both criterions *gini* and *entropy* are tested for every *min_samples*. Additionally, a 2-fold cross-validation (CV) is used for every parameter combination. This CV is repeated 5 times using different random states, which results in 10 AUC values for every parameter combination and prediction method. For the evaluation, those 10 results are averaged (averaging of the CV).
Overall, 24 methods are tested, 18 of which beeing variations of the *confidence bounds* idea (see section 4.6, more details about the variations follow in the evaluation chapter). The other 6 methods, as introduced in chapter 4, are *Probability*, *Linear*, $\text{Prior}_{1,1}$, $\text{Prior}_{\text{prev-node}}$, *UCB-DT* and *Plausibility*.

6.3.2 Random Forest Experiment Setup

The second experiment is aimed at finding out how the classification results of random forest differ and possibly improves when using alternative prediction and aggregation methods. Therefore, we look at the accuracy (for differences and possible improvements in classification) as well as the AUC (to make it easier to find differences if accuracies are similar) scores. The number of estimators (decision trees) within the random forests is 50 in all cases. In practice, you might want to have an even higher number of trees but because running the experiment for every configuration and method is fairly time consuming, we stick with 50 estimators, which should be enough for a suitable evaluation. Like in the previous experiment, a bunch of different configurations are executed. For every random forest, either the *min_samples* parameter is set to $2^n, n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ or the *max_depth* parameter is set to $x \in \{2, 3, 5, 7, 10\}$. Examining all combinations of those two parameters is in many cases redundant because one parameter will "dominate" the other one[4]. For other cases, both parameteres will do roughly the same. Nevertheless, there are definitely cases which construct different trees when combining both parameters. However, looking at those cases is most likely not worth the extra time needed to run this setup. Moreover, it is easier to evaluate the results when we know which parameter is responsible for building the tree in the way it was built. Just like in the previous experiment, we also again tried out both splitting criterions supported in scikit-learn, *gini* and *entropy*.

[4] Imagine min_samples = 64 and max_depth = 3. The trees of large datasets will be heaviliy restricted because of the *max_depth* value, making the *min_samples* parameter superfluous. Similarly, for small datasets, the *max_depth* parameter will not matter.

For this experiment, we looked at 55 prediction / aggregation methods. However, this number comes from the fact that we can use the same prediction values of the decision trees but aggregate them in different ways. Because of long run times, only one *confidence bounds* variation was tested in this experiment (detailled explanation in the evaluation chapter). Besides that, all prediction methods from the decision tree experiment were also applied here. To aggregate one prediction per tree for a final prediction, we looked at 7 different ways of doing so, averaging, median, max and their respective GM variants, as well as unweighted voting. Those are $7 \cdot 7 = 49$ of the 55 variations. Furthermore, we took a look at *instance pooling*, the two proposed possibilities to combine beta distributions (max or area) for a final prediction, dempster's rule of combination, the cautious rule and the *Epistemic-Weight* aggregation. For the dempster's rule and the cautious rule, clipping on mass functions was applied so we can make sure that a combination is always possible. This is because if the mass function of {+} or {-} is 1, this would usually mean that the respective class must be true. However, this is not the case in our application (due to numerical inaccuracies, in theory the respective values can never become exactly 1 but only close to 1) which is why we have to make sure that those value are always < 1 which is done by clipping, i.e. decreasing the value of those mass functions (here, 1−0.00001 is used for clipping).

7 Evaluation

This chapter covers the evaluation of both experiments. First, the decision tree experiment will be analysed and afterwards an evaluation of the random forest experiment will be conducted.

7.1 Decision Tree Experiment Results

As described in the previous chapter, the experiment returned a lot of results because of the high number of configurations. Although the goal of this thesis is not to find alternative ways of building decision trees, we still not only want to find out how different prediction methods behave on specific trees but we want, at least to a certain degree, find out which methods benefit of which tree structures. In other words, only because one method performs best on a certain type of tree (for example large, unpruned trees) and another methods performs worse on the same kind of tree, that does not mean that this second method is not the overall best performing method when considering a different type of tree. This is why we take a look at a variety of tree building parameters and we hope to gain valuable insights that way, however, even this setup is obviously not exhaustive. For this experiment, we have 15 datasets, 2 splitting criteria, 10 different *min_samples* values, 10 iterations (resulting from 5 times 2-fold CV) and 24 different prediction methods. This makes a total of

$$15 \cdot 2 \cdot 10 \cdot 10 \cdot 24 = 72000$$

results to be used for the evaluation. To reduce randomness and the influence of outliers, we averaged over the CV results, therefore having $\frac{72000}{10} = 7200$ results to work with.
In order to deal with the number of results, a statistical evaluation of the decision tree experiment is conducted. Following an approach suggested by Demšar (2006), first, a Friedman test (Friedman, 1937) is applied, which can be followed by a Nemenyi test (Nemenyi, 1963). By applying the Nemenyi test on top of the Friedman test, we can state which methods perform significantly better than other methods. However, there will be some cases where the differences of the ranks in the Friedman test will not be significant. Nevertheless, it might still be worth to recognize those small differences, too. Therefore, I will make clear when we talk about statistically significant differences and when we only talk about slight differences in the rank, which have no statistical significance.
First, I will analyse the differences between the confidence bounds method variants. This makes it possible to only consider the better variants, thereby reducing the complexity of the following analysis. After that, I will take a look at how the different decision tree parameters influence the strength of the prediction. Here, I will generalize over the prediction methods. A closer look at how the prediction methods behave on different decision trees will be done afterwards.

7.1.1 Confidence Bounds Variants Comparison

In this decision tree experiment, 18 of the 24 prediction methods are based on the confidence bounds idea. Considering those 18 methods, we expect those to share some properties. So, in order to split the evaluation of 7200 results in smaller parts, a comparison of the confidence bounds variations among themselves seems useful.
First of all, what even are the 18 variations used? The following explanation briefly repeats the ideas introduced in 4.6 and states some abbreviations in brackets[1] which allow us to identify those variations

[1] Thereby, any method can be describes by its distribution, the way it gets a value used for the prediction, and the way this value is used to return the final prediction value. For example, a variation can be *beta_norm-height_value*.

without having to write half a sentence each time when naming one. For this experiment, we applied the confidence bound idea on the binomial (bin), beta (beta) and beta-binomial (beta-bin) distribution. For every distribution, we looked at the prediction using the area (area), the height (height) and the normalized height (norm-height). Moreover, for every of those 9 possibilities, we tried out two ways to get to the final prediction. Once, by using that value itself (value), only with a different sign depending on the predicted class and once by generating a probability which lies between 0.5 and the class ratio probability of the respective leaf (prob).
When comparing the ranks of those 18 variants, we get the ranks shown in Table 7.1. Beyond that, I conducted a couple Friedman Tests on the data to find the best variations. Therefore, my goal was finding out how 1. the probability distributions, 2. the value calculation method and 3. transformation of that value into the final prediction compare to their alternatives. For this reason, I fixed the specific values of two of those 3 factors and only compared the alternatives of one factor. For example, for a comparison of the distributions, I first compared bin_area_value, beta_area_value, and beta_bin_area_value and looked at the result of this comparison. Afterwards, I did the same using the other 5 combinations you can get from using one of (area, height, norm-height) and (value, prob) each. Using this procedure, I can make sure to appropriately compare the parts of the variations. Anyway, you can also get a good impression by just looking at the table. This analysis gave the following insights. When comparing the probability distributions, the beta-binomial distribution always got the best results with a statistically significant difference[2]. It is not surprising, that the binomial distribution falls short of the other two distributions, since when looking at distributions build on pure leafs, we do not get values which represent the uncertainty of the respective leafs. It is possible that the beta distribution performs worse than the beta-binomial distribution because it is continuous and maybe discrete probability distributions are better suited for those approaches. Furthermore, you can also see that the beta distribution performs much worse for the unnormalized height. This is a result of its continuous nature because resulting values can be arbitrarily high when no normalization is applied. Also, the prob prediction works significantly better than the value prediction for every conducted Friedman test. This indicates, that the intermediate values we extract from the probability distributions rather model uncertainty than estimating a probability. As a result, mixing the class ratio of that leaf into the final decision gives better results. When comparing area, height, and norm-height however, the results were not quite as clear. As just noted, for the beta distribution, the height is worst alternative and the area seems to be slightly better than even the normalized height. The binomial distribution however got better results for the height in both cases but here all three alternatives are very close to each other[3]. Normalizing the height achieved the best results when using the beta-binomial distribution but only by a small margin.
To sum up, of all the variants tested, the beta-binomial distribution with the final prediction incorporating the probability works quite a bit better than the respective alternatives. For the calculation of the value based on the distribution, the normalized height seems to be a little bit better than the unnormalized height. While there are cases where the area achieved better results, we conclude that the normalized height seems to be the best alternative. Still, the difference here is very small and we can not say that the area is strictly worse. All in all, we take the beta-binomial distribution using the normalized height calculation and the final prediction including the probability as the best of the 18 variants which will be used representative for the confidence bounds idea for the rest of this thesis.

7.1.2 Decision Tree Parameter Comparison

When comparing the different *min_samples* values for the experiment[4], we get the average ranks shown in Table 7.2. We can see that, on average, $min\_samples = 16$ has the best results. That fits our expectation that unpruned trees are overfitting but too large trees are underfitting. However, we should not conclude

[2] Following, *significant* will always mean *statistically significant*.
[3] But due to the fact that the binomial distribution is much worse overall, this was not investigated further.
[4] Those values are obtained by generalizing over the prediction methods.

value	bin	beta	beta-bin
area	14.558	9.838	7.145
height	14.065	13.788	7.802
norm-height	13.703	9.577	6.432

prob	bin	beta	beta-bin
area	11.163	7.162	5.495
height	10.820	10.090	5.457
norm-height	10.988	7.628	**5.288**

Table 7.1: The table was built comparing the ranks of those 18 confidence bounds variants. For better visibility, the *value* and the *prob* variations (how to get to the final prediction value) are shown in two seperate tables.

that *16* is the best or even somewhere around the best *min_samples* value to fit trees with because this parameter mostly depends on the size and the variance of the dataset. If the dataset has zero noise and is large enough, unpruned trees can give an accuracy of 100%. However, this assumption is unreasonable. Therefore, the results about the quality of different *min_samples* parameter are not surprising and do not give new insights.
However, one thing we can keep in mind for the random forest experiment is that *entropy* seems to work better for larger trees compared to *gini*. While this does not mean that *entropy* generally performs better, another Friedman test comparing only these criteria gave an average rank of 1.330 for *entropy* and an average rank of 1.670 for *gini*. This difference even increased when only looking at large (*min_samples* = 1 or *min_samples* = 2) trees (1.257 for *entropy*, 1.743 for *gini*).

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.386	5.200	4.948	4.057	**3.214**	3.810	5.024	6.362	8.090	8.910

(a) Gini and Entropy

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.162	4.943	4.781	4.000	**3.381**	4.105	5.100	6.519	8.081	8.929

(b) Entropy

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.610	5.457	5.114	4.114	**3.048**	3.514	4.948	6.205	8.100	8.090

(c) Gini

Table 7.2: These tables show the average ranks of the decision tree parameter *min_samples*. The first table only compares those, generalizing over every other parameter and method (7.2a). The second and third table only consider trees built using *entropy* (7.2b) or *gini* (7.2c), respectively.

7.1.3 Decision Tree Prediction Method Comparison

Let us again start with a table comparing the average ranks of our seven remaining prediction methods (7.3). It is promising to see that the probability performs worse than every other method. The $\text{Prior}_{1,1}$ method gets the best results, with the other methods ranking in between $\text{Prior}_{1,1}$ and the probability. Furthermore, we can see that for fully grown trees, the probability actually performs terrible in comparison to any other method. This is a result of the amount of small, pure leafs, which give a terrible estimation because they always return a probability of either 0 or 1. For such large trees, the *Linear* prediction method actually performs better and $\text{Prior}_{1,1}$ falls down a bit, otherwise the order is similar. Interestingly enough, for small trees, the probability actually gets very good results, comparable to *Plausibility* and *CB* and only beaten by $\text{Prior}_{1,1}$. Not shown here is, that the criteria *entropy* and *gini* have no obvious influence of the performance of the methods in comparison.

Overall, the probability estimation based on the leafs seems to be a very good estimation for leaf predictions, but only if there is enough data in the leafs (i. e. a high number of instances). However, this might lead to underfitting. Therefore, when looking at a single decision tree, all proposed methods achieved better AUC score (better ranking) than the probability in general. However, only adapting the probability slightly as done by $\text{Prior}_{1,1}$ got the best results for this experiment. That is probably because it maintains the accuracy of the probability for a high number of instances (value nearly the same) but it incorporates uncertainty for smaller leafs, thereby reducing the main disadvantage of the probability. While it seems that $\text{Prior}_{1,1}$ is the best choice for single decision trees, there are more factors to keep in mind when looking at random forests. Therefore, the other methods should not be ruled out (yet).

prediction method	Probability	Linear	$\text{Prior}_{1,1}$	$\text{Prior}_{\text{prev-node}}$	UCB-DT	Plausibility	CB
rank	5.323	4.212	*2.742*	4.410	3.548	4.078	3.687

(a) All decision trees

prediction method	Probability	Linear	$\text{Prior}_{1,1}$	$\text{Prior}_{\text{prev-node}}$	UCB-DT	Plausibility	CB
rank	7.000	*2.433*	2.892	4.683	2.642	4.483	3.867

(b) Large trees

prediction method	Probability	Linear	$\text{Prior}_{1,1}$	$\text{Prior}_{\text{prev-node}}$	UCB-DT	Plausibility	CB
rank	3.642	5.087	*2.987*	4.725	4.325	3.654	3.579

(c) Small trees

Table 7.3: This table shows the average ranks of the 7 remaining prediction methods, excluding all but one of the confidence bound variants. CB is that confidence bound variant, as described earlier in this chapter. The first table generalizes over all tree parameters (7.3a). Also shown are the ranks of the methods for very large trees (*min_samples* ≤ 2, (7.3b)) and smaller trees (*min_samples* ≥ 64, (7.3c)).

7.2 Random Forest Experiment Results

The random forest experiment evaluation is based on the AUC metric because that makes it easier to see differences than looking only at the accuracy.

7.2.1 Preliminary Evaluations

The decision tree experiment compared different variants of the confidence bounds idea. We found out, that, of all tested variants, the beta-binomial distribution worked best. Also, the prediction using the normalized height achieved slightly better results than using the area. While this difference was rather small, using the height also makes it so the calculation can be done much more efficiently and does not scale with the number of instances in each leaf (because for the area, you need to calculate the whole probability distribution), thereby being much more reasonable to run for larger experiments.

7.2.2 Random Forest Parameter Comparison

Before considering the methods, we will again briefly look at the random forest parameter, i.e. which trees work better in a random forest[5]. In single decision trees, large trees often overfit and small trees

[5] Again, this is done by generalizing over all prediction methods.

often underfit the data. Therefore, restricting the tree size a little and avoiding leafs with only one or slightly more instances gives the best results on average. In the case of random forests, there are two additional factors to keep in mind. Firstly, trees are built with randomness, thereby probably being less effective alone as decision trees built without randomness (random feature selection). Secondly, it does not necessarily hurt our overall prediction if some decision trees within the random forest make false prediction because they only play a small role in the ensemble. So what does change when looking at the tree parameters in random forests compared to single decision trees?

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.152	4.315	**4.283**	4.692	5.775	7.273	9.108	11.524	12.655	13.383

max_depth	2	3	5	7	10
rank	11.325	9.768	7.888	6.891	5.965

(a) Gini and Entropy

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.029	**4.316**	4.356	4.642	5.744	7.417	9.256	11.526	12.672	13.385

max_depth	2	3	5	7	10
rank	11.347	9.825	7.909	6.850	5.724

(b) Entropy

min_samples	1	2	4	8	16	32	64	128	256	512
rank	5.276	4.315	**4.210**	4.741	5.807	7.130	8.961	11.522	12.638	13.380

max_depth	2	3	5	7	10
rank	11.304	9.712	7.868	6.932	6.205

(c) Gini

Table 7.4: These tables show the average ranks of the decision tree parameters *min_samples* and *max_depth*. The first table only compares those, generalizing over every other parameter and method (7.4a). The second and third table only consider trees built using *entropy* (7.4b) or *gini* (7.4c), respectively.

The first thing that is apparent when looking at the parameter (7.4) is that, compared to the analysis of decision trees, even larger trees perform better. Previously, the best *min_samples* value was 16 and now it went down to 4. The reason for this is probably because small leafs allow the decision trees to cover more information but the main disadvantage, overfitting, is reduced by using multiple trees in the random forest ensemble. Furthermore, nothing indicates that choosing a particular *max_depth* parameter can greatly benefit the performance of random forests. In general, *max_depth* is very similar to *min_samples* anyway, it seems like another approach to the same problem. In particular, the problem being overfitting and the solution reducing the size of the tree in order to generalize better. Thereby, choosing a specific *min_samples* parameter is arguably more appropriate when looking at multiple datasets. Particularly for this thesis, *min_samples* can guarantee us that we have x or more number of instances which are used for every prediction in a decision tree. Thereby, we can reduce the epistemic uncertainty (while risking underfitting). The goal of *max_depth* is the same but cutting a tree at a depth makes it so small datasets might be (completely or almost) fully grown on that depth while larger datasets are still underfitting due to the large amount of features and instances. However, setting *max_depth* can make sure to reduce the memory consumption of a model which is more difficult to estimate when using *min_samples*. However, for the further analysis, we will not focus on the *max_depth* parameter much because it is easier to understand trees built using *min_samples* and because the tested

values of *max_depth* do not indicate any special advantage.
A comparison of the splitting criteria gives comparable results to the decision tree experiment. In general, the average rank of *entropy* (1.252) succeeds that of *gini* (1.748). Again, this difference slightly increases when looking at large trees (1.222 *entropy*, 1778 *gini*) and decreases when considering smaller trees (1.300 *entropy*, 1700 *gini*). This also matches the result shown in Table 7.4, where we can see that *entropy* gave better results for *min_samples* $= 2$ as opposed to *min_samples* $= 4$ (still, this difference is very small).
Overall, it seems like larger trees are outperforming smaller trees even more than for single decision trees. For the following analysis, it will be interesting to see how and which methods perform better for *min_samples* between 1 and 16 since larger values clearly give worse results (trees with higher *min_samples* or trees using the *max_depth* parameter will no longer be considered).

7.2.3 Random Forest Methods Evaluation

The large number of prediction methods, aggregation methods, and parameter configurations makes evaluating the methods rather difficult. Ideally, you would like to rank the methods in terms of performance from best to worst. However, there are at least two factors which make this very difficult if not impossible. One problem is that the averaged performance of the methods in our experiment is not sufficient to appropriately assess the strengths and weaknesses of the methods. For example, the method which is the best on average might be worse than another method for a specific parameter configuration. Now, maybe that second method would actually get the best results overall on the parameter configuration and thereby this would be the best approach for random forest classification. Another problem we are dealing with is that the performance of many methods is very similar. Many instances get the same prediction either way and slight differences in the performance might just be a result of randomness in the data or the classifier. In order to deal with the extent of this evaluation, we will take a look at which methods performs worse in general and afterwards only evaluate the best methods more closely.

Simple Aggregation and Generalized Mixture Functions

Of the 55 overall prediction methods, 49 are different aggregation of the same values calculated from decision trees. Can we find aggregation functions that clearly perform better than others? Table 7.5 shows, how the aggregation functions perform for the respective decision tree prediction methods. Thereby, a Friedman test was conducted for each of the seven prediction methods, comparing the seven aggregation methods. While these tests do not compare the prediction methods among each other, you can get a feeling for the performance when looking at the voting aggregation, since voting has the exact same results for six of the seven prediction methods, excluding $\text{Prior}_{\text{prev-node}}$. Before even comparing the aggregation functions, it can be seen that for *Linear* and *UCB-DT*, voting gave the best results. Since voting does not even consider the prediction methods, we can conclude that those two prediction methods are not very good overall. Moreover, the *Linear* prediction with max aggregation gave some of the worst results in general (the results were not bad per se but significantly worse compared to other methods). This fits the idea that the ensemble learning of random forests really helps to correct errors of the decision trees within that. Furthermore, we can see that in general aggregation methods which only forward the prediction of one decision tree per instance (maximum and median) give worse results than aggregation methods which mix the predictions of the trees. On the other hand, using those functions as a baseline for the aggregation using GM-functions can work well.

	Average	Median	Max	Voting	GM-Avg	GM-Med	GM-Max
Probability	3.177	4.290	6.831	4.200	3.157	3.307	**3.057**
Linear	4.887	2.330	6.667	**2.123**	3.863	3.363	4.767
$\text{Prior}_{1,1}$	3.587	4.070	6.693	3.847	3.373	3.263	**3.167**
$\text{Prior}_{\text{prev-node}}$	3.017	4.760	5.637	5.067	3.197	3.547	**2.777**
UCB-DT	4.306	3.067	6.683	**2.877**	3.713	3.393	3.907
Plausibility	3.177	4.387	6.660	4.223	3.263	3.373	**2.917**
CB	3.490	4.073	6.707	3.800	3.440	3.420	**3.070**

Table 7.5: This table shows the performance of seven prediction methods with seven aggregation methods. As previously mentioned, only random forests built with min_samples ≤ 16 are considered.

Futher Aggregation Methods

For now, let us consider the generalized mixture variant of the maximum aggregation for the *Probability*, $\text{Prior}_{1,1}$, $\text{Prior}_{\text{prev-node}}$, *Plausibility* and *CB* prediction methods. Thereby, we do not have too many methods and we can compare those and the other six methods we did not look at in the previous step. Doing so, we get the results of Table 7.6.

method	Probability	$\text{Prior}_{1,1}$	$\text{Prior}_{\text{prev-node}}$	Plausibility	CB
rank	**3.430**	4.227	8.223	3.550	4.013

method	Pooling	Beta-Sum	Beta-Max	Dempster	Cautious	Epistemic-Weight
rank	8.073	6.040	8.417	5.860	10.310	3.857

(a) min_samples ≤ 16

method	Probability	$\text{Prior}_{1,1}$	$\text{Prior}_{\text{prev-node}}$	Plausibility	CB
rank	4.233	4.600	6.850	3.733	4.250

method	Pooling	Beta-Sum	Beta-Max	Dempster	Cautious	Epistemic-Weight
rank	8.533	6.383	7.967	5.767	9.967	**3.717**

(b) min_samples $== 1$

Table 7.6: These tables show the average ranks across 11 of the tested prediction methods. To reduce the complexity and the visibility of this comparison, only five of the 49 methods, which have been inspected in the previous section and had good results, are considered. Therefore, the five methods in the respective upper tables use the generalized mixture version of the max function as the aggregation method. The first table (7.6a) considers tree with min_samples ≤ 16, the second table (7.6b) only trees built using min_samples $== 1$.

The first thing which is probably surprising is that the probability is the best methods when considering trees until min_samples ≤ 16. Other methods which are more or less close to the performance of probability are $\text{Prior}_{1,1}$, *Plausibility*, *CB*, *Beta-Sum*, *Dempster* and *Epistemic-Weight*. The methods $\text{Prior}_{\text{prev-node}}$, *Pooling*, *Beta-Max* and *Cautious* perform quite a bit worse. Anyway, comparing eleven methods over two tables can be confusing, so let us look at every method individually. Now, I will first describe what can be concluded by the Friedman tests for each method and afterwards I will pose questions about the performances of some methods which might be surprising when comparing it with other methods.
The probability gives the best results even for rather large trees. However, when we only look at fully grown trees, the two similar *Plausibility* methods achieve slightly better results. Following the motiva-

tion of this thesis, it is not surprising that the performance of probability gets worse on larger trees, it is however surprising that it still performs better than most methods and is still the best on relatively large (but not fully grown) trees. $\text{Prior}_{1,1}$ has good results but always stays behind the probability in performance which is contrary to the single decision tree experiment. The fact that $\text{Prior}_{\text{prev-node}}$ achieves bad results makes sense when considering that large trees performed better in the random forest environment. Therefore, considering the nodes of the depth before the leafs seems to be inferior to the leafs themselves. *Plausibility* and *Epistemic-Weight*, two very similar methods in their nature, achieve very good results, particularly on unpruned trees. Thereby, it seems like the inclusion of epistemic uncertainty helps those methods to perform better. The confidence bounds method find itself among the good, but not the best methods. *Pooling* of the leafs also does not give great results, this can be justified in the same way as the *Linear* prediction method, as that it ignores a main advantage of ensemble learning methods by focusing to much on predictions of large leafs only. The beta distribution aggregation methods had decent if not bad performances. The fact that the max aggregation of those has the best results can be reasoned by the strong influence of large, pure trees on the aggregated distribution. Contrary to expectations, the Cautious rule was considerably worse than Dempster's rule, which actually performed rather well, which is counter-intuitive because the Cautious rule was meant to compensate a disadvantage of Dempster's rule that our items of evidence , i. e. the instances in the leafs, are not distinct amongst the random forest.
So, as far as possible, it was argued why certain methods performed better or worse than other methods. Before I continue, keep in mind, that we are still comparing average ranks. Therefore, we do not say that any method is bad per se but that some methods are just slightly better than others. Nevertheless, the methods are all not bad for random forest classification and the actual differences in accuracy are minimal most of the time. However, we would like to find out what the advantages and disadvantages of certain methods are, thereby only considering the differences, even if those are small.
Some of the methods performance are surprising and should be looked at more closely.

1. Why does the probability perform so well even if the decision tree experiment showed it performing worse than any alternative?

2. Why has $\text{Prior}_{1,1}$ worse results compared to the decision tree experiment, moreover, why is it worse than the probability?

3. Is it possible to find out what the plausibility approaches do better than the other methods, particularly for larger trees?

4. All in all, what is the best method to use? (How) Does it depend on the dataset and which parameter should be chosen?

The following sections try to go into detail about those questions.

Performance of the Probability Prediction

We want to find out why probability prediction works so well. Therefore, we can look at Figure 7.1. The first row shows heatmaps which indicate which prediction of single leafs in a random forest match the true prediction. Thereby, we insert a point with 0 into a cell whenever the prediction does not match (more positive than negative instances in that leaf and the prediction is negative or vice versa) and we insert 1 if it does match[6]. The heatmaps display the average of those counts. Thereby, the x-axis shows the number of instances in the respective leaf and the y-axis shows the class ratio (hereby including the lower bounds and excluding the upper bound, an exception is the 90-100 range, where 100 is included). These heatmaps show, that larger leafs with a clearer class ratio more often contain the true prediction

[6] In case that number of positive and negative instances are equal, 0.5 is inserted.

than smaller or more mixed leafs. For telescope, the accuracy is lower starting at a class ratio of ≈ 50%. This can be explained by the fact that this dataset is not as balanced as the other two and only 35% of the instances there are positive. Therefore, predicting the negative class is true more often.
The next two rows show the same heatmap but on different color mappings. These two heatmaps get the same points (class ratio and instance number) as the first heatmaps but here the value is 1 if the prediction of the forest of the respective instance is true and 0 otherwise. In other words, leafs can be inserted with 1 even if their own prediction is false if the random forest ensemble corrects that prediction. The first of those two rows (second row overall) can be compared with the first row of heatmaps. We see that although many mixed and small leafs often have bad predictions, they overall perform very well in the forest. We can conclude that the ensemble learning of the forest adds significant value to the predictions. The next row can be viewed seperately. While all leafs give good predictions on average, we can see that mixed leafs are less helpful compared to pure leafs. Also small, pure leafs are false more often than larger leafs. However, it seems like even the smallest leafs (size 1) do not perform worse, maybe even better, than larger, mixed leafs. This might indicate that, even for probability, we prefer small leafs over larger, less pure leafs.
The fourth row of heatmaps is the difference of the first and second (or third) row of heatmaps (order: second minus first). This means we can see which leafs profit the most of being part of a random forest ensemble, as opposed to predicting the class itself. We can see that the difference in large, pure leafs is very small. Those most of the time predict the correct class in either case. However, we can see a clear improvement on more mixed and on smaller leafs. The improved performance on mixed leafs makes sense since the predictions of those leafs automatically have a lesser influence on the final prediction[7]. For smaller leafs, this is more surprising because those leafs in particular were expected to worsen the performance of the forest, avoiding which was the motivation of this thesis. However, it seems like the randomness and the construction of trees in random forests makes it so even those predictions are more often than not corrected for the final random forest prediction. This can also be seen as a confirmation of Wyner et al. who claimed that random forests, just like AdaBoost, build models which are not prone to overfitting. Furthermore, this would mean that the weakness of random forests which build the motivation of this thesis is rather a weakness of single decision trees and less a weakness of random forests. It seems as random forest in their nature are less prone to problems of that kind. All in all, when we argue that the probability is always a good measure when considering large leafs and we now know that small leafs are not a large problem either, it makes sense that the probability prediction performance could not be surpassed significantly by any other method.
The last row of Figure 7.1 shows the number of instances per cell which were used to construct the upper heatmaps.

Analysis of the $\text{Prior}_{1,1}$ Prediction

When comparing the performance of the $\text{Prior}_{1,1}$ and the probability predictions on different leafs (Figure 7.2) we can find some small differences. Overall, both methods are very similar but we can find differences for smaller leafs. For larger leafs, the heatmaps show almost identical behavior, which makes sense since both prediction methods are very similar in that case. For small instances however, there are some deviations but it is difficult to draw conclusion out of those. We can see that particularly the leafs with one instance work better for the probability (highest negative number in the comparison $\text{Prior}_{1,1}$ minus probability). For other small, but less pure instances, sometimes the opposite is the case. However, we know that the probability overall performs just slightly better most of the time. I think the reason here is that random forests, as shown in the previous section, are very good at dealing with small leafs despite their epistemic uncertainty and that they utilize the information given in small leafs because this

[7] For example, two predictions of 40% and 100%, respectively, combined result in a prediction of the postive class, therefore the 100% prediction has a larger influence.

information can be valuable as well (i. e. the prediction of that leaf can be true, arguably with a probability of $\geq 50\%$). The $\text{Prior}_{1,1}$ prediction however shifts the prediction from the pure values 0 and 1 to a more neutral value, 0.5. While the purpose of reducing uncertainty seems useful, the experiments show that it does not work as well as the probability itself. Therefore, I think that the loss of information by the $\text{Prior}_{1,1}$ methods is actually a larger disadvantage than the reduction of uncertainty is an advantage for that method.

Analysis of the Simpler Plausibility Approaches

Particularly for large trees, the *Plausibility* prediction methods and the *Epistemic-Weight* prediction performed very well. What both of those methods do is calculating prediction values by taking the epistemic and the aleatoric uncertainty into account. Figure 7.3 compares the methods with each other. All in all, the differences are small and seem to be rather random. I do not think that I can reliably draw conclusion here but I would argue that the *Plausibility* prediction is more appropriate because it already incorporates the epistemic uncertainty and thereby *Epistemic-Weight* is doing redundant work. However, the experiment has shown that this redundancy is barely noticeable when comparing the results of both methods. A comparison of the *Plausibility* and the probability prediction is done in Figure 7.4. However, I find it difficult to draw conclusions here. We can see that differences are more noticeably for smaller leafs. It seems like *Plausibility* tends to perform slightly better on small, mixed ratio leafs but leafs with only one instance give better results for the probability. All in all, I do not know which situation makes*Plausibility* behave better or worse than the probability, what I can say though is that *Plausibility* performed comparably well on the experiments when using large trees. Therefore, this approach definitely has potential and might even be superior in certain situations, maybe when using different learning tasks.

Search for the Best Prediction Method

Overall, it would be great if we could say which prediction method and decision tree parameter are the best to choose. However, the experiment and further analysis have shown that a few of the proposed methods are too similar to be sorted in a order of performance. It is possible to try to find some different behavior, particularly on smaller leafs but all in all it would probably be overstated to say that only one of the proposed methods is superior to every alternative. Not looking at the Friedman ranks but also at the best prediction per dataset does not give much more information. It turns out that larger trees generally behave better, mostly in the range from 1 to 8 minimum instances per leaf (*min_samples*). Furthermore, sometimes the *Plausibility* prediction has the best accuracy on a random forest, in other cases the probability and in some cases even one of the methods which we categorized as *worse* methods can by chance classify an instance correctly which was not correctly classified by the other methods. Therefore, the randomness in the data and the random forest itself can make one method perform better than another one, just by chance.
Nevertheless, some things about the methods in random forests can be said. Firstly, the probability and *Plausibility* prediction methods seem to be the best two alternatives. Thereby, using a GM-function might or might not give slightly better results than averaging but voting, median and max usually perform worse, probably because they are not utilizing all the information which is available in the random forest model. So, if slightly increasing the accuracy of the model is worth the extra time, testing out different prediction methods might be worth it. Overall, I think that using the probability with the average function gives very good results which can not significantly be surpassed by different methods. Moreover, the *Plausibility* approach takes more time for the calculation. On the other hand, the *Plausibility* can give additional information about the uncertainty of the respective leaf prediction but that was not investigated further in this thesis.

Open Questions

There are some method performances which do not fit my intuition. Unfortunately, despite some further testing I was not able to figure out meaningful reason for the differences in performance. Firstly, the GM-functions and their variation. As previously mentioned, all of them consider all predictions in the aggregation, opposed to the regular median and maximum functions which only choose one. This makes them perform better. However, it is unclear to me which of the GM-functions (and the regular average function) perform better or worse in certain situations. In this experiment, the performance of all was very similar.
Another open question is why Dempster's rule of combination achieved better results than the cautious rule. Possible reasons are the numerical inaccuracies inherent to computer science, the measures which were used to allow for the computation (clipping) and the transformation of the plausibility values and the uncertainties into mass functions. Furthermore, it is possible that the belief-function framework does not fit the random forest classification environment well (which might be connected to the fact that clipping was necessary in the first place) . However, Hüllermeier's approach managed to perform well when aggregating the values in a simpler fashion.

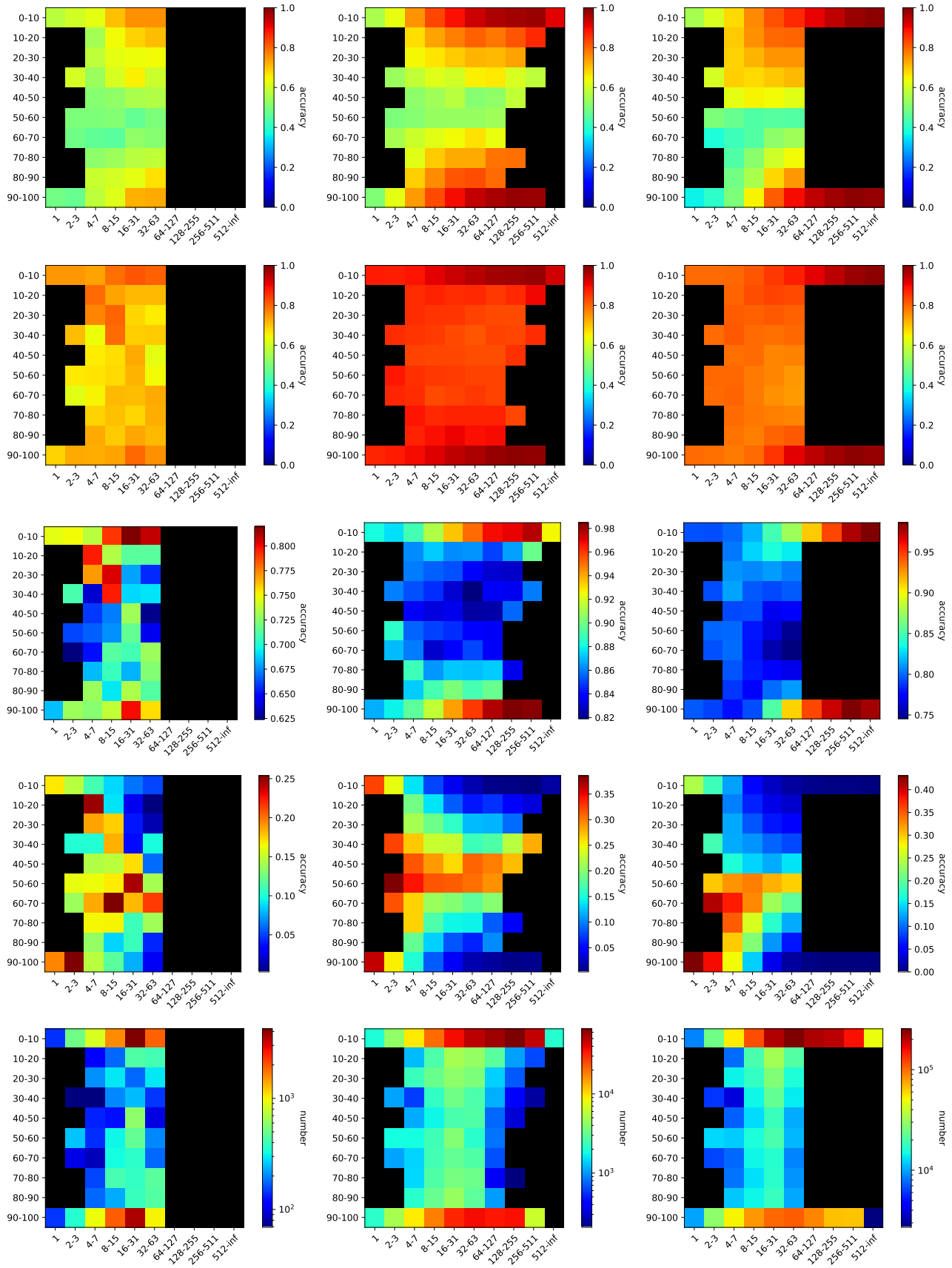


Figure 7.1: Each column shows the same heatmaps for a different dataset. For each row, a different heatmap was created. The datasets are (left to right): sonar, spambase, and telescope. Further explanation on the construction and meaning of the heatmaps can be found in the text.

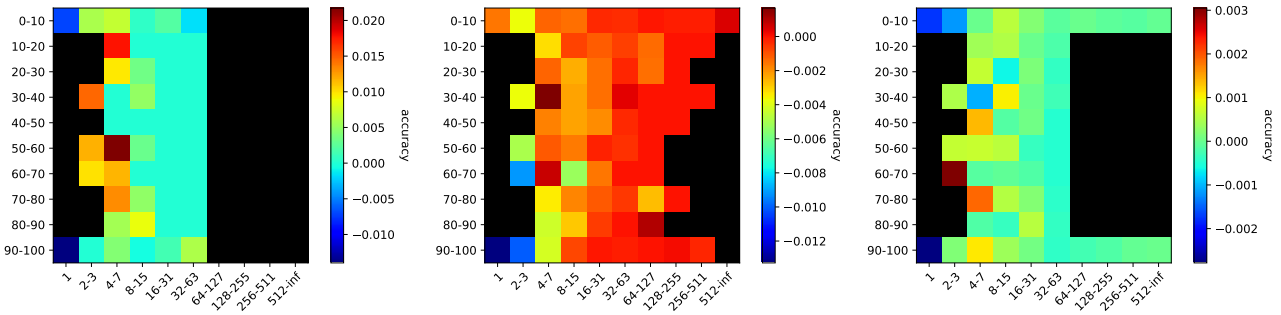


Figure 7.2: This heatmaps show for three different datasets (sonar, spambase, telescope) the difference (Prior$_{1,1}$ minus probability) between the probability and the Prior$_{1,1}$ predictions. The heatmaps describe how often leafs are part of the correct prediction.

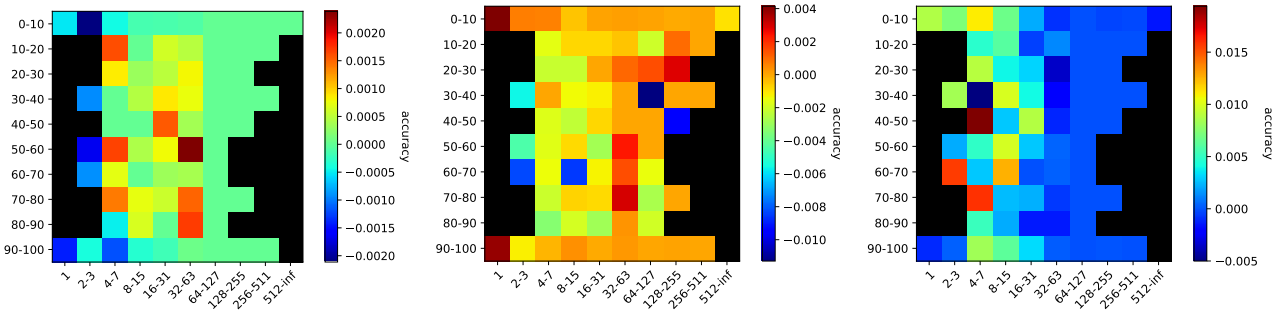


Figure 7.3: The heatmaps here show the difference of the Epistemic-Weight prediction minus the Plausibility prediction.

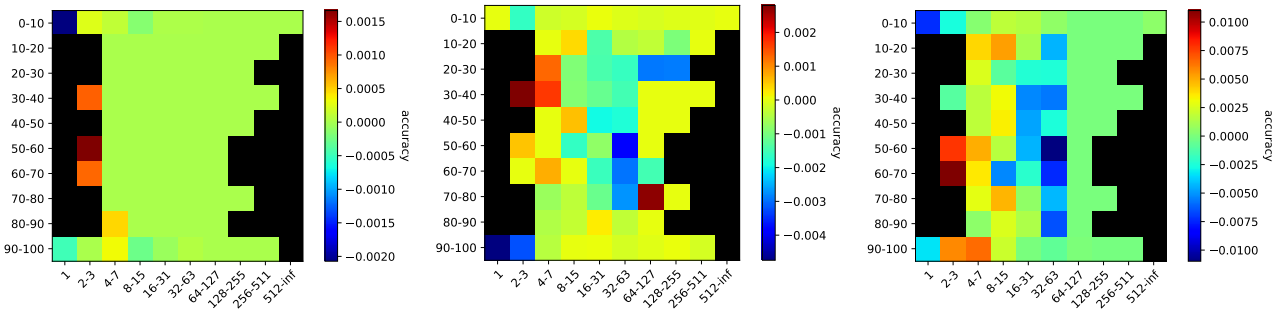


Figure 7.4: The heatmaps here show the difference of the Plausibility minus the probability prediction.

8 Conclusion and Future Work

This final section will give a summary of the results presented in this thesis and information about possible future work which can be done in this area.
Dealing with uncertainty is a hard task in machine learning. Variance in the dataset and bias in the model can make it so that it can be very difficult to find the correct model and also make the right decision. This thesis considered decision trees and random forests, particularly the leafs used for the respective predictions. It was argued that a small number of instances per leaf might not be an appropriate basis for a prediction, particularly when a probability of 100% or 0% is predicted for small, pure leafs. A number of different prediction methods were introduced using the number of instances in a leaf to return a prediction value which not only gives insight about the preference of a class based on the class ratio, but also based on the uncertainty.
An experiment conducted on single, not random decision trees confirmed that such methods achieve better results when ranking the scores (i. e. the confidence) of the predictions made using that tree. Thereby, the $\text{Prior}_{1,1}$ prediction which only changes the way predictions are made slightly (compared to the probability), gave the best ranking results. This falls in line with Provost and Domingos work which also stated that this method (there called the *Laplace correction*) gives considerably better results for probability estimation. Nevertheless, other prediction methods proposed in this thesis also gave better ranking than the probability but they overall stayed behind the $\text{Prior}_{1,1}$ approach.
Considering an even wider set of prediction and aggregation methods and combinations, another experiment was conducted which tested the performance of such methods on random forests. The results given here were surprising considering the decision tree experiment. While larger trees (i. e. trees with smaller leafs) on average gave better results than smaller trees, we could still observe than no other of the proposed methods was able to achieve significantly better results than the probability. Particularly surprising was that the $\text{Prior}_{1,1}$ prediction, which achieved the top performance on decision trees, ranked below the probability. We reasoned that the uncertainty resulting from a single prediction is noticeably less relevant in random forests because the randomness and the ensemble learning applied by random forests make the random forest compensate mistakes of the single decision trees very well. Moreover, further analysis indicated that the predictive information in small leafs (even in leafs with only one instance) more often helps the aggregated prediction of the random forest ensemble than it hurts the prediction because of its uncertainty. Consequently, the weakness of decision trees, that uncertainty resulting from low instance counts worsens the predictive power of the model, is not nearly as present in random forests as it is in single decision trees. Also, this behavior of random forests support the statements in the paper by Wyner et al. where they claimed that random forest are not prone to overfitting. For that reason, the probability gives great results in random forests, even when small leafs are used.
However, some other insights could be given by the experiment. Aggregation methods which only forward one prediction perform worse for a random forest aggregation than predictions which give positive weight to every decision trees in the classifier. Methods should not focus too much on one single prediction only for the reason that this prediction is based on a large leaf because if the prediction of one such leaf is false, then it is difficult for the ensemble to outweigh that decision. One prediction method which achieved remarkable performance, even on random forests, was the *Plausibility* approach. This approach (Hüllermeier, unpublished), using aspects of belief-functions (Shafer, 1976), probability, and aleatoric and epistemic uncertainty managed to perform comparably to the probability on larger trees, event though it uses a greatly different way of reaching prediction values.
All in all, my recommendation for using prediction methods on random forest is sticking to probability as no other methods achieved significantly better performance and because calculation of probability is very fast and efficient which is not given for every other method. However, when small improvements for the ensemble are desired, it might be worth trying out different approaches as done in this thesis.

Nevertheless, keep in mind that small improvements could also be a results of randomness and not of actual superior performance of the respective prediction and aggregation methods.
We did not get significant improvements on random forest prediction but that does not mean that this is not possible. Maybe building trees differently could give better results for particular methods or it at least seems to be worth trying out. For example, you could use a certain measure, such as a threshold based on epistemic uncertainty, to stop the trees from growing instead of using a number of minimum instances. However, I do not expect that last suggestion to achieve significantly improved results because very large trees were shown to perform better on average.
Furthermore, you can think of many more prediction or aggregation methods similar or entirely different to those proposed in this thesis. While they did not outperform the probability in the experiment here, it is possible that there are applications on different machine learning tasks where such measures considering uncertainty are more beneficial than in a random forest environment. That such methods can give improvements in the estimation of the certainty of predictions was successfully shown in the decision tree experiment.

Bibliography

The collection of arff datasets of the connectionist artificial intelligence laboratory (liac). `https://github.com/renatopp/arff-datasets`. Accessed: 2019-01-29.

Milton Abramowitz and Irene A Stegun. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*, volume 55. Courier Corporation, 1965.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

Rajen Bhatt and Abhinav Dhall. Skin Segmentation Dataset. UCI Machine Learning Repository.

Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Davide Ceolin, Willem Robert Van Hage, Wan Fokkink, and Guus Schreiber. Estimating uncertainty of categorical web data. In *URSW*, pages 15–26, 2011.

Valdigleis S Costa, Antonio Diego S Farias, Benjamin Bedregal, Regivan HN Santiago, and Anne Magaly de P Canuto. Combining multiple algorithms in classifier ensembles using generalized mixture functions. *Neurocomputing*, 313:402–414, 2018.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Thierry Denœux. The cautious rule of combination for belief functions and some extensions. In *2006 9th International Conference on Information Fusion*, pages 1–8. IEEE, 2006.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Marco F Duarte and Yu Hen Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, 2004.

Antonio Diego Silva Farias, Regivan HN Santiago, and Benjamín Bedregal. Some properties of generalized mixture functions. In *IEEE International Conference on Fuzzy Systems*, pages 288–293. IEEE, 2016.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society for Artificial Intelligence*, 14(771-780):1612, 1999.

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

Bhumika Gupta, Aditya Rawat, Akshay Jain, Arpit Arora, and Naresh Dhami. Analysis of various decision tree algorithms for classification in data mining. *International Journal of Computer Applications*, 163 (8):15–19, 2017.

Eyke Hüllermeier. Aleatoric and epistemic uncertainty in machine learning. Paderborn University, Department of Computer Science, unpublished.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

DD Lucas, R Klein, J Tannahill, D Ivanova, S Brandon, D Domyancic, and Y Zhang. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4): 1157–1171, 2013.

Kamel Mansouri, Tine Ringsted, Davide Ballabio, Roberto Todeschini, and Viviana Consonni. Quantitative structure–activity relationship models for ready biodegradability of chemicals. *Journal of Chemical Information and Modeling*, 53(4):867–878, 2013.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

Martin Možina, Janez Demšar, Jure Žabkar, and Ivan Bratko. Why is rule learning optimistic and how to correct it. In *European Conference on Machine Learning*, pages 330–340. Springer, 2006.

Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

Peter Nemenyi. *Distribution-Free Multiple Comparison*. Princeton University, 1963.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Moacir P Ponti Jr. Combining classifiers: from the creation of ensembles to the decision fusion. In *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*, pages 1–10. IEEE, 2011.

Foster Provost and Pedro Domingos. Well-trained pets: Improving probability estimation trees. 2000.

Benjamin Quost, Marie-Hélène Masson, and Thierry Denœux. Classifier fusion in the dempster–shafer framework using optimized t-norm based combination rules. *International Journal of Approximate Reasoning*, 52(3):353–374, 2011.

Glenn Shafer. *A Mathematical Theory of Evidence*, volume 42. Princeton university press, 1976.

Philippe Smets. Belief functions: the disjunctive rule of combination and the generalized bayesian theorem. *International Journal of Approximate Reasoning*, 9(1):1–35, 1993.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1): 1558–1590, 2017.

I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, 36(3):5866–5871, 2009.

Zhi-Qiang Zeng, Hong-Bin Yu, Hua-Rong Xu, Yan-Qi Xie, and Ji Gao. Fast training support vector machines using parallel sequential minimal optimization. In *2008 3rd International Conference on Intelligent System and Knowledge Engineering*, volume 1, pages 997–1001. IEEE, 2008.