TU Dortmund, Faculty of Statistics

Bachelor Thesis

Recursive Partitioning in Classification: Comparing Random Forests of CART and CIT

Patrick Köhler

September 19, 2018

Advisor: Dr. Uwe Ligges

Contents

1 Introduction **1**

2 Motivating Random Forests in Classification **3**

2.1 Recursive Partitioning in Classification 3

2.2 Classification Trees 5

2.3 Ensembles and Random Forests 6

3 Scope of this Thesis **10**

4 Statistical Concepts and Algorithms **12**

4.1 Statistical Concepts 12

4.1.1 Statistical Tests 12

4.1.2 Multiple Hypothesis Testing by Šidák Adjustment 14

4.1.3 Bayes Partition 15

4.1.4 Generalization and Overfitting 16

4.2 Algorithms 16

4.2.1 CART 16

4.2.2 Conditional Inferences Trees 18

4.2.3 Random Forests 21

5 Exploratory 2D Simulation in R **28**

5.1 Experimental Design 28

5.2 Analysis of the Results 30

5.3 Code Documentation and Computational Remarks 35

6 Summary, Conclusion and Outlook **37**

References **39**

Appendix **41**

1 Introduction

Classification, also known as supervised learning with discrete labels, is a central issue in applied statistics. Many real world problems, such as early cancer detection (Murty and Babu, 2017) and risk factor analysis for road safety (Kwona et al., 2015), can be approached with classifcation algorithms. Approaches from different theoretical perspectives lead to various kinds of classifiers.
Random forests are ensembles (i.e. aggregations) of individual classifiers that can be understood as tree structured graphs (Breiman, 2001). Each tree is fitted to a randomly drawn subset of the available data set. Additional randomness is injected into the algorithm at each split: only *mtry* randomly selected variables are considered for splitting. Even though various tree structured classification algorithms have been proposed (cf. Quinlan, 1993, Sonquist and Morgan, 1964 and Hothorn et al., 2006), random forests are widely used in their original form, i.e. with the CART algorithm (Breiman et al., 1984) for each tree. In contrast to many traditional statistical models, random forests have the potential to solve AI-level tasks, as they can discriminate among a number of regions exponential to the number of trees (Bengio, 2009).

This thesis explores the recursive partitioning principle that is implemented by random forests. Two algorithms for constructing the individual trees, as well as some theoretical foundations, are described. Central theoretical results concerning the random forest algorithm are explained. One states that random forests do not overfit without boundary as the number of trees increases (Breiman, 2001). Furthermore an upper bound for the generalization error, depending only on the predictive accuracies and the correlations of the trees, can be derived (Breiman, 2001). Additionally it is argued that variable selection biases of individual tree algorithms can be mitigated in random forests by using small values for the *mtry* parameter.

The traditional CART algorithm is compared to an alternative tree classifier which is built upon the well studied field of statistical permutation tests (Hothorn et al., 2006). In particular, the so called conditional inference trees (CIT) overcome a variable selection bias of CART (Strobl et al., 2007, Hothorn et al., 2006).
The effects of using CITs instead of CARTs as base learners in random forests are studied in a small exploratory simulation conducted in `R` (R Core Team, 2017). For a constructed set of 90 two dimensional scenarios both forests are fit to 100 data sets with grid-search optimized hyper parameters. The prediction accuracies (estimated with test sets) and the resulting optimal hyper parameters are analyzed.
Even though the maximal difference in accuracy is estimated to be below 5%, CART forests yield higher prediction accuracies than CIT forests in most of the 90 scenarios. Visualizing the scenario with the maximal difference shows that there exist problems for which CIT forests can incorporate more information about the underlying distribution. However, this

does not lead to higher prediction accuracies in this specific case.
In contrast to Breiman's (2001) initial recommendation of always growing trees to their maximal size, the simulation shows that there exist scenarios in which forests with smaller trees yield higher prediction accuracies (for both base learners).

This introduction is followed by a comprehensive motivation of the random forest algorithm in section 2. The classification problem is put into a probabilistic context and it is described how forests are constructed from individual trees. As all relevant terms have been introduced by then, section 3 sets the scope of this thesis.
section 4 contains detailed explanations of the theoretical framework for this work. Explanations regarding the fundamental statistical concept of (multiple) hypothesis testing enable readers without statistical backgrounds to understand the conditional inference trees. The three algorithms themselves (CART, CIT, random forest) are explained in the second subsection. Relevant results regarding the algorithms are also discussed in the same subsection.
section 5 covers detailed information about the simulation study. After the concrete experimental design is outlined, the analysis of the results is performed. Computational remarks in the end of the section make the results easily reproducible. The full source code of all files can be found in the appendix.
Finally, a separate section summarizes the thesis, highlights central results once again and gives a brief outlook on future research to be conducted on the field of random forests.

2 Motivating Random Forests in Classification

In this section an overview of the context in which random forests appear is given. At first the general classification problem is defined and basic principles of statistical learning are put into its context. This allows to motivate the approach of recursive partitioning leading to two different tree structured algorithms: Classification and Regression Tree (CART) and Conditional Inference Tree (CIT). After defining trees as an algorithmic tool for classification, a separate subsection deals with learning ensembles and emphasizes the concept of random forests.

2.1 Recursive Partitioning in Classification

This section introduces the classification problem and gives a rationale for the recursive partitioning approach in accordance with Hastie et al. (2009).

The Classification Problem

Most machine learning algorithms can be divided into two categories: Unsupervised and supervised learning. Whereas unsupervised learning algorithms reveal general structures in unlabeled data, supervised algorithms aim to learn an association between input variables and a respective label. Supervised learning is further divided into regression and classification problems. While the label is continuous in regression problems, the set of possible values for the label is discrete and finite in classification.
Fitting the supervised models generally aims for high predictive performance. Different approaches to estimating the performance of an algorithm are possible and will be motivated here briefly. All these approaches require the choice of a specific loss function. This function measures the correctness of the predictions and depends on the true as well as the predicted labels for given data points. While various appropriate loss functions can be constructed for regression problems (e.g. Mean Square Error, Hinge Loss), many classification algorithms only distinguish between correct and incorrect predictions. Depending on the a priori knowledge of the problem, wrong classifications can be weighted differently in order to obtain a more appropriate loss function.
Considering the observations and labels as random variables, maximizing predictive performance is equivalent to minimizing the expected loss, given the data.
The most intuitive approach divides the data set into so called training and test sets. After specifying the size of the two sets, observations are drawn uniformly random from the original data set. By only considering the (larger) training set to fit the model, predicting the values of the test set yields a loss estimate. The model constructed via the train and test method can not incorporate all information available in the original data set.

Resampling techniques estimate the prediction accuracy by fitting several models to different subsets of the original data and average a respective loss estimate over all fitted models. In k-fold cross validation (CV) the data set is divided (randomly) into k groups of approximately equal size. Now the train and test method is applied k times, where each time one different group is defined as the test set for each model. The mean of all k loss estimates yields the total loss estimate. The choice of k depends on the number of observations n in the data set and the computational cost of fitting the model.
Bootstrap estimates draw B new data sets, each of size n, with replacement from all observations and fit a model to all B sets. For each new data set the respective test set consists of all observations not included in the training set. The expected loss, given the data, is also estimated by computing the mean of all B Bootstrap samples. This approach has very high computational cost but yields an estimate for the whole distribution of estimated loss values.

The Probabilistic Perspective

The probabilistic perspective on the classification problem allows to perform statistically rigorous analyses. Consider the available data set

$$\{(y_i, x_{i1}, \dots, x_{ip})| \quad i = 1, \dots, n\} =: \mathcal{L},$$

which will also be referred to as the learning sample in the following, as observations of respective independent and identically distributed random vectors

$$(Y, X_1, \dots, X_p) \sim \mathcal{P},$$

with some distribution $\mathcal{P}$. If specific aspects of $\mathcal{P}$ are known, an optimal classifier can be derived which will be explained in section 4.1.3. The probability that the label Y is equal to one class j is maximized over j, given that the input variables had already been observed. It results some “true” relationship f between the p-dimensional space of input variables $\mathcal{X}$ and the discrete label:

$$f: \quad \mathcal{X} \to \{1, \dots, k\}.$$

Many algorithms aim to approximate f by an estimated model $\hat{f}$ with the knowledge of the learning sample.

Recursive Partitioning

The true function f can be understood as a partition of the variable space $\mathcal{X}$. Define the subspace of points which are associated with the k-th class:

$$C_i := \{x \in \mathcal{X} : f(x) = i\}, \qquad i = 1, \dots, k$$

and write $\mathcal{X}$ as an union of the k classes:

$$\bigcup_{i=1}^{k} C_i = \mathcal{X}.$$

Therefore $\hat{f}$ has to divide $\mathcal{X}$ into $r \geq k$ disjoint subsets which each assign its points exactly one class. Because the set of possible partitions is extremely large and contains infinitely complex configurations, an algorithmic approach has to limit its possible space of estimated functions, denoted by $\hat{\mathcal{F}}$ in the following.
Recursive partitioning algorithms divide $\mathcal{X}$ initially into two subsets. Each resulting subset is then divided recursively until a stop criterion is satisfied. By doing so $\hat{\mathcal{F}}$ is restricted to the algorithm's form of dividing a given subset. A computationally easy dividing scheme is univariate binary splitting. At each step of the partitioning process one variable is selected to find its optimal splitting threshold.

2.2 Classification Trees

The commonality of all recursive partitioning approaches to classification is their representation as trees. A tree is defined as an undirected graph where any two vertices are connected by exactly one path (Turau and Weyer, 2015). A classification tree expects a data point from the variable space in which it was constructed, and assigns it one of its leaf nodes. Figure 1 illustrates a tree with three leafs. By evaluating the logical operations A, B, C and D, observations are passed through the tree until one leaf node is reached (Strobl et al., 2010).
Considering $\mathcal{L}$ as a random sample of observations, $\hat{f}$ should find k subspaces $\hat{C}_i$, for which the probability of the respective points belonging to class i is as high as possible (Hastie et al., 2009). Accordingly, in order to reach high prediction accuracy, the general structure in the data has to be incorporated into the algorithm, without adopting too much of its random variation. Consequently, the splitting and stop criteria have to be chosen appropriately.
The CART (Classification and Regression Tree) algorithm splits the given subset of $\mathcal{X}$ in order to maximize the purity of the resulting daughter nodes (Breiman et al., 1984). A stop criterion is chosen heuristically. Possible options are splitting until all leaf nodes are pure (i.e. only contain observations of one class) or splitting until a certain threshhold

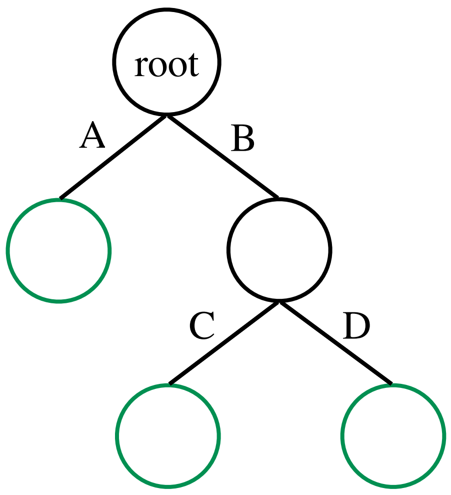

Figure 1: A tree structured graph with three leafs.

of impurity reduction can not be reached anymore (Breiman et al., 1984). In contrast, conditional inference trees (CIT) perform statistical permutation tests for splitting. The splitting and stop criteria are defined through a non-parametric test statistic at a given significance level α (Hothorn et al., 2006).

2.3 Ensembles and Random Forests

In supervised learning several algorithms approach the prediction task by fitting different models and combining their individual results to one prediction. Such methods are referred to as ensembles (Hastie et al., 2009).
Dietterich (2000) came up with three conceptual reasons why ensembles can outperform single classifiers, which are illustrated in Figure 2. His first argument is statistical. In some settings with little data and a large space of possible relations $\hat{\mathcal{F}}$ multiple classifiers with equal performance but different biases can be found. Combining these models by averaging their decisions also averages their biases, so that the probability of constructing a one-sided biased classifier is reduced. Furthermore Dietterich argues from a computational perspective: In situations with sufficient amounts of data, finding global optima of the loss function may not be possible in all scenarios. An average of multiple classifiers which got stuck in local optima of the loss function may provide a better approximation of f than single classifiers in local optima. At last a representational justification for ensembles is given. While one algorithm is restricted to its space of possible models $\hat{\mathcal{F}}$, an ensemble can construct additional types of functions which do not lie inside of $\hat{\mathcal{F}}$.
Bengio (2009) illustrates a major advantage of tree ensembles over individual trees in Figure 3. The number of distinguishable regions for an individual tree is linear in the number of its leafs (each leaf defines one region). Ensembles, on the other hand, can distinguish between

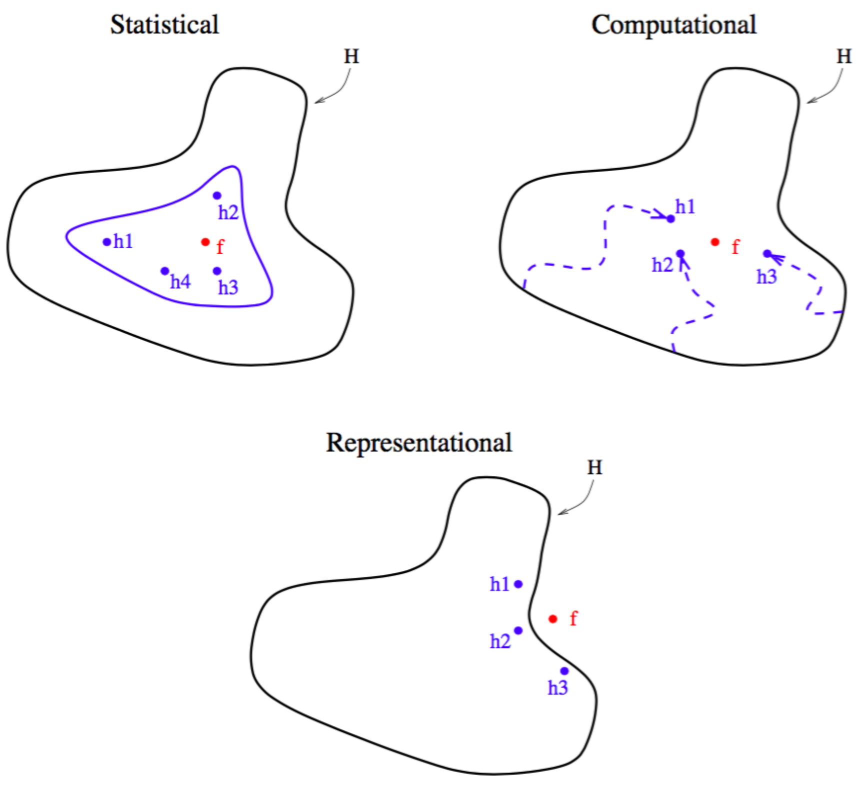

Figure 2: Three conceptual rationales for ensembles. Taken from Dietterich (2000).

a number of regions exponential to the number of trees as long as the number of trees does not exceed the number of variables. Each new tree can separate many individual regions into two new regions. Hence, tree ensembles may be suitable for fitting highly non-linear partitions.
After introducing bagging, a fundamental paradigm for ensembles, the concept of random forests is finally motivated.

Bagging

Bagging stands for bootstrap aggregation and was initially proposed by Breiman (1996). The resampling method bootstrap (introduced in section 2.1) is used to average different models in order to obtain a single, more accurate model (Hastie et al., 2009). For each of the B bootstrap samples $\mathcal{L}_i$ a model is fit. For practical reasons it is assumed here that the same algorithm is applied to all $\mathcal{L}_i$, however the bagging principle itself has no such restriction. In order to predict the class for a given input, all B models have to be evaluated. Afterwards each model casts a vote for one class and the class with the most votes is predicted. This procedure can reduce the variance of the classifier (Hastie et al., 2009) and can be fully computed in parallel.
Assuming a stable classifier, the predictions do not change very much when $\mathcal{L}$ is varied slightly. Therefore, the aggregated predictions of many bootstrap samples will not vary

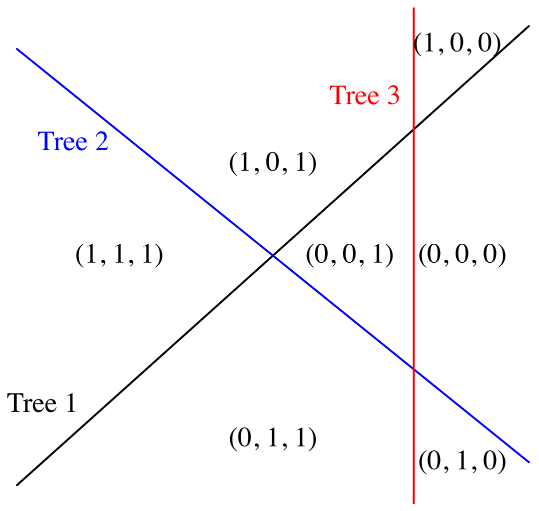

Figure 3: Partition of $\mathbb{R}^2$ induced by a tree ensemble.
The brackets contain the decisions of all trees for each region, where a tree decides between 0 (right, from an upward facing perspective) and 1 (left)

much from the predictions produced by a single model of the same class fit to $\mathcal{L}$. According to Breiman (1996) bagging improves the predictive performance of algorithms for which small changes in $\mathcal{L}$ result in large changes for the predictions. Furthermore, it can be shown by means of rigorous statistical analysis that strong classifiers can be improved by bagging, whereas this may not be possible for weak classifiers (Breiman, 1996).

Combining Trees to Random Forests

Random forests are bagging ensembles of trees with an additional random component. *ntree* classification trees are fit to one bootstrap sample of $\mathcal{L}$ each. Additionally, at each split, only $mtry \leq p$ randomly chosen variables are considered for splitting. Predicting the unknown label for an observation requires the evaluation of all trees. Every tree casts a vote for one class and ultimately the class with the most votes is predicted.
The additional randomization of the tree construction assures that even variables with a low contribution to the class separation take part in the decision process (Breiman, 2001). Despite its non-parametric nature, random forests can compute internal estimates for variable importance and certainty of a specific classification (Breiman, 2001). Although the straight forward interpretation of classification trees gets lost in forests, the space of estimated functions $\hat{\mathcal{F}}$ allows for a high variable interaction (Strobl et al., 2010). The resulting partitions are much smoother now, as they can be understood as the average of the single tree partitions (Strobl et al., 2010).

Intuitively speaking, an optimal ensemble consists of classifiers that predict as accurately as possible while disagreeing as much as possible (Dietterich, 2000). Trees can adapt very well to the data points in the learning sample, if the parameters are chosen appropriately. If they adopt the random variation in the respective $\mathcal{L}_i$ while reflecting the general structure of the data, a diverse forest can be grown.
Random selection of splitting variables causes individual trees to not perform optimally. However, it assures low correlations between the trees. Using the Strong Law of Large Numbers, Breiman (2001) derives an upper bound for misclassification. This theorem reveals the tradeoff between the correlation in between the trees and their indivudal strengths in a formal manner. It will be discussed in detail in section 4.2.3.

3 Scope of this Thesis

As this work is rather basic research than an approach to a specific yes/no question, its scope is quite widely spread. The random forest mechanism is explored in several directions with the focus on assessing the effects of two conceptually different base learners. All analyses revolve around the question whether the random forest algorithm can profit from a tree algorithm that is based on a rigorous statistical foundation. The theory section emphasizes on motivating all important aspects of the algorithms sufficiently. Motivations and general intuitions are elaborated with rigorous mathematical formulations. However, not all concepts can be discussed in exhaustive detail, as the work focuses on the random forest concept itself.

Statistical tests, for example, are only motivated briefly and formalized without further illustrations. Readers that are new to hypothesis testing are referred to standard statistical literature such as Fahrmeir et al. (2007) and Hartung et al. (2009) for an in depth motivation of this concept.

The core theory section 4.2 does not give an exhaustive description of all major concepts revolving around the three algorithms as this would shift the focus away from their commonalities and differences. Only those concepts which are necessary to understand and compare random forests of CART and CIT, as proposed by their authors, are covered. In particular, extensions such as optimal pruning strategies for CART or minor theoretical results such as conditions for consistency of individual trees are not discussed as they play no role for the comparison of both forests.

Regardless, it is necessary to mention that CITs can be applied to all kinds of supervised learning problems, not only classification. They can be modified easily in several ways, e.g. other test statistics within the class of linear statistics can be chosen. This is indicated by using the highly general notation from the original paper, but not discussed further. Moreover different methods of computing the $p-$values are implemented.

The subsection on the random forest algorithm is restricted to one specific algorithm proposed in Breiman (2001). While this seems to be the most popular one, other kinds of randomized tree ensembles are also discussed in Breiman (2001). The remarks on the upper bound of generalization error do not reflect Breiman's proof to its full extent. They should be seen as an introduction to the formal concepts which were used in the proof.

Simulations can always cover only a finite subset of the infinite space of possible scenarios. The constructed framework for the simulation is two dimensional with two classes in order to visualize problems easily. The classification problems themselves are constructed to be relatively easy to solve so that the interpretations are not confused by the complexity of the problems.

Computational resources restrict the extent and precision of the simulation. It was chosen to only use one standard laptop with two physical cores and a maximum computation time of one night per simulation run.

4 Statistical Concepts and Algorithms

The intuitive introductions to random forests of CARTs and CITs are formalized in this section. Before the algorithms are explained in detail, necessary statistical concepts such as (multiple) hypothesis testing, Bayes classifiers and generalization errors are discussed.

4.1 Statistical Concepts

4.1.1 Statistical Tests

This paragraph defines the concept of a statistical hypothesis test. At first, an intuition for statistical testing is given. Afterwards the concept of significance is formalized by defining the $p-$value. Furthermore the class of permutation tests is discussed briefly in order to understand the tests that are performed by conditional inference trees.

The introduction to statistical testing is explained in accordance with Hartung et al. (2009, pp. 133ff.) and Fahrmeir et al. (2007, pp. 411ff.). A statistical hypothesis test solves a decision problem. A so called null hypothesis (H_0) is formulated and can either be rejected or kept. H_0 and its logical complement, the so called alternative hypothesis (H_1) are constructed to be mutually exclusive. Either H_0 or H_1 is true, but it can never be the case that both or neither are true.
If a decision was made to reject or to keep H_0, there is always the possibility that the decision was wrong. If H_0 is rejected despite being true, this wrong decision is called a type I error. Making a type II error means to keep H_0 even though H_1 is true. A statistical hypothesis test to the level α is a function of the observed variables, which either rejects or keeps the null hypothesis while the probability to make a type I error does not exceed α.
Let $X_1, \dots, X_n$ be random variables with domain D. The assumption of identical and especially independent distributions in between all X_i is often made. Denote the respective observations by $x_1, \dots, x_n$ and define the decision function

$$\phi : \quad D^n \to \{0,1\}, \qquad (x_1, \dots, x_n) \mapsto \phi(x_1, \dots, x_n).$$

The definition of a statistical test requires

$$\mathbb{P}_{H_0} := \mathbb{P}(\phi = 1 | H_0 \quad \text{true}) \leq \alpha. \tag{1}$$

In order to measure the extent to which the observed variables disagree with the null hypothesis, every test uses a specific test statistic

$$T : \quad D^n \to \mathbb{R}, \qquad (X_1, \dots, X_n) \mapsto T(X_1, \dots, X_n).$$

Their individual forms and rationales are specific to the H_0, mostly of heuristic nature. It is important to note that there may exist several (reasonable) tests for the same problem, as different test statistics may be suitable for the same problem. To suffice condition (1) the probability distribution of T, denoted by $\mathcal{T}$, has to be derived under the assumption that H_0 is true.

A large class of tests make assumptions about the distributions of $X_1, \ldots, X_n$ and derive $\mathcal{T}$. Having $\mathcal{T}$ at hand, the probability that T is equal to its observation t or even speaks more against H_0, can be computed. This probability is referred to as the $p-$value of the respective test scenario and can be used to formulate the decision function ϕ:

$$\phi(x_, \ldots, x_n) = \begin{cases} 1, & p\text{-value} \leq \alpha \\ 0, & p\text{-value} > \alpha. \end{cases}$$

In circumstances where the variables' distributions are not known, the distribution of T under H_0 can be calculated or approximated by conditioning on the set of all possible permutations of the observed variables. This procedure is referred to as a permutation test and will be discussed for independence tests in accordance with Neuhaeuser (2011) briefly in the following. Consider the test problem:

$$H_0 : X \perp Y \qquad \text{vs.} \qquad H_1 : X \not\perp Y,$$

where X could be one covariate and Y the label of a multidimensional classification problem. Without loss of generality for the definition of this concept, assume that X is discrete with possible values $h_1, \ldots, h_k$. Denote the n observed data points in two separate n-dimensional vectors w and v. w contains all observed values for X and v at each index the respective observed value for Y. If H_0 were true, all permutations of v would have equal probabilities. Capture all these permutations in the set $\mathcal{S}$ and compute the test statistic for all its elements. The $p-$value can then be calculated as the fraction of elements in $\mathcal{S}$ which yields a test statistic equal to or greater than t.

For large n this approach is computationally extremely intensive, as the number of possible permutations is $n!$. Even though some permutations will yield the same test statistic in practical scenarios, for large n the $p-$value needs to be approximated in a simulation. In this case a prespecified number of uniformly random chosen permutations (without replacement) is treated as $\mathcal{S}$.

However, recent theoretical developments in the field of permutation tests allow to approximate the $p-$value computationally highly efficient for a class of linear statistics (Strasser and Weber, 1999). They derive the distribution of a very generally formulated test statistic under the assumption of H_0, conditioned on the respective $\mathcal{S}$. These distribution results are used to implement a stop criterion for conditional inference trees which will be explained in section 4.2.2.

4.1.2 Multiple Hypothesis Testing by Šidák Adjustment

In the following, a crucial problem that arises when multiple hypotheses are tested at the same time is discussed. Afterwards a commonly used approach known as Šidák adjustment, a refinement of the popular Bonferroni adjustment, is explained.
Consider a testing problem with k individual hypotheses $H_0^1, \dots, H_0^k$. The global null hypothesis is defined as

$$H_0^{\text{glo}} : \bigcap_{i=1}^{k} H_0^i.$$

In order to test the global null hypothesis, the definition of a type I error has to be extended. Besides the probabilities for individual falsely rejected hypotheses, probabilities for multiple falsely rejected hypotheses have to be considered. Especially interesting are the cases in which at least one hypothesis is falsely rejected. In practice one would like to control the probability that at least one hypothesis is falsely rejected for any possible combination of true and false individual hypotheses. Some authors refer to this as strong control of the familywise error and others call it control of the multiple level (Shaffer, 1995).
Denote the decision function for the i-th hypothesis pair by ϕ^i and the respective $p-$value by p_i. If the individual hypotheses can be assumed to be independent, the following function defines a test to the multiple level of α:

$$\phi^i(x) = \begin{cases} 1, & \text{if} \quad p_i \leq 1 - (1-\alpha)^{\frac{1}{k}} \\ 0, & \text{else} \end{cases} \qquad \forall i = 1, \dots, k$$

(Hochberg and Tamhane, 1987, section 1).
The underlying idea of this approach is easily accessible. Let $\tilde{\alpha}$ be the individual level of all ϕ^i and α the desired multiple significance level. Calculate the probability that at least one null hypothesis is falsely rejected:

$$\begin{aligned}
&\mathbb{P}(\exists i : \phi^i = 1 | H_0^i \quad \text{true}) = 1 - \mathbb{P}(\forall i : \phi^i = 0 | H_0^i \quad \text{true}) \\
&=^{\text{indep.}} 1 - \prod_{i=1}^{k} \mathbb{P}(\phi^i = 0 | H_0^i \quad \text{true}) = 1 - \prod_{i=1}^{k} 1 - \mathbb{P}(\phi^i = 1 | H_0^i \quad \text{true}) \\
&\leq 1 - (1-\tilde{\alpha})^k.
\end{aligned}$$

Now, calculate $\tilde{\alpha}$ so that this probability equals α.

$$\begin{aligned}
&\alpha = 1 - (1-\tilde{\alpha})^k \iff 1 - \alpha = (1-\tilde{\alpha})^k \iff (1-\alpha)^{\frac{1}{k}} = 1 - \tilde{\alpha} \\
&\iff \tilde{\alpha} = 1 - (1-\alpha)^{\frac{1}{k}}.
\end{aligned}$$

While early implementations of CITs used the popular Bonferroni adjustment, this method is used in the `R` (R Core Team, 2017) package `party` (Hothorn et al., 2018) from version 0.8-4 on. Global tests of independence between k covariates and the label in conditional inference trees are performed.

4.1.3 Bayes Partition

In the following paragraph a theoretically optimal classifier is derived for the special scenario of two classes. It requires the knowledge of the class distributions and the a priori probabilities (i.e. the probabilities for one observation belonging to a specific class), which is not the case in most practical scenarios. However, it will serve as a reference for the two dimensional simulation, as the necessary information is available by the experimental design. A more general explanation of this concept can be found in Hastie et al. (2009) in Chapter 2.
In the following, equal misclassification costs are assumed. This means that classifying an object from class 1 as class 2 and the complementary misclassification are treated as equally undesirable. A Bayes decision rule chooses the class with the highest a posteriori probability, i.e. $\mathbb{P}(Y = k|X = x)$, for any given observation x. Rewrite this by applying the Bayes rule:

$$\mathbb{P}(Y = k|X = x) = \frac{\pi_k \mathbb{P}(X = x|Y = k)}{\sum_{j=1}^{k} \pi_j \mathbb{P}(X = x|Y = j)}.$$

As the denominator is constant for all classes, the a posteriori probability is proportional to

$$\pi_k \mathbb{P}(X = x|Y = k),$$

whereas the second multiplier is the density of class k evaluated at x. For reasons of clarity, this will be denoted in the more conventional form as $f_k(x)$.
Consequently the Bayes decision rules can be written as

$$h^{\text{Bayes}}(x) = \begin{cases} 1, & \pi_1 f_1(x) \geq \pi_2 f_2(x) \\ 2, & \text{else} \end{cases} .$$

Note that $\geq$ can be arbitrarily replaced by $>$ if X is continuous.
This classifier implicitly defines the bayesoptimal partition by evaluating h^{Bayes} on the full domain of X.

4.1.4 Generalization and Overfitting

Two termini related to predictive performance are defined in this paragraph, as they play a central role throughout the thesis. The principle goal in supervised learning is to reach high predictive performance on new data points. Algorithms should extract the general structure in the data without adopting too much of the random variation. If an algorithm performs well on new data points one says that it generalizes well (Hastie et al., 2009). It takes the information which it was fed and extrapolates the underlying structure to parts of the variable space which are not covered by the learning sample.
Overfitting can be seen as the contrary of generalization. It happens when the algorithm adopts too much of the random variation in the data, without generalizing the structure to unobserved points in the variable space. Typically this happens when a model allows for too many parameters. The resulting function may fit perfectly to the learning data, but has low predictive performance for future observations (Hastie et al., 2009).

4.2 Algorithms

Now the three algorithms CART, CIT and random forest are explained in detail. The CART subsection includes some remarks on the general construction of classification trees.

4.2.1 CART

This section summarizes the key aspects of Breiman et al. (1984) for basic classification trees. Generally speaking, the construction of a tree can be traced back to three elements: At first, it has to be defined how splits are selected. Next, a stop criterion has to be chosen. This requires a condition under which nodes are not splitted further and are declared as terminal nodes. At last, each terminal node has to be assigned to exactly one class. CART's implementations of these elements are described in the following one by one.
CART only considers binary splits of the form $x_k \leq a$ or $x_k = a$, where x_k denotes the observation of a given covariate X_k and a denotes a possible value of X_k. For a given data set, all possible splits of all covariates are evaluated by a goodness of fit function. The split yielding the highest goodness of fit is chosen and defines the subsets of observations for the daughter nodes.
As CART aims for as "pure" as possible terminal nodes, the goodness of fit is measured by the reduction of impurity. A completely pure node only contains observations of one class. The formal impurity measure at node t is defined as

$$i(t) := \psi\left(p(1|t), \dots, p(k|t)\right),$$

where ψ denotes an impurity function and $p(j|t)$ estimates the probability of an observation in node t belonging to class j.
Breiman et al. (1984) define ψ as an function of a k-tuple of numbers $(p_1, \ldots, p_k)$, satisfying

$$p_j \geq 0 \quad \forall j \in \{1, \ldots, k\} \qquad \wedge \qquad \sum_{i=1}^{k} p_i = 1.$$

They further restrict the form of an impurity function by three intuitive conditions:

1. Impurity is only maximal for equal class probabilities, i.e.:
   $\text{argmax}_x \psi(x) = (\frac{1}{k}, \ldots, \frac{1}{k})$
2. Impurity is only minimal when only one class is represented, i.e.:
   $\text{argmin}_x \psi(x) = e_j, \quad j = 1, \ldots, k,$
   with e_j being the j-th unit vector.
3. It does not matter which classes are over- or underrepresented, i.e.:
   $\psi(x)$ is symmetric in x.

Let a split s send the proportion p_l of observations in the parent node to its left daughter node t_l and the proportion p_r to its right daughter node t_r. The decrease of impurity is then defined as

$$\Delta i(s,t) := i(t) - p_r i(t_r) - p_l i(t_l).$$

At a given node t, $\Delta i(s,t)$ is computed for all possible splits s and

$$s^* := \text{argmax}_s \Delta i(s,t)$$

is selected.
By default the CART implementation in the R package `rpart` (Therneau et al., 2017) uses the GINI impurity measure:

$$i(t) = 1 - \sum_{i=1}^{k} p^2(j|t).$$

Nodes are declared as terminal nodes when the next split would create a daughter node with less than *minobs* observations. The largest trees are grown for *minobs* $= 1$.
For individual tree classifiers more sophisticated stop criteria are commonly used. For example trees can be grown until a prespecified threshold of impurity reduction can not be reached anymore.
The terminal nodes are assigned to the class with the highest frequency of observations

in it. If more than one class has the maximal number of observations, the assignment is chosen randomly from the respective classes.

Missing Values

A big advantage of CART over other classifiers is its dynamic handling of missing values. This mechanism of surrogate variable construction is explained before a variable selection bias towards variables with many missing values is pointed out.
If one variable is missing for an observation that should be classified, it may happen that the path to one of the leafs can not be completed. One or more logical comparisons can not be made and thus the observation can not be classified. Therefore, CART saves five (by default) surrogate variables at each split. A surrogate variable implements a split in a way that mimics the optimal split as good as possible. As good as possible means that the number of observations in the two daughter nodes are as close as possible to the daughter nodes of the optimal split.

Even though the handling of missing values seems quite elegant, CART's variable selection is biased towards variables with many missing values (Strobl et al., 2007). Furthermore, there is a bias towards variables with many categories or few ties (for continuous variables). Strobl et al. (2007) give statistical reasons and examine empirical evidence for those biases. Those reasons can not be discussed in detail due to the scope of this work. However, the missing value bias is presented briefly. Note that $\hat{G}_j$, the empirical Gini Index for some split at the j-th variable, can be understood as an estimator of the true underlying Gini Index G. Strobl et al. (2007) show that this estimator is negatively biased:

$$\mathbb{E}(\hat{G}_j) = \frac{N_j - 1}{N_j} G,$$

where N_j is the number of observations the estimation is based on. This leads to the expected Gini gain

$$\mathbb{E}(\widehat{\Delta G_j}) = \frac{G}{N_j}.$$

Assuming a variable is not informative at all, the true Gini gain ΔG_j equals 0. Therefore the estimator for ΔG_j is positively biased. Missing values result in a decrease of N_j, increasing the bias of $\widehat{\Delta G_j}$. If the input variables have different numbers of missing values (i.e. different N_j), those with many missing values are artificially preferred.

4.2.2 Conditional Inferences Trees

Hothorn, Hornik and Zeileis introduced a conditional inference framework for unbiased recursive partitioning in 2006. Their so called conditional inference trees (CITs) embed the

idea of recursive partitioning into the well studied field of statistical permutation tests. This section presents their highly general framework from Hothorn et al. (2006) in the context of classification. Even though this is not discussed here in detail, CITs can be applied to continuous, discrete, censored and ordinal regression problems as well.
CITs overcome two fundamental problems of the CART algorithm. By using the concept of statistical significance, overfitting is prevented without computationally intensive pruning procedures. Considering distributional properties of the split measures allows CITs to avoid the variable selection biases of CART.
After sketching the generic algorithm, the individual steps are discussed in detail. Following the original paper, the highly general notation from Strasser and Weber (1999), which covers all mentioned regression problems, is used to describe the test statistics.

The generic algorithm for conditional inference trees is as follows:

1. For a given node, test the global null hypothesis of independence between any of the p covariates and the response to a prespecified significance level α.
2. If the hypothesis can be rejected, select the covariate X_{j^*} with the strongest association to the response. Otherwise declare the node as a terminal node.
3. Select the best split for X_{j^*} by maximizing the discrepancy between the class frequencies of the two daughter nodes. (A test statistic from the permutation test framework is used in this step.)
4. Recursively repeat Steps 1, 2 and 3 until all nodes reached their terminal node.

The global null hypothesis H_0^{glo} is tested by p partial hypothesis

$$H_0^j : X_j \perp Y \qquad \text{vs.} \qquad H_1^j : X_j \not\perp Y, \qquad j = 1, \dots, p.$$

In step 1 the algorithm faces the test problem

$$H_0^{\text{glo}} : \bigcap_{i=1}^{p} H_0^i \qquad \text{vs.} \qquad H_1^{\text{glo}} : \exists k \in \{1, \dots, p\} : \quad X_k \not\perp Y.$$

Throughout the tree growing process a binary vector $w \in \{1, 0\}^n$ indicates for each observation whether it is considered in the current node $(w_i = 1)$ or not $(w_i = 0)$. The association between Y and X_j is measured by linear statistics of the form

$$T_j(\mathcal{L}, w) = \text{vec}\left(\sum_{i=1}^{n} w_i g_j(X_{ji}) h(Y_i, (Y_1, \dots, Y_n))^T \right) \in \mathbb{R}^{p_j q},$$

with vec(.) being the vector operator combining the $p_j \times q$ matrix columnwise into a $p_j q$-dimensional vector. $g_j : \mathcal{X}_j \to \mathbb{R}^{p_j}$ is a transformation of the covariate X_j. For numerical covariates it is chosen to be the identity function. For nominal covariates with levels $1, \dots, M$ it is defined as

$$g_{ji}(x) = e_M(x),$$

where $e_l(z)$ denotes the z-th l-dimensional unit vector.

$$h : \mathcal{Y} \times \mathcal{Y}^p \to \mathbb{R}^q, \qquad h(Y_i, (Y_1, \dots, Y_n)) = e_k(Y_i)$$

is called the influence function and its summation over all considered observations captures the class frequencies.
While the unconditional distribution of $T_j(\mathcal{L}, w)$ under H_0^j is unknown, the conditional distribution, given all permutations of the responses, has been derived by Strasser and Weber (1999). Knowing the conditional expectation μ_j and covariance Σ_j of this distribution (see Hothorn et al., 2006 for the explicit forms) allows to construct a univariate standardized test statistic. In general, the statistic can be of arbitrary form, however the well known χ^2 statistic is chosen here:

$$c_{\text{quad}}(t, \mu, \Sigma) = (t - \mu)\Sigma^+(t - \mu)^T.$$

$p-$values are computed in order to decide the test problems and to compare the different covariates on the same scale. Denoting the symmetric group of all permutations of the elements of $(1, \dots, n)$ with corresponding case weights $w_i = 1$ by $\mathcal{S}(\mathcal{L}, w)$, the $p-$value for the null hypothesis H_0^j can be written as

$$P_j := \mathbb{P}_{H_0^j}\left(c_{\text{quad}}(T_j(\mathcal{L}, w), \mu_j, \Sigma_j) \geq c_{\text{quad}}\left(t, \mu_j, \Sigma_j\right) | \mathcal{S}(\mathcal{L}, w)\right).$$

This probability is computed asymptotically, using that the distribution of c_{quad} under H_0^j converges to a χ_1^2 distribution as $\sum_{i=1}^n w_i, n \to \infty$ (Rasch, 1995, Theorem 6.20).
H_0^{glo} is rejected to the level α if all P_j^{adj} are smaller than α (Hothorn et al., 2018). If so, X_{j^*} with

$$j^* := \text{argmin}_j P_j$$

is selected for determining the next split.
Step 3, selecting the best split for a given covariate, requires a measure for goodness of fit as for CART. Although the impurity measures discussed in 3.1, in particular the GINI measure, can be applied, Hothorn et al. (2006) propose a statistic within the permutation test framework due to its applicability for variables of arbitrary scale.
Consider one daughter node of a possible split as a subset of the possible variable values:

$A \subset \mathcal{X}_{j^*}$. The linear statistic

$$T_{j^*}^A(\mathcal{L}, w) = \sum_{i=1}^{n} w_i \mathbb{1}(X_{j^*i} \in A) e_k(Y_i)^T,$$

where $\mathbb{1}(.)$ denotes the indicator function, measures the class frequencies in the daughter node. Its expectation $\mu_{j^*}^A$ and covariance $\Sigma_{j^*}^A$ under H_0^j and conditioned on all permutations of the responses, can be calculated as for μ_{j^*} and Σ_{j^*}. Measure the difference between the under independence expected class frequencies $\mu_{j^*}^A$ and the observation $t_{j^*}^A$ by means of the univariate test statistic c_{quad}. Choose the split which yields the highest test statistic, i.e. which produces class frequencies (in one daughter node) furthest away from those expected under H_0^j:

$$A^* = \text{argmax}_A c_{\text{quad}}(t_{j^*}^A, \mu_{j^*}^A, \Sigma_{j^*}^A).$$

A high value of c_{quad} indicates here, that the dependency between the covariate and the response was used to find a high dependency between the class frequencies of one daughter node and the covariate. In turn, a low value of c_{quad} means that the standardized quadratic difference between the class frequencies in the daughter node and those expected under independence is low. Therefore implementing this specific split is not as effective to separate the given subset of $\mathcal{L}$ as splits with higher c_{quad} are. To emphasize this aspect, the ultimate goal of fitting a classification tree is rephrased: The individual paths to the terminal nodes should reveal a high association between the variables contained in the path and the pattern of observations (the class frequencies) in the leaf nodes' respective hyper-rectangles.
Missing values in the learning sample are ignored for tree construction. For the computation of $T_j(\mathcal{L}, w)$ and $T_j^A(\mathcal{L}, w)$ set all corresponding case weights w_i of missing values equal to 0. Missing values in new observations are handled by surrogate variables as in CART.

4.2.3 Random Forests

While the original definition of random forests (Breiman, 2001) includes different forms of tree ensembles with an additional random component, in this thesis the term random forest refers to a specific algorithm proposed by Breiman (2001). It can be seen as an extension of a bagging tree classifier and is described in accordance with Breiman (2001) in this section. Breiman's algorithm was mainly influenced by a paper on character recognition by Amit and Geman (1997). They used a randomly selected subset of variables from a large set of geometrically defined features to implement splits at each node of the trees. Breiman generalizes this idea by growing bagging tree classifiers and implementing the splits based on a subset of randomly selected variables at each node.
Denote the algorithm that constructs one classification tree by TREE. While random forests

can be constructed with all kinds of tree algorithms, in this thesis TREE stands either for a slightly feature reduced version of CART (Breiman et al., 2015) or the CIT algorithm.

The construction of a random forest, can be described as follows:

1. Choose algorithm TREE to grow the individual trees.
2. Set $ntree \in \mathbb{N}$ and $mtry \in \{1, \dots, p\}$.
3. Draw bootstrap samples $\mathcal{L}_i \quad i = 1, \dots, ntree$ from the original data set $\mathcal{L}$.
4. Fit TREE to each $\mathcal{L}_i$ with additional random component:
   At each node draw $mtry$ variables without replacement and with equal probabilities for all variables.
   Only consider those as potential split variables. The randomly chosen variables at one node are independent from those drawn in all the other nodes.

In order to classify a new data point, it is classified by all $ntree$ trees. Each tree casts a vote for the respective class. The random forest assigns the data point to the class with the most votes.

Convergence of Generalization Error

In the following paragraph an important theoretical result derived by Breiman (2001) is presented. It states that random forests do not overfit without boundary as $ntree$ increases. In other words: The generalization error approaches a theoretical limit, which can be derived by the Strong Law of Large Numbers.
In order to grasp the generalization error mathematically, first define the margin of a random forest. Given an ensemble of trees

$$\{h_1, \dots, h_{ntree}\} =: F, \qquad h_i : \mathcal{X} \to \{1, \dots, k\}$$

the margin of F is defined as

$$mg_F(X, Y) := av_k \mathbb{1}(h_k(X) = Y) - \max_{j \neq Y} av_k \mathbb{1}(h_k(X) = j),$$

where av_k denotes some form of averaging over k. This function calculates the difference between the proportion of votes for the true class and the proportion of votes for the most popular false class. It measures the confidence of a classification.
The generalization error of F can then be written as

$$PE^*(F) = P_{X,Y}(mg(X, Y) < 0),$$

which gives the probability that the entity (X, Y) is falsely classified, because there have to be more votes for a false class than for the true class so that the margin is smaller than

zero.
In random forests, every tree h_i depends on a random vector Θ_i which contains the information about the respective bootstrap sample and the randomly chosen variables at each node:

$$h_i(X) = h_i(X, \Theta_i) \quad i = 1, \dots, ntree.$$

Theorem 1.2 in Breiman (2001) states that

$$\begin{aligned} &PE^*(F) \longrightarrow LE^*(F) \quad \text{as} \quad ntree \to \infty, \\ &LE^*(F) := P_{X,Y}(P_\Theta(h(X,\Theta) = Y) - \max_{j \neq Y} P_\Theta(h(X,\Theta) = j) < 0), \end{aligned}$$

for almost surely all sequences $(\Theta_n)_{n \in \mathbb{N}}$.
Using the definition of almost surely convergence (Rasch, 1995) this can be rephrased as:

$$P_\Theta \left(\{(\Theta_n)_{n\in\mathbb{N}} \in \mathcal{F} : \quad \lim_{ntree \to \infty} PE^* \left(F((\Theta)_{n\in\mathbb{N}})\right) = ME^*(F)\} \right) = 1,$$

where $\mathcal{F}$ denotes the set of all possible sequences that generate trees for F. This leads to the following interpretation. A forest F approaches its limit of generalization error as the number of trees increases under all practically relevant circumstances. The set of sequences for which the limit is not approached has probability zero and therefore does not play a role for actual forest constructions. This cumbersome way to express a result which also can be grasped by means of intuitive explanations goes back to its proof which mainly uses the Strong Law of Large Numbers and the tree structure of $h_i \quad i = 1, \dots, ntree$ (Breiman, 2001, Appendix I).
Looking at the bigger picture of classification problems in general, this theorem gives an insight into the strengths of random forests. High dimensional datasets with a high amount of interaction require a classifier to distinguish between many regions in the variable space (Bengio, 2009). Traditional models, such as linear models adapt to such problems by allowing for many parameters (Hastie et al., 2009). As described in section 2.3, the number of regions an ensembles of trees can distinguish from another is exponential in the number of trees, assuming that the number of trees does not exceed the number of variables. This allows forests to adopt high dimensional relations with many interactions. Therefore, forests could reach high prediction accuracies in those specific settings, while other algorithms would suffer from overfitting.
Of course many algorithms can be prevented from overfitting by regularization. This term referes to constraining the parameters (often a specific norm of a vector containing all parameters) such that overfitting can be controlled (cf. Hastie et al., 2009, section 18). However, this is quite a case-specific procedure. The fact that large random forests deal with regularization in high dimensions implicitly distinguishes them from most other and

especially linear classifiers. In the next paragraph the upper bound for generalization error is investigated further.

Upper Bound for Generalization Error

Another theoretical result by Breiman (2001) identifies an upper bound for the generalization error. As the complete and mathematically rigorous explanation (cf. Breiman, 2001) goes beyond the scope of this work the result can only be sketched in the following. Consider a classification problem with the random variables X and Y. The theorem states that for any random forest F with $s \geq 0$

$$PE^*(F) \leq \frac{\bar{\rho}(1-s^2)}{s^2}, \tag{2}$$

where $\bar{\rho}$ and s are measures for the correlation in between all trees and the strength of the whole random forest respectively.
The strength of a forest (related to the accuracy) is defined through the expected margin:

$$s := \mathbb{E}_{X,Y}\left(P_\Theta(h(X,\Theta)=Y) - \max_{j\neq Y} P_\Theta(h(X,\Theta)=j)\right).$$

This intuitive definition can be motivated as follows: Pick one entity (x,y), compute its probability and the respective value of the margin function. The margin function contains now the difference of the probability (over all possible trees) to identify the true class and the maximum probability (over all possible trees) of classification for any other class.
The concept of correlation between two classification trees requires a function of which the correlation should be computed. Hence define the raw margin function

$$rmg(\Theta,X,Y) = \mathbb{1}(h(X,\Theta)=Y) - (h(X,\Theta) = \text{argmax}_{j\neq Y} P_\Theta(h(X,\Theta)=j)).$$

For observations of all three arguments it is either -1 or 1. Those two cases represent an incorrect classification for the false class with the highest probability (to be decided for) or correct classification. The correlation between two different trees with their random construction vectors Θ and Θ' measures the correlation between their two raw margin functions:

$$\rho(\Theta,\Theta') := \text{cor}_{X,Y}\left(rmg(\Theta,X,Y), rmg(\Theta',X,Y)\right).$$

$\bar{\rho}$ then describes the expected correlation over all possible tree-pairs. It can be interpreted as follows: If $\bar{\rho}$ is high, many trees agree on their classification decisions. Their decisions are highly correlated in the sense that they often decide for the same class. On the other hand if $\bar{\rho}$ is low, the individual trees in the forest tend to disagree in their votes for classification. This theorem formalizes to some extent what has already been argued by means of intuition in section 2.3. Assuming $s > 0$, which seems reasonable in all practical scenarios, the right side of inequality (2) is monotonously decreasing in s for a fixed $\bar{\rho}$. A random forest with highly accurate individual trees (high s), which disagree as much as possible (low $\bar{\rho}$) yields accurate classifications.

Out of Bag Estimates

As for now random forests were described as black box models which could not be interpreted in the subject context once the training procedure is finished. However an algorithm specific method allows to assess the importance of the variables.
In the first step a bagging specific procedure, called out of bag estimation, is used to estimate the prediction accuracy internally. For every tree, all observations which are not part of the respective bootstrap sample but part of the original data set, are called out of bag (OOB). All observations are classified by the subforest of trees for which they are OOB. This leads to an accuracy estimate without setting aside observations. In order to assess the importance of variable X_j, all trees classify their OOB observations again, but now with randomly permuted values of X_j. If X_j is very important for the classification, the random permutation will cause a substantial decrease in accuracy. In turn, the decrease in accuracy will be relatively small for an unimportant variable. Although the specific values of decrease in accuracy can not be interpreted directly, variables can be ordered by their importance in a non-parametric way.

Variable Selection Biases in Random Forests

This short paragraph elaborates an argument concerning the role of trees' variable selection biases in random forests. It is by no means a mathematical proof, but formalizes an intuition to the extent of making it mathematically feasible.
The missing value bias, discussed in 4.2.1, alongside other variable selection biases (cf. Strobl et al., 2007) can be counterpoised by varying $mtry$. Intuitively speaking, such a bias can be understood as an imbalanced probability of selecting a variable for a split. Setting $mtry = 1$ assures that all variables are selected at each split with equal probabilities. Higher values have a similar but weaker effect, as the selection bias is not present in every possible split then. Not all subsets of $mtry < p$ variables contain those which are artificially preferred by the algorithm.

For a fix classification problem, i.e.

$$(Y, X_1, \ldots, X_p) \sim \mathcal{P},$$

any classification tree can be considered as a random graph, depending on the (randomly drawn) training set. Unbiased trees follow a distribution which will be referred to as the desired tree distribution

$$T_{\text{unbiased}} \sim P_{\text{desired tree}}.$$

Algorithms with variable selection biases, such as CART, create trees which follow a biased distribution

$$T_{\text{bias}} \sim P_{\text{biased tree}}.$$

For practically relevant biases it can be assumed that $P_{\text{biased tree}} \neq P_{\text{desired tree}}$.
The desired classifier is unbiased and results from averaging all possible trees in $P_{\text{desired tree}}$ weighted by their probabilities. Only this way, the "optimal" tree incorporates the information about all possible training sets. Assuming its existence, this average can be formulated as the expectation from the distribution of unbiased trees:

$$\mathbb{E}_{\mathcal{L}}(T_{\text{unbiased}}) =: T_{\text{desired}},$$

where the index $\mathcal{L}$ emphasizes that the individual trees and thus their distribution depend on the randomly drawn training sets. Note, that the density of T_{desired} does not need to be positive, as the underlying algorithm may not be able to produce such a tree. From this perspective, averaging decisions of many trees in forests seems natural.

Bagging predictors average many trees, so that they approximate what can be understood as an expected tree:

$$av_i\{T_Z^i(\mathcal{L}_i), \quad i = 1, \dots, \text{ntree})\} \longrightarrow \mathbb{E}(T_Z), \quad \text{as ntree} \to \infty,$$

where $\mathcal{L}_i$ is the i-th bootstrap sample, av_i the majority vote averaging and Z specifies which tree algorithm is used. Therefore bagging of biased trees can not yield an unbiased classifier if

$$\mathbb{E}(T_{\text{bias}}) \neq \mathbb{E}(T_{\text{unbiased tree}}).$$

However, inducing additional randomness by setting $mtry = 1$ can under specific circumstances. This parameter setting may be able to transform the distribution so that

$$\begin{aligned} &\mathbb{E}_{\mathcal{L}}(T_{\text{bias}}^{mtry=1}) \approx T_{\text{desired}}, \\ &T_Z^{mtry=1} \sim P_{\text{random trees}}, \quad Z \in \{\text{bias}, \text{unbiased}\}. \end{aligned}$$

At least some real data experiments and simulations give empirical evidence that their prediction accuracies are more or less equal (Breiman, 2001). For higher values of $mtry$ the random forest averages trees with smaller biases (as bias does not effect every split) than

individual trees fit to the whole data set.
In conclusion, the additional randomness from small *mtry* changes the distribution of the trees. How, or with which exact consequences remains unclear here.

This claim cannot be demonstrated in a simulation easily as the `R` randomForest package (Breiman et al., 2015) does not support proper handling of missing values (it discards the whole observation if one covariate is missing). Implementing the random forest algorithm by hand, using rpart (Therneau et al., 2017), only enables bagging ensembles, i.e. $mtry = p$, as random split selection is not supported in rpart.

In practice, a data set's potential bias has to be assessed before applying random forests of trees with known selection biases. Scenarios with high potential for biases (i.e. strongly varying number of missing values in the variables, strongly varying number of categories) should not be approached with high values of *mtry*.

5 Exploratory 2D Simulation in R

The main objective of the simulation experiment is to analyze whether the random forest algorithm can profit from conditional inference trees as base learners in comparison to the (traditional) CART random forests. While CITs have several advantages when used as single tree classifiers, it is not clear whether their sophisticated statistical foundation improves prediction accuracy in conditional inference forests.
Although CART and CIT have been studied for individual trees and differences have been pointed out clearly, there is no literature that explicitly compares their behaviors in random forests.

A small two dimensional simulation serves as the basis for an exploratory comparison of both algorithms. It focuses on predictive performance and the structure (i.e. the hyper parameters) of optimal forests.
In order to identify weaknesses and strengths of the forests, different scenarios are constructed and both algorithms are fitted multiple times. The forests are not fitted for only one hyper parameter setting (i.e. a treesize parameter and the number of trees), but a whole grid of those is searched for the optimal setting. Furthermore, the resulting partitions of a specific scenario are visualized and compared to the bayesoptimal partitions. At last it is analyzed how the optimal forests look like in terms of hyper parameters.

Before the actual analysis is performed, the experimental design is explained. Remarks on the implementation in `R` (R Core Team, 2017) conclude this section.

5.1 Experimental Design

One crucial point has to be highlighted before the experimental design is explained in detail. Even though classification problems with two covariates are easy to simulate and to visualize, they can not cover the full scope of the random forest algorithm. Choosing $mtry = 1$ causes total uniformly random variable selection at each node in each tree. The goodness of fit measure is then only relevant for finding the optimal splitting threshold for a randomly preselected variable. Consequently, the structure of the forest mainly depends on the stop criterion as well as the pseudo-randomly generated numbers for the bootstrap samples and the randomly selected split variables.
Setting $mtry = 2$ yields a bagging ensemble of the respective trees. No variables are preselected before evaluating their potential for splitting. Accordingly, the correlation between the trees is influenced only by the randomly chosen bootstrap samples. In particular the additional random component of preselection can not improve the diversity of the forest.

90 different scenarios are studied in the simulation using `R` version 3.4.2 (R Core Team, 2017). Each scenario is a classification problem where each class is defined through a bivariate normal distribution:

$$\text{Class 1:} \quad X \sim \mathcal{N}(\mu_1, \Sigma_1), \qquad \text{Class 2:} \quad X \sim \mathcal{N}(\mu_2, \Sigma_2).$$

Two different relations of the two means were chosen:

$$\begin{aligned} &\text{Relation 1:} \quad \mu_1 = (-2, 2)^\top, \qquad \mu_2 = (2, 2)^\top \\ &\text{Relation 2:} \quad \mu_1 = (-2, 2)^\top, \qquad \mu_2 = (2, -2)^\top. \end{aligned}$$

For both relations, all possible tuples of covariances, from a set of covariances $\mathcal{C}$, define one scenario. $\mathcal{C}$ is constructed as follows. Choose positive values for the two parameters `covlow` and `covhigh`. Select the four tuples of those parameters as the possible pairs of variances (σ_{11}, σ_{22}). For each variance tuple, construct four 2×2 matrices. Each matrix contains either `covlow`, `covhigh` or one of their negative values on both entries in the off-diagonal. Of those 16 constructed matrices, choose the ten that are positiv semidefinite. The ten elements in $\mathcal{C}$ lead to 45 tuples of covariances. With two different mean relations those lead to 90 parameter configurations in total. `covlow` and `covhigh` were arbitrarily set to 2 and 7.

The following process was applied to each scenario.
Construct a grid of hyper parameters for both algorithms. For CART, *minobs* is varied from 1 to 25, and *ntree* can be set equal to 5, 10, 100 or 500. For CIT, $1 - \alpha$ is varied between 0 and 0.95 in steps of 0.05 and *ntree* can take the same values as for CART. This yields 100 combinations for CART and 80 for CIT forests.
Draw `L = 100` training and test sets independently, each containing 25 observations for both classes. For both grids, fit a model to all training sets for each hyper parameter combination. Hence, $100 * (80 + 100) = 18 * 10^3$ forests are fit for each scenario (leading to roughly 1.6 million models in total). Compute the prediction accuracies for all models on the respective test sets. Compute the means of all 100 prediction accuracies for all hyper parameter combinations in both grids. Select the maximum of those means for both algorithms and refer to the respective hyper parameter combinations as the optimal ones. Both values $mtry = 1$ and $mtry = 2$ were tried in two separate simulations.

The theoretical rationale for this approach is pointed out briefly.
The simulation's objective is to compare the performances of the two algorithms. After defining the scenarios, the hyper parameters have to be chosen so that the highest prediction accuracy is achieved. Even though Breiman's (2001) recommendation is to grow trees to their maximal size, no theoretical justification assures that changing the treesize parameters will decrease the prediction accuracy. In fact, inequality (2) justifies changing the treesize

in expectation of a potentially higher s without a counterpoising increase in $\bar{\rho}$.
The prediction accuracy (PA) of a random forest fit to some learning sample $\mathcal{L}$ is a random variable, as $\mathcal{L}$ and the construction vector Θ are random variables themselves. Let τ denote a vector of the treesize and *ntree* hyper parameters of one random forest. The simulation then aims to find

$$\tau^* := \text{argmax}_{\tau \in T} \mathbb{E}_{\Theta,\mathcal{L}}(PA_{\mathcal{L}}(h(\Theta, \tau))),$$

where T denotes the set of all possible hyper parameter combinations.
The expectation is estimated by

$$\widehat{\mathbb{E}_{\Theta,\mathcal{L}}}\left(PA_{\mathcal{L}}(h(\Theta, \tau))\right) = \frac{1}{L}\sum_{i=1}^{L} PA_i,$$

where PA_i denotes the prediction error for one random forest for one data set and L denotes the number of data sets.

5.2 Analysis of the Results

After comparing the performances, the structures of the fitted forests are investigated in terms of their hyper parameters. First, the results are analyzed for $mtry = 1$ and afterwards the relation to $mtry = 2$ is discussed.

Performance

The CART forests outperformed the CIT forests in most of the scenarios. In 76 out of 90 cases, the difference of the mean accuracies (CARTf − CITf) was greater than zero, whereas only in 14 scenarios it was smaller than zero. Table 1 gives an overview of the empirical distribution of these differences.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-0.006	0.003	0.010	0.010	0.015	0.047

Table 1: Summarizing Statistics for the Accuracy Differences' Distribution (CARTf − CITf)

While the CIT forests could reach an accuracy only up to 0.006 higher than the CART forests, the traditional algorithm outperformed the CIT forests by 0.047 in the most extreme case. That is, averaged over the 100 replications, the number of correctly classified observations (in the test sets of size 50) is higher by 2.4 for the CART forests. In 50% of the cases, the difference of the prediction accuracies was 1% at most. However, in 25% of the cases the CART forests were more accurate by at least 3.3%. These results have to be interpreted with caution, as the mean accuracies are estimates for the true prediction

accuracies. Variances of those estimates are not known and therefore the exact numbers only give a general intuition for the performances.
Even though the differences are quite small, it can be said with certainty that CART forests generally perform slightly better in the given setting. Figure 4 illustrates that with a histogram of the differences in mean accuracy. Many positive differences are quite small

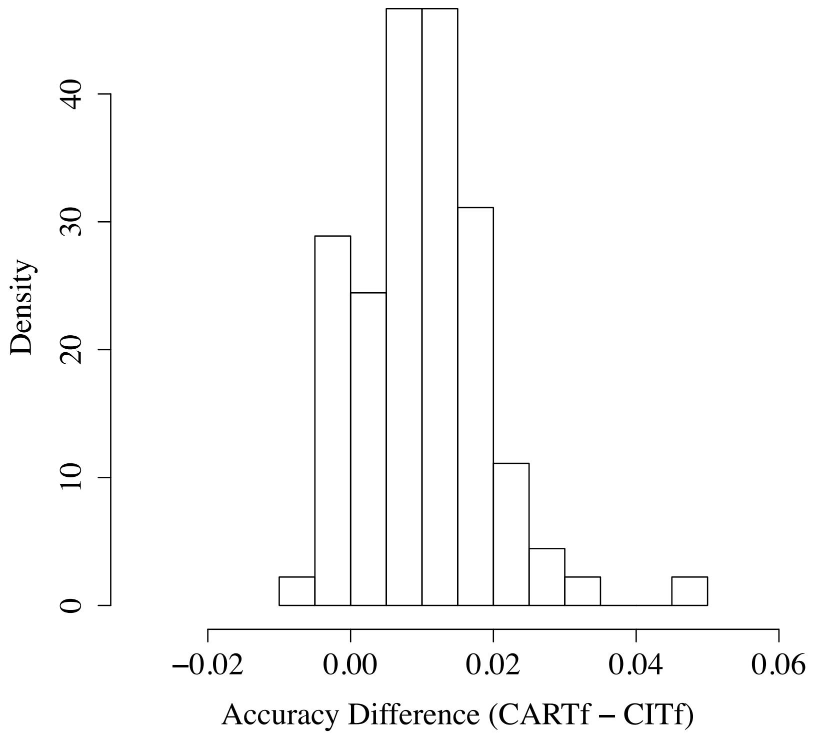

Figure 4: Histogram of Differences in Mean Accuracy

(below 0.02) and can not be interpreted as “CART outperformed CIT” with a reasonable amount of statistical confidence. On the other hand, almost all of the mass lies on the positive values. With 100 repetitions per scenarios and 90 scenarios in total, it seems quite unlikely that almost all the mass lies on the positive values just because of random variation in the accuracy difference estimates.

Visualizing the 10% of the classification problems with the highest difference in mean accuracy shows that they are structurally very similar to each other. The bayesoptimal partitions consist of straight lines, reflecting that the underlying distributions do not have the characteristic form of a normal distribution.
For those cases, the correlations between X_1 and X_2 are very high and often equal to 1. Those scenarios may not correspond to typical two dimensional classification problems but are part of the investigated set of scenarios per construction of the covariance matrices. Figure 5 illustrates such a scenario with the bayesoptimal partition and one randomly

drawn training set for the parameters:

$$\mu_1 = (-2,2)^\top, \qquad \mu_2 = (2,2)^\top$$

$$\Sigma_1 = \begin{pmatrix} 7 & -7 \\ -7 & 7 \end{pmatrix}, \qquad \Sigma_2 = \begin{pmatrix} 7 & 7 \\ 7 & 7 \end{pmatrix}.$$

The grey area in the bayesoptimal partition shows that the densities are equal at those points. This scenario resulted in the highest difference in mean accuracy (0.047), i.e. the

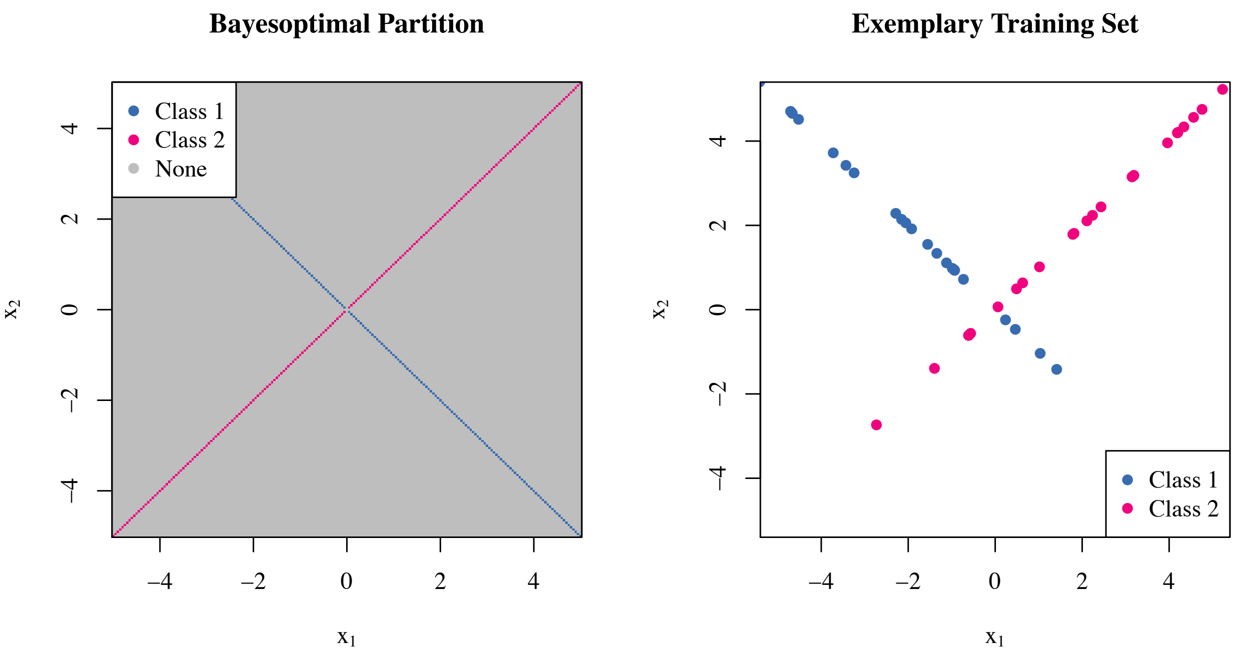

Figure 5: Bayesoptimal Partition and Exemplary Training Set for Maximal Difference Scenario

CART forest outperformed the CIT forest on average by 5%. Figure 6 visualizes the resulting partitions of both forests. 100 models were fitted, each to a separate data set, for the optimal hyper parameter configuration. Their predictions were averaged in order to visualize an estimate for the expectation of the resulting partition.

Both forests can reflect the general structure in the data. They identify a diagonal shape in the decision boundary. While the CART forest rather divides $\mathbb{R}^2$ into four equally sized quadrants, the CIT forest assigns larger areas close to the true means μ_1 and μ_2. The permutation test framework allows the CIT forest to distinguish regions with higher and lower densities (here the upper and lower quadrants), whereas the CART forest weighs them equally. On the one hand this is a desireable feature of a classification algorithm, but on the other hand it decreases accuracy in this specific scenario. Even though it seems that the CIT forest reflects more detail of the general structure in the data, the CART forest has higher prediction accuracy without doing so. The underlying reason is that all

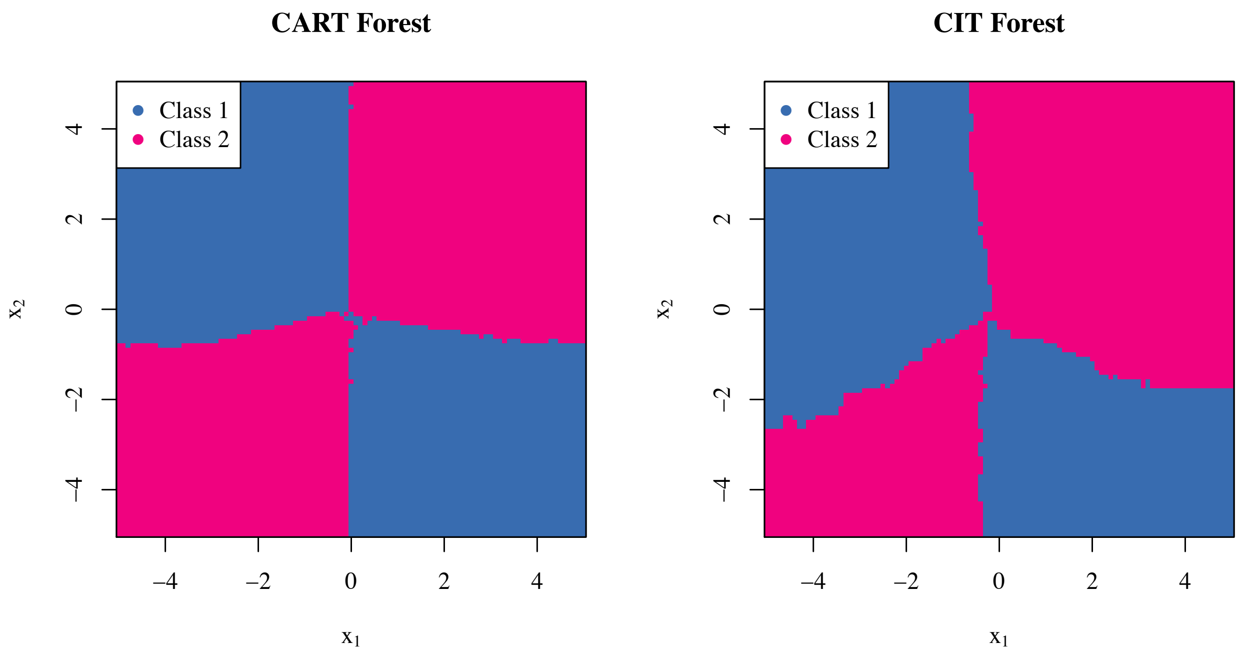

Figure 6: From 100 Models Averaged Partitions for the Maximal Difference Scenario

observations for one class lie exactly on a straight line. Therefore it is not necessary to increase the areas around μ_1 and μ_2 in order to attain high accuracy.
Both algorithms fitted large forests with large trees. The optimal CART forest consists of 100 trees with $minobs = 1$. For CIT, 500 trees and $\alpha = 1$ (i.e. no test procedure for splitting, always implement a new split) were found to be the optimal hyper parameters. In general, only highly overfitted (large) trees are able to detect diagonal structures in the data, due to the binary splitting procedure. Each split is either a vertical or horizontal line. Many of those are necessary to separate a diagonal shape from other clusters of observations. Although this is not the case here, as this problem can be solved quite well with one vertical and one horizontal line, large trees yield higher accuracy than small ones. This observation supports Breiman's (2001) initial recommendation of always growing maximally sized trees.

Forest Structures

In general, large forests yielded higher performances than small ones. In all scenarios, except for one, the CART forests consisted of 100 or 500 trees. The same is true for the CIT forests. To investigate Breiman's (2001) recommendation further, Figure 7 shows mosaicplots of the hyper parameters without the two small forests. One column represents one forestsize. The size of its individual rectangles are proportional to the fraction of observations having the respective value for the treesize parameter. Whereas the proportions of size 100 and size 500 forests are equal in the CIT case, there are more CART forests of size 500 than of size 100. The CART plot shows that most of the optimal forests consist of large trees

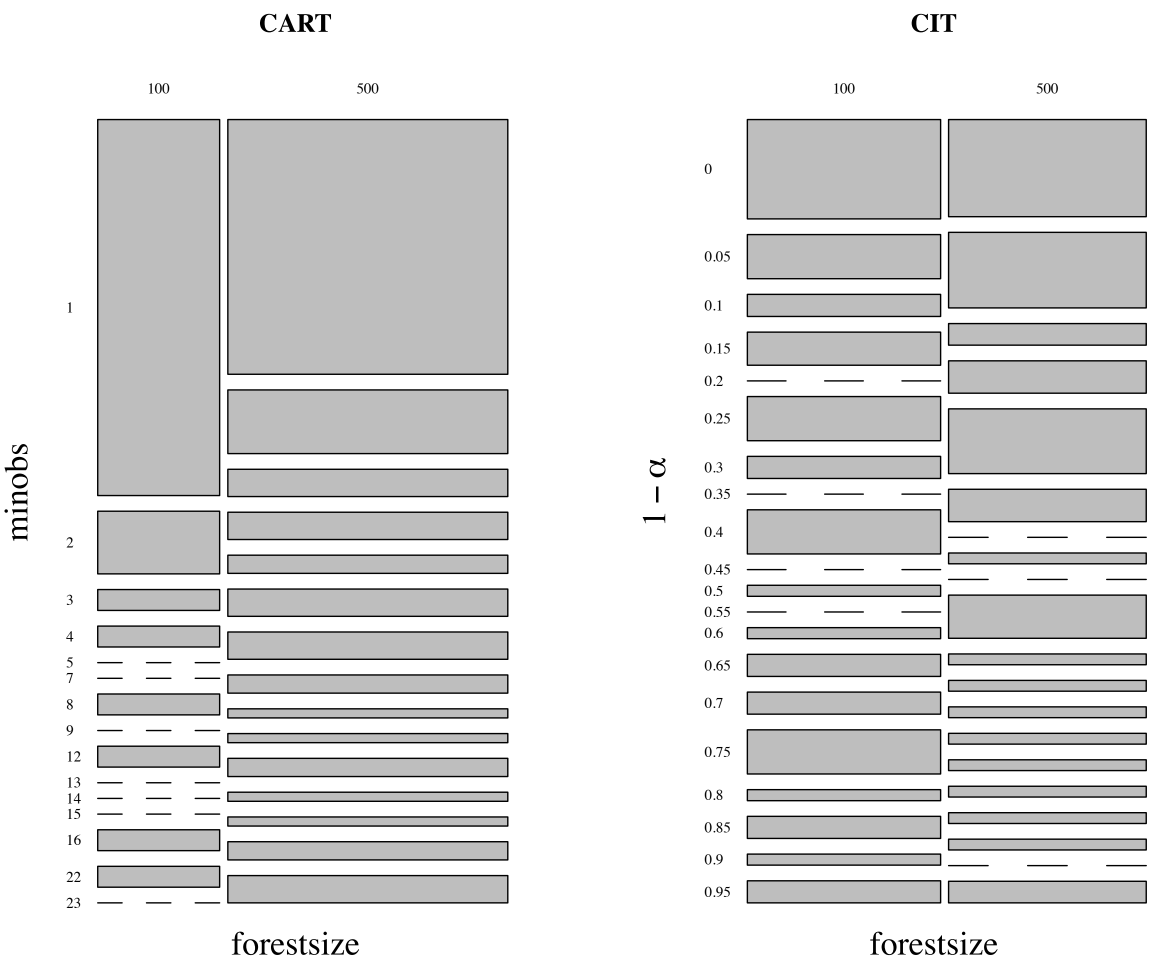

Figure 7: Mosaicplots of Optimal Hyper Parameters (small forests omitted)

($minobs = 1$). This also holds for the CIT forests, even though only to a smaller extent. Forests with large trees (i.e $1 - \alpha = 0$) make up the biggest proportions and few forests consist of small trees.

Although many optimal forests are grown with maximally large trees, this simulation shows that Breiman's (2001) recommendation (always growing trees of maximal size) is not optimal in every case. If CARTs with $minobs > 10$ (i.e. at least 10 observations in each terminal node) are considered as small, in 17% of the scenarios the optimal CART forests consist of small trees. In 30% of the scenarios CIT forests with small trees (defining them as $\alpha < 0.5$) perform better than those with large trees.

mtry = 2

Setting $mtry = 2$ instead of $mtry = 1$ does not change any major aspect of the previously presented analysis. The CART forests benefit from this hyper parameter change, as they outperform the CIT forests in 3 scenarios more than for $mtry = 1$. The specific scenario discussed alongside Figures 5 and 6 is still the one with the highest difference in mean prediction accuracy (now 3.8%). The optimal treesize parameters did not change, only the optimal forestsizes switched (500 for CIT, 100 for CART now). The resulting partitions, as well as the interpretation of the mosaicplots for the hyper parameters do not change substantially.

5.3 Code Documentation and Computational Remarks

The simulation's code is organized in six files. `make_trials.R` constructs a list of length 90 which contains the parameters (i.e. the two means and covariances) for each scenario as described in 5.1.1. `draw_fit_evaluate.R` contains the definition of a function which expects the parameters for one scenario as well as the hyper parameters for both algorithms and outputs a list of two elements. Each of those elements is one list containing the optimal hyper parameter setting as well as the respective `L` accuracies. This function is used by `sim2d.R` and `sim2d_mtry2.R` which perform the actual simulations as described in 5.1.1 in parallel on two cores.
When reasonable, the files are coded to be executable in batch mode from the command line.

```
R CMD BATCH make_trials.R
```

saves the file `trial_pars.RData` to the working directory. The simulations itself are run by

```
R CMD BATCH sim2d.R
R CMD BATCH sim2d_mtry2.R
```

which save the file `results_2d_mtry1.RData` and `results_2d_mtry2.RData` respectively. Two separate files `analysis_sim2d.R` and `analysis_sim2d_mtry2.R` produce the plots and metrics mentioned in 5.1.2 and are not executable from the command line.
`analysis_sim2d_mtry2.R` uses the same code as its version for $mtry = 1$ except for loading the data and the `my_mtry` object in line 10.
The source code for the mentioned `.R` files can be found in the Appendix. Note that `sim2d_mtry2.R` is identical to `sim2d.R` except for lines 13, 15 and 17. The name of the result object and the file in which it was saved has the suffix `mtry2`. In line 15 the additional argument `my_mtry = 2` is passed to the `draw_fit_evaluate` function.

Both simulations were run on a MacBook Air (built in 2013) with an 1,3GHz Intel Core i5 with two physical cores. They took about 5:15h (`mtry = 1`) and 6:15h (`mtry = 2`) of computation time.

The following packages were used in the simulations: mvtnorm (Genz et al., 2018), randomForest (Breiman et al., 2015), party (Hothorn et al., 2018) and parallel (R Core Team, 2017).

6 Summary, Conclusion and Outlook

This thesis explored the recursive partitioning principle for classification that is implemented by decision trees and particular ensembles of those which are referred to as random forests. In section two, two decision tree algorithms were contrasted in a thorough theoretical comparison. It was argued that variable selection biases of individual trees can be mitigated in random forests and a small exploratory simulation study was conducted. The central question was whether random forests can profit from a tree algorithm which is based on a rigorous statistical foundation in contrast to the rather heuristically motivated original tree algorithm. Moreover, the random forest algorithm (Breiman, 2001) itself was motivated exhaustively and two theoretical results were presented. One states that random forests do not overfit without boundary as the number of trees increases and the other gives an upper bound for the generalization error in terms of strengths and correlations of the individual trees (Breiman, 2001).

While the partitioning paradigm remains the same for both algorithms: choose one variable and search the optimal threshold to perform a binary split, the conceptual approaches of CARTs (Breiman et al., 1984) and CITs (Hothorn et al., 2006) are fundamentally different. Whereas CART aims to maximize the purity of the terminal nodes via an estimate for the Gini gain, CIT uses a framework of statistical permuation tests (Strasser and Weber, 1999) to embed the concept of statistical significance. By doing so, CIT overcomes two fundamental weaknesses of CART. Variable selection biases towards variables with many missing values or many categories are eliminated by taking distributional properties of the split measure into account (Strobl et al., 2007). Additionally, the stop criterion is not chosen heuristically as in CART, but can be interpreted through the significance levels of the tests.

In section 4.2.3 it was demonstrated that the effect of these weaknesses in CART are mitigated in random forests. Considering only *mtry* variables at each node for splitting reduces the impact of the variable selection bias, as not all subsets of the original variables contain all the biased ones. Choosing $mtry = 1$ eliminates the selection bias completely. Even though empirical evidence indicates that the prediction accuracy is not overly sensitive to *mtry* and even $mtry = 1$ can yield high accuracies (Breiman, 2001), it is not clear whether the completely random selection of split variables can compete with an unbiased base learner.
90 different scenarios were constructed for a two dimensional simulation study with two classes in section 5. For each scenario, 100 data sets were fit for both algorithms for a whole grid of hyper parameter combinations. The hyper parameter combination which yielded the highest mean accuracy on 100 test sets was chosen to be optimal.
Without any known bias in the data sets, the CART forests performed slightly better than the CIT forests. Even though the differences in performance are quite small (maximum

difference in prediction accuracy of 4.7%, see Table 1), the CART forest performed better in almost all scenarios (see Figure 4). While most of the optimal forests consisted of many large trees, some scenarios were solved best by large forests of small trees. On the one hand this result shows that Breiman's (2001) recommendation of always growing trees to their maximal size should be treated with caution. On the other hand it reveals that most random forests will not profit from the sophisticated stop criterion in CITs, as their optimal hyper parameter setting does not require a stop criterion at all (growing trees to maximal size). Finally the comparison of $mtry = 1$ and $mtry = 2$ supports the empirical evidence that the accuracy is not overly sensitive to $mtry$. Of course, those conclusions only apply to the very specific setting constructed for the simulation and only mathematically rigorous analyses can prove these claims without loss of generality.

In summary, no convincing argument has been found to always use random forests with CITs instead of CARTs. Regardless, there may exist scenarios which require unbiased base learners. In practice the potential for selection biases should be evaluated for each data set before applying CART random forests with high $mtry$ values.
Scenarios which require unbiased trees should be identified in future research. Those may include social and clinical surveys, where individual variables may contain a much higher amount of missing values than others.
The influence of the $mtry$ parameter on the prediction accuracy needs to be investigated further in order to fully understand the random forest mechanism. This additional random component (compared to conventional bagging) separates random forests from many other algorithms. It assures that all variables take part in the partitioning process (and fastens up the algorithm). The phenomenon that completely random split selection (i.e. $mtry = 1$) can yield high accuracies could be approached with the concept for correlation in between trees as presented in 4.2.3. Furthermore a mathematical analysis could determine the extent to which $mtry$ can counterpoise selection biases.

References

Amit, Y. and D. Geman (1997).
"Shape quantization and recognition with randomized trees".
In: *Neural Computation* 9, pages 1545–1588.

Bengio, Y. (2009). "Learning Deep Architectures for AI".
In: *Found. Trends Mach. Learn.* 2.1, pages 1–127.

Breiman, L. (1996). "Bagging Predictors". In: *Machine Learning* 24.2, pages 123–140.

Breiman, L. (2001). "Random Forests". In: *Machine Learning* 45.1, pages 5–32.

Breiman, L., A. Cutler, A. Liaw, and M. Wiener (2015).
randomForest: Breiman and Cutler's Random Forests for Classification and Regression.
`https://CRAN.R-project.org/package=randomForest`, last visit: 8/27/18.

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

Dietterich, T. G. (2000). "Ensemble Methods in Machine Learning".
In: *Multiple Classifier Systems.* Springer, pages 1–15.

Fahrmeir, L., R. Kuenstler, I. Pigeot, and G. Tutz (2007).
Statistik, der Weg zur Datenanalyse. Sixth. Springer.

Genz, A., F. Bretz, T. Miwa, X. Mi, and T. Hothorn (2018).
mvtnorm: Multivariate Normal and t Distributions.
`https://CRAN.R-project.org/package=mvtnorm`, last visit: 8/27/18.

Hartung, J., B Elpelt, and K. H. Kloesener (2009).
Statistik, Lehr- und Handbuch der angewandten Statistik. Fiveteenth.
Oldenbourg Wissenschaftsverlag.

Hastie, T., R. Tibishirani, and J. H. Friedman (2009).
The Elements of Statistical Learning: Data Mining, Inference and Prediction. Second.
Springer.

Hochberg, Y. and A. C. Tamhane (1987). *Multiple Comparison Procedures.* First.
John Wiley & Sons.

Hothorn, T., K. Hornik, C. Strobl, and A. Zeileis (2018).
party: A Laboratory for Recursive Partytioning.
`https://CRAN.R-project.org/package=party`, last visit: 8/27/18.

Hothorn, T., K. Hornik, and A. Zeileis (2006).
"Unbiased Recursive Partitioning: A Conditional Inference Framework".
In: *Journal of Computational and Graphical Statistics* 15.3, pages 651–674.

Kwona, O. H., W. Rhee, and Y. Yoon (2015). "Application of classification algorithms for analysis of road safety risk factor dependencies".
In: *Accident Analysis and Prevention* 75, pages 1–15.

Murty, R. and P. Babu (2017). “A Critical Study of Classification Algorithms for LungCancer Disease Detection and Diagnosis”. In: *International Journal of Computational Intelligence Research* 13.5, pages 1041–1048.

Neuhaeuser, M. (2011). *Computer-intensive und nichtparametrische statistische Tests.* First. Oldenburg Wissenschaftsverlag.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers.

R Core Team (2017). *R: A Language and Environment for Statistical Computing.* `https://www.R-project.org/`, last visit: 8/27/18. R Foundation for Statistical Computing. Vienna, Austria.

Rasch, D. (1995). *Mathematische Statistik.* Johann Ambrosius Barth Verlag.

Shaffer, J. P. (1995). “Multiple Hypothesis Testing”. In: *Annual Reviews of Psychology* 46, pages 561–584.

Sonquist, J.A. and J.N. Morgan (1964). *The Detection of Interaction Effects.* Survey Research Center, Institute for Social Research, University of Michigan, Ann Arbor.

Strasser, H. and C. Weber (1999). “On the Asymptotic Theory of Permutation Statistics”. In: *Mathematical Methods of Statistics* 8, pages 220–250.

Strobl, C., A.-L. Boulesteix, and T. Augustin (2007). “Unbiased split selection for classification trees based on Gini Index”. In: *Computational Statistics & Data Analysis* 52, pages 483–501.

Strobl, C., J. Malley, and G. Tutz (2010). “An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests”. In: *Psychological Methods* 14.4, pages 323–348.

Therneau, T., B. Atkinson, and B. Ripley (2017). *rpart: Recursive Partitioning and Regression Trees.* `https://CRAN.R-project.org/package=rpart`, last visit: 8/27/18.

Turau, V. and C. Weyer (2015). *Algorithmische Graphentheorie.* Third. De Gruyter Berlin.

Appendix

```r
make_trials <- function(covlow = 2, covhigh = 7){

  mean_settings <- list(matrix(2*c(-1, 1, 1, 1), ncol = 2),
                        matrix(2*c(-1, 1, 1, -1), ncol = 2))
  # Each matrix represents one two-center setting,
  # where each column contains one mean.

  # Construct set of covariances:
  possible_values <- c(-covhigh, -covlow, covlow, covhigh)
  # Define all pairs of variances (variances have to be positive):
  variances <- list(c(covlow,covlow), c(covlow,covhigh),
                    c(covhigh,covlow), c(covhigh,covhigh))
  # For each variance combination four covariances are possible
  # => 16 matrices in total.
  all_mats <-
    lapply(possible_values, function(offdiag){
      lapply(variances, function(x) matrix(c(x[1], offdiag,
                                              offdiag, x[2]), ncol = 2))})
  all_mats <- unlist(all_mats, recursive = FALSE)
  # Covariance matrices have to be psd!
  psd_mats <- lapply(all_mats,
                     function(mat) all(eigen(mat)[["values"]] >= 0))
  all_covs <- all_mats[unlist(psd_mats)]

  # Create all two tuples of all_covs
  ind_combs <- combn(length(all_covs), 2)

  cov_settings <- apply(ind_combs, 2, function(s){
                          list(all_covs[[s[1]]], all_covs[[s[2]]])})

  # length(cov_settings) * 2 lists each containing two elements
  # first: matrix for mean setting
  # second: list containing both covariance matrices
  trials <- list(lapply(cov_settings,
                        function(l) list(mean_settings[[1]], l)),
                 lapply(cov_settings,
                        function(l) list(mean_settings[[2]], l)))
  trials <- unlist(trials, recursive = FALSE)
  return(trials)
}

trial_pars <- make_trials()
save(trial_pars, file = "trial_pars.RData")
```

Source Code 1: make_trials.R

```r
library(mvtnorm)
library(randomForest)
library(party)

draw_fit_evaluate <- function(trial, L, ntrain = 25,
                              CART_nodesizes = 1:ntrain,
                              indep_levels = seq(0, 0.95, 0.05),
                              forestsizes = c(5, 10, 100, 500),
                              my_mtry = 1){
  # Input: - trial: List of two (distribution parameters for a trial).
  #               First: Matrix containing means in columns.
  #               Second: List containing the two covariance matrices.
  #        - L: Number of repitions per scenario
  #        - CART_nodesizes, indep_levels, forestsizes, my_mtry are
  #          hyper parameters for which random forests are fit.
  # Output: List of two: For both algorithms one list with the optimal
  #         hyper parameters and the respective L accuracies
  #
  # Outline: 1. Construct all combinations of hyper parameters
  #          2. Draw L trainings and test sets
  #          3. Define functions which expect training/test sets as well
  #             as one hyper parameter combination and return the
  #             prediction accuracy.
  #          4. Apply this function on all hyper parameters on all L
  #             training/test sets.

  # 1. Hyper parameter combinations
  CART_par_combs <- data.frame(
    forestsizes = rep(forestsizes, each = length(CART_nodesizes)),
    CART_nodesizes)

  CIT_par_combs <- data.frame(
    forestsizes = rep(forestsizes, each = length(indep_levels)),
    indep_levels)
  # 2. Draw L training and test sets
  make_data <- function(npoints = ntrain, e1 = trial[[1]][,1],
                        cov1 = trial[[2]][[1]], e2 = trial[[1]][,2],
                        cov2 = trial[[2]][[2]]){
    # Draw points from bivariat normal distributions
    a <- rmvnorm(n = npoints, mean = e1, sigma = cov1)
    b <- rmvnorm(n = npoints, mean = e2, sigma = cov2)
    # coerce to one data frame and assign classes
    predictors <- rbind(a, b)
    data_points <- data.frame(predictors, Y = factor(c(rep(1, npoints),
                                                       rep(2, npoints))))
    return(data_points)
  }
```

```

data_sets <- lapply(1:L, function(l)  list(train = make_data(),
                                             test = make_data()))

# 3. Functions for one hyper par setting and one pair of data sets.
one_CART_forest <- function(data_list, hyper_pars){
  # Fit forest model on training set
  CART_forest <- randomForest(Y ~ ., data = data_list[["train"]],
                              mtry = my_mtry, ntree = hyper_pars[[1]],
                              nodesize = hyper_pars[[2]])
  # Compute prediction accuracy on test set
  return(sum(predict(CART_forest, newdata = data_list[["test"]]) ==
        data_list[["test"]][["Y"]]) / (2*ntrain))
}

one_CIT_forest <- function(data_list, hyper_pars){
  # Fit forest model on training set
  my_cforest_control <- cforest_control(teststat = "quad",
                                        testtype = "Bonferroni",
                                        mtry = my_mtry,
                                        minsplit = 2,
                                        minbucket = 1,
                                        ntree = hyper_pars[[1]],
                                        mincriterion = hyper_pars[[2]])
  CIT_forest <- cforest(Y ~ ., data = data_list[["train"]],
                        controls = my_cforest_control)
  # Compute prediction accuracy on test set
  return(sum(predict(CIT_forest, newdata = data_list[["test"]]) ==
        data_list[["test"]][["Y"]]) / (2*ntrain))
}

# 4. Put everything together in order to obtain a list with a list for
#    each algorithm.
#    It contains optimal hyper parameters and resp. accuracies.

# CART:
# Take one hyper parameter tuple and fit forests to the L different
# data sets. Apply this on all hyper parameter combinations.
optim_accuracies_CART <-
  apply(CART_par_combs, MAR = 1,
        function(pars){ sapply(data_sets, function(ds){
          one_CART_forest(data_list = ds, hyper_pars = pars)})})

# CIT:
# Analogous to CART
optim_accuracies_CIT <-
  apply(CIT_par_combs, MAR = 1,
        function(pars){ sapply(data_sets, function(ds){
```

```r
            one_CIT_forest(data_list = ds, hyper_pars = pars)})})

  # Find best hyper parameter combination for both algorithms:
  # Choose the combination with the highest mean prediction accuracy over
  # all L replications.
  optim_ind_CART <- which.max(colMeans(optim_accuracies_CART))
  optim_ind_CIT <- which.max(colMeans(optim_accuracies_CIT))

  # Put those together with the respective accuracies in a list which are
  # assembled together in one list for convenience
  optim_results_CART <-
    list(optim_hyper_pars = CART_par_combs[optim_ind_CART, ],
         accuracies = optim_accuracies_CART[, optim_ind_CART])

  optim_results_CIT <-
    list(optim_hyper_pars = CIT_par_combs[optim_ind_CIT, ],
         accuracies = optim_accuracies_CIT[, optim_ind_CIT])

  return(list(CART = optim_results_CART, CIT = optim_results_CIT))
}
```

Source Code 2: draw_fit_evaluate.R

```r
# Run simulations for all trials in trial_pars.RData
library(parallel)
load("trial_pars.RData")

# Initiate cluster with two cores
simu_clust <- makeCluster(rep("localhost", 2), type = "SOCK")
# Initiate Random Number Generators on both cores
clusterSetRNGStream(simu_clust, iseed = 90)
# Prepare cores for simulation: Define the function for one trial
#                               on both cores.
clusterCall(simu_clust, function(i) source("draw_fit_evaluate.R"))
# Apply function on all trials parallel on both cores
results_2d_mtry1 <-
  parLapply(simu_clust, trial_pars, function(onetrial){
    draw_fit_evaluate(trial = onetrial, L = 100)})
# Save results and stop cluster
save(results_2d_mtry1, file = "results_2d_mtry1.RData")
stopCluster(simu_clust)
```

Source Code 3: sim2d.R

```r
library(randomForest)
library(party)

load("results_2d_mtry1_new.RData")
source("bayes_plot.R")
source("part_plot.R")
source("visualize_both.R")
load("trial_pars.RData")
results <- results_2d_mtry1
my_mtry <- 1

# 1. Performance:
# Difference of mean accuracy per scenario
acc_diffs <- sapply(results, function(lel){
 mean(lel[["CART"]][["accuracies"]]) - mean(lel[["CIT"]][["accuracies"]])
})

sum(acc_diffs > 0)
# [1] 76
sum(acc_diffs < 0)
# [1] 14

round(summary(acc_diffs), digits = 3)
#  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#-0.006   0.003   0.010   0.010   0.015   0.047

pdf("acc_diffs_hist_new.pdf")
op <- par(family = "serif", cex.axis = 1.5, cex.lab = 1.5,
          mar = c(5,5,4,2) + 0.1)
hist(acc_diffs, freq = FALSE, breaks = 15, xlim = c(-0.03, 0.06),
       main = "", xlab = "Accuracy Difference (CARTf - CITf)")
par(op)
dev.off()

# Investigate case with maximal mean accuracy difference
sorted_accs <- sort(acc_diffs, decreasing = TRUE, index.return = TRUE)
max_diff_ind <- sorted_accs[["ix"]][[1]]
max_diff_trial <- trial_pars[[max_diff_ind]]

# After looking at bayes partitions for the next five higher indices, it
# is clear, that CIT is weak at finding 'straight lines'.
bayes_plot(trial_pars[[sorted_accs[["ix"]][[9]]]], title ="Bayes")
# Extract respective optimal hyper parameters from simulation results
max_diff_hyper_CART <-
  results[[max_diff_ind]][["CART"]][["optim_hyper_pars"]]

max_diff_hyper_CIT <-
  results[[max_diff_ind]][["CIT"]][["optim_hyper_pars"]]


# Plot the models' partitions:
# Define data generating function (copied from draw_fit_evaluate.R)
make_data <- function(npoints = ntrain, e1 = trial[[1]][,1],
                     cov1 = trial[[2]][[1]], e2 = trial[[1]][,2],
                     cov2 = trial[[2]][[2]]){
  # Draw points from bivariat normal distributions
  a <- rmvnorm(n = npoints, mean = e1, sigma = cov1)
  b <- rmvnorm(n = npoints, mean = e2, sigma = cov2)
  # coerce to one data frame and assign classes
  predictors <- rbind(a, b)
  data_points <- data.frame(predictors, Y = factor(c(rep(1, npoints),
                                                      rep(2, npoints))))
  return(data_points)
}

# 1. Draw again L data sets, fit L models each and save them.
fit_models <- function(trial, L = 100, ntrain = 25, hyper_CART,
                       hyper_CIT){
  # Draw L training sets and put them in a list
  training_sets <- lapply(1:L, function(i){
    make_data(npoints = ntrain, e1 = trial[[1]][,1],
              cov1 = trial[[2]][[1]], e2 = trial[[1]][,2],
              cov2 = trial[[2]][[2]])})

  # Fit one model to each data set per algorithm
  CART_models <- lapply(training_sets, function(ds){
   randomForest(Y ~., data = ds, mtry = my_mtry,
                ntree = hyper_CART[["forestsizes"]],
                nodesize = hyper_CART[["CART_nodesizes"]])
  })

  CIT_models <- lapply(training_sets, function(ds){
    my_cforest_control <-
      cforest_control(teststat = "quad", testtype = "Bonferroni",
                      mtry = my_mtry, ntree = hyper_CIT[["forestsizes"]],
                      mincriterion = hyper_CIT[["indep_levels"]],
                      minbucket = 1, minsplit = 2)

    CIT_forest <- cforest(Y ~ ., data = ds,
                          controls = my_cforest_control)
  })

  return(list(CART = CART_models, CIT = CIT_models ))
}
set.seed(90)
max_diff_models <- fit_models(trial = max_diff_trial,
                              hyper_CART = max_diff_hyper_CART,
                              hyper_CIT = max_diff_hyper_CIT)


# 2. Per Algorithm: Predict all L models on a grid and average decisions.
```

```r
average_decision <- function(model_list, gridsize = 5, finesse = 0.1){
  # construct a 2D grid
  seq_x1 <- seq_x2 <- seq(-gridsize, gridsize, by = finesse)
  grid <- expand.grid(X1 = seq_x1, X2 = seq_x1)
  # Predict all models on this grid
  model_grids <-
    sapply(model_list, function(model){
      as.numeric(predict(model, newdata = grid))})
  # Each row resembles one point in the grid.
  # Construct average prediction
  mean_decision <- apply(model_grids, 1, function(row) round(mean(row)))
  return(list(seq_x1, seq_x2, mean_decision))
}

max_diff_av_dec_CART <- average_decision(max_diff_models[["CART"]])
max_diff_av_dec_CIT <- average_decision(max_diff_models[["CIT"]])


# 3. Visualize Both Partitions in one Plot.
#    Define Function for plotting one partition.
plot_one_parti <- function(partilist, title){
image(x = partilist[[1]], y = partilist[[2]],
      z = matrix(partilist[[3]], length(partilist[[1]]),
                 length(partilist[[2]])),
      col = c("#386CB0", "#F0027F"),
      xlab = expression(x[1]), ylab = expression(x[2]))
title(main = title)
legend("topleft", legend = c("Class 1", "Class 2"),
       col = c("#386CB0", "#F0027F"), bg = "white", pch = 16)
box()
}

# Create actual Figure with both partitions
pdf("max_diff_model_partitions.pdf", width = 9, height = 5)
op <- par(mfrow = c(1,2), family = "serif")
plot_one_parti(max_diff_av_dec_CART, title = "CART Forest")
plot_one_parti(max_diff_av_dec_CIT, title = "CIT Forest")
par(op)
dev.off()

# Create Figure with one data set and resp. bayesoptimal partition
trial <- trial_pars[[max_diff_ind]]
ex_ds <- make_data(npoints = 25)

pdf("straight_line_scenario.pdf", width = 9, height = 5)
op <- par(mfrow = c(1,2), family = "serif")
bayes_plot(trial_pars[[max_diff_ind]], title = "Bayesoptimal Partition",
           finesse = 0.05)
```

```r
plot(ex_ds[,1:2], col = c("#F0027F", "#386CB0")[(ex_ds["Y"] == 1) + 1],
     pch = 16,
     xlab = expression(x[1]), ylab = expression(x[2]),
     xlim = c(-5, 5), ylim = c(-5, 5),
     main = "Exemplary Training Set")
legend("bottomright", legend = c("Class 1", "Class 2"),
       col = c("#386CB0", "#F0027F"), pch = 16)
par(op)
dev.off()

# Parameters for max_diff scenario
trial_pars[[sorted_accs[["ix"]][[9]]]]

# 2. Forest structure
# Optimal hyper parameters for max_diff scenario (s. above)
# Mosaic Plots of Optimal Hyper Parameters:
# Extract Optimal Hyper Parameters:
get_optim_pars <- function(algo, par){
  sapply(results, function(res){
   res[[algo]][["optim_hyper_pars"]][[par]]
  })
}

optim_forestsizes_CART <- get_optim_pars("CART", "forestsizes")
optim_nodesizes_CART <- get_optim_pars("CART", "CART_nodesizes")
tab_optim_pars_CART <- table(optim_forestsizes_CART,
                             optim_nodesizes_CART)

optim_forestsizes_CIT <- get_optim_pars("CIT", "forestsizes")
optim_indep_level_CIT <- get_optim_pars("CIT", "indep_levels")
tab_optim_pars_CIT <- table(optim_forestsizes_CIT, optim_indep_level_CIT)

pdf("mosaic.pdf", width = 9, height = 6)
op <- par(mfrow = c(1,2), family = "serif", cex.axis = 1.5,
          cex.lab = 1.5, las = 1)
mosaicplot(tab_optim_pars_CART[-1, ], xlab = "forestsize",
           ylab = "minobs", main = "CART")
mosaicplot(tab_optim_pars_CIT[-1, ], xlab = "forestsize",
           ylab = expression(1 - alpha), main = "CIT")
par(op)
dev.off()

sum(tab_optim_pars_CART[,9:15])/90
# [1] 0.1666667
sum(tab_optim_pars_CIT[,11:20])/90
# [1] 0.3
```

Source Code 4: `analysis_sim2d.R`

Eidesstattliche Versicherung
(Affidavit)

Köhler, Patrick

Name, Vorname
(Last name, first name)

183663

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Recursive Partitioning: Comparing Random Forests of CART and CIT

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.