Predictive Performance of AdaBoost and Random Forest in Binary Classification Tasks

Markus Söderqvist

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2024:5
Matematisk statistik
Maj 2024

Mathematical Statistics
Stockholm University
Bachelor Thesis **2024:5**
http://www.math.su.se

Predictive Performance of AdaBoost and Random Forest in Binary Classification Tasks

Markus Söderqvist*

May 2024

Abstract

Binary classification is the task of classifying an observation into one of two classes. In this thesis we compare the predictive performance of two machine learning algorithms for binary classification, AdaBoost and random forest. We do so on nonlinear data sets, where nonlinearity is achieved by enclosing one class in a geometrical shape in the predictor space. The comparisons are conducted on data sets with (i) noise, (ii) skewed class distribution and (iii) redundant predictors. In addition, we investigate how predictor dimension affects performance. Overall, we find that the two methods have similar performance, although some differences emerge. Random forest has a higher accuracy on noisy data sets, while AdaBoost has a higher accuracy on data sets with skewed class distribution and redundant predictors. Moreover, AdaBoost tends to outperform random forest on data sets with higher predictor dimension. The cost of this advantage is a considerably longer runtime. These findings are in line with previously reported findings. One unexpected finding is that the performances of both methods improve when the class distribution is skewed. A further analysis shows that one class is easier to classify at the expense of the other class for skewed data sets. Therefore, one should be careful about drawing conclusions from these results. Finally, an in-depth analysis in higher predictor dimensions shows that random forest has superior accuracy on one class while AdaBoost has superior accuracy on the other class. One possible explanation could be how the algorithms are constructed, and this can have important implications for choice of method in other classification problems.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: soderqvist.markus@gmail.com. Supervisor: Ola Hössjer and Johannes Heiny.

Acknowledgements

I would like to sincerely thank my supervisors Ola Hössjer and Johannes Heiny for their encouragement, guidance and rewarding theoretical discussions during the process of writing this thesis. Artificial intelligence was not used in the writing of this thesis.

Contents

1 Introduction

Machine learning is an interdisciplinary field at the intersection of statistics and computer science that is concerned with the question of how algorithms can *learn from data*. (Hastie et al., 2017, pp. 1). Although inference and prediction are important areas both in classical statistics and machine learning, the former has generally focused more on inference while the latter has focused more on prediction (Lindholm et al., 2022, pp. 18). In this thesis, we will focus on prediction and *binary classification*, i.e., prediction where the variable to be predicted is binary.

Some of the machine learning algorithms for classification have been around for some time. For example, the idea behind the algorithm k-nearest neighbors dates back to around 1030, and it was described in the literature 1967 (Lindholm et al., 2022, pp. 36). Other examples of machine learning algorithms for classification that have been used for a while are neural networks (since the 1980's) and support vector machines (since the 1990's) (James et al., 2013, pp. 6).

But it has not been until the advent of more powerful computational resources that the potential of these algorithms has started to be fully realized. In the recent decades, classification algorithms have been used for a wide variety of tasks such as detection of spam email (Hastie et al., 2017, pp. 2), prediction of treatment outcome in psychotherapy on the basis of brain imaging data (Månsson et al., 2015) and classification of soil based on smartphone pictures (Pandiri et al., 2024).

Over the years, more sophisticated and effective classification algorithms have emerged. In this thesis, we will focus on two popular classification algorithms, AdaBoost and random forest. AdaBoost was invented in 1995 by Yoav Freund and Robert Schapire (Breiman, 1996, pp. 17; pp. 20) and random forest in 2001 by Leo Breiman and Adele Cutler (Liaw, 2018, pp. 1). Both algorithms share similarities; both are so-called ensemble methods that average predictions from several learners (Hastie et al., 2017, pp. 605), but they also have important differences.

While both algorithms have been used successfully in a wide range of problem settings, a number of limitations have been observed for both methods. AdaBoost has been shown to be sensitive to data that contains noise (Dietterich, 2000, pp. 147; Li et al., 2017, pp. 5; Long & Servedio, 2008, pp. 1). In contrast, random forest has been shown to be more robust against noise compared to AdaBoost (Breiman, 2001, pp. 1; Dietterich, 2000, pp. 147). However, random forest has been reported to have difficulties with data where the distribution of classes is skewed (Dudoit & Fridlyand, 2003, pp. 138; Zhu, 2020, pp. 3). Moreover, a reduction in performance has also been noted on data with redundant predictors (Kubus, 2018, pp. 1; Nguyen et al., 2015, pp. 4).

1.1 Aims

The primary aim of this thesis is to investigate and compare the predictive performance of AdaBoost and random forest on simulated data. In light of the findings of previous studies mentioned above, we will also explore weaknesses and limitations of both methods by simulating data sets where we have isolated specific characteristics. These characteristics are (i) noisy data, (ii) skewed class distributions and (iii) redundant predictors. In addition, we will also investigate how the number of predictors affects the predictive performance of both methods. We will also describe their theoretical underpinnings to better understand how both algorithms work.

1.2 Disposition

Section 2 provides the reader with the necessary theoretical background and starts with a brief introduction to machine learning and related concepts. The purpose of this section is to explain key concepts that are important to understand before moving forward. The reader that is already familiar with the fundamental concepts of machine learning can skip this section. The theoretical review then continues with a review of decision trees. After that we introduce ensemble methods, bagging, boosting and the specific bagging and boosting algorithms that we will compare in this thesis. In Section 3 we describe the simulation study. The thesis continues with Section 4 where we present the results. In Section 5 we discuss the results and suggest possible improvements and further developments for future studies. The thesis concludes with Section 6 where we summarize our main findings.

2 Theory

Unless otherwise stated, the theoretical background that we will present is based on Hastie et al. (2017) and Lindholm et al. (2022), which also have influenced our choice of mathematical notation to a large extent.

2.1 Learning paradigms

This section is based on chapter 1 in Hastie et al. (2017), unless otherwise noted. Prediction problems that can be solved with machine learning algorithms can be divided into *supervised*, *unsupervised*, and *semi-supervised* learning problems (Lindholm et al., 2022). In the supervised learning setting, we want to predict an outcome of some output variable (commonly called *response* or *target*) given realizations of one or several input variables (commonly called *predictors* or *features*). To put it another way, one could say that the input data is labeled. In contrast, the unsupervised learning setting only provides us with a set of input variabes; the input data is unlabeled. The goal in unsupervised learning is to find and describe patterns and associations between the input variables. In the semi-supervised learning paradigm we have both labeled and unlabeled data (Lindholm et al., 2022, pp. 247). From here on, we will employ the names predictor and response for input and output variables, respectively.

Supervised learning problems can be further categorized into *regression* and *classification* problems. Regression encompasses all problems where the response is quantitative, while classification concerns the case where the response is qualitative.

2.2 Supervised learning as function approximation

This section is based on section 2.6 in Hastie et al. (2017), unless otherwise stated. From a mathematical perspective, supervised learning algorithms can be understood as different ways of approximating a function by analyzing its input and output values. Let $\boldsymbol{X} = (X_1, ..., X_p)^T$ be a p-dimensional vector of predictors and let Y be the response. Let $\mathcal{D} = \{\boldsymbol{x_i}, y_i\}_{i=1}^N$ be a set of independent data points. Furthermore, let the function

$$Y = f(\boldsymbol{X}) + \epsilon \tag{1}$$

describe how $\boldsymbol{X}$ and Y are related. This function maps the i:th observation vector $\boldsymbol{x_i} = (x_{i1}, ..., x_{ip})^T$ in the predictor space $\mathbb{R}^p$ to the i:th observed response y_i. The term ϵ denotes the error in each observation introduced due to measurement error and the fact that we restrict ourselves to the p predictors while we ignore other variables that also have an influence on Y. The error term does not depend on $\boldsymbol{X}$ and it is irreducible. Furthermore, we assume that its expected value $\mathbb{E}[\epsilon] = 0$.

Suppose that we now are given a previously unseen and unlabeled observation $\boldsymbol{x_\star} = (x_{\star 1}, ..., x_{\star p})^T$ and that we would like to compute a prediction $\hat{y}_\star$ of the corresponding response $y_\star$. Now, the function $f(\boldsymbol{X})$ is unknown. In order to compute the prediction $\hat{y}_\star$, we therefore construct a function $\hat{Y} = \hat{f}(\boldsymbol{X})$ that estimates $f(\boldsymbol{X})$ as well as possible. Put differently, we want the differences $y_i - \hat{f}(\boldsymbol{x_i})$ to be as small as possible.

An important distinction to make here is between *parametric* and *nonparametric* models. Parametric models rest on an assumption about the form of $f(\boldsymbol{X})$; it is assumed that the function can be described by a finite set of parameters and the function approximation amounts to estimating these parameters. In contrast, nonparametric models do not have any such assumption; the number of parameters can even be infinite (Held & Bové, 2014, pp. 10).

2.3 Training and evaluating a model

Unless otherwise stated, this section is based on chapter 2 in Lindholm et al. (2022). In order to approximate $f(\boldsymbol{X})$, we need labeled data from which the algorithm can learn the relation between $\boldsymbol{X}$ and Y. In the machine learning community this is commonly called *training* the model.

Therefore, we assemble a *training data* set by randomly selecting a subset of observations $\mathcal{T} = \{\boldsymbol{x_i}, y_i\}_{i=1}^{n}$ from $\mathcal{D}$ of size $n < N$. The subset of $N - n$ data points from $\mathcal{D}$ that is not in $\mathcal{T}$ will not be "seen" by the algorithm during training and will instead serve as *test data*, with which we can evaluate the predictive performance. One can explicitly write $\hat{f}(\boldsymbol{X}; \mathcal{T})$ to make it more evident that the approximated function will depend on the data that it has been trained on (Lindholm et al., 2022, pp. 64).

We want the algorithm to make decisions during training that lead to a model with high predictive accuracy on training data. We also want to assess the performance of the final model on new data to make sure it has not overfit, as well as comparing its performance with other models. For this, we need a means of measuring prediction error. This is done by evaluating a *loss function*, from here on denoted $L(y_i, \hat{f}(\boldsymbol{x_i}))$. During training, the loss is also called *training error*. By averaging the loss over all data points, we get a measure of how well the model fits to the data. This way, the problem of minimizing prediction error during training becomes a function minimization problem. One usually prefers a loss function to be convex since this ensures that a unique minimum can be found, but this is not always possible (Lindholm et al., 2022, pp. 113). In regression problems, the mean squared error (MSE) $\sum_{i=1}^{n}(y_i - \hat{f}(\boldsymbol{x_i}))^2/n$ is probably well known to the reader as a common measure of prediction error, but this measure is not applicable in classification problems. We will describe appropriate choices of loss functions further below. In some cases, no closed form solution exists that can be computed analytically. In these cases, there are several numerical iterative optimization algorithms that can be used depending on the situation, for example gradient descent and Newton Raphson's method to name two (Lindholm et al., 2022, section 5.4). We will not discuss optimization algorithms further.

It is important to note that the choice of loss function to minimize during training and choice of loss function when evaluating the model do not have to be the same. Sometimes it is even preferable to have different loss functions as this can lead to smaller prediction error on new data (Lindholm et al., 2022, pp. 65).

2.4 The bias-variance tradeoff

This section is based on section 4.4 in Lindholm et al., 2022), unless otherwise noted. In (1) the error term ϵ is considered random noise specific to the training data that can not be measured, and as such it should not be modeled as systematic covariation between $\boldsymbol{X}$ and Y. This phenomenon is known as *overfitting* and happens when the model fits the training data too closely. Such a model will perform poorly on unseen data. On the contrary, a model that generates large prediction errors on training data is said to *underfit* to training data, i.e. it misses systematic covaration that is present between $\boldsymbol{X}$ and Y. Related to overfitting and underfitting is the term *model complexity*. A model that overfits to training data is said to have too high model complexity whereas a model that underfits to training data has too low model complexity. The term model flexibility is sometimes also used to describe this property.

Thus, an optimal level of model complexity should lead to a model that to a large extent neither overfits nor underfits to training data. The reader might be familiar with the decomposition of the MSE of an estimator into its variance and squared bias (Held & Sabanés Bové, 2014, pp. 55). It turns out that this decomposition is possible for a machine learning model as well. We will in the following restrict ourselves to the case of regression with MSE as loss function, as this particular case is illustrative. Let $\mathbb{E}[(\hat{f}(\boldsymbol{x}_\star) - y_\star)^2]$ denote the expected MSE for the new observation $\boldsymbol{x}_\star$, i.e. the error that we would arrive at if we repeatedly retrained our model on a large number of different training data sets, computed the MSE for each training data set and then computed the average MSE over all training data sets. It can be shown that the expected MSE for a new observation $x_\star$ can be expressed as follows (James et al., 2013, pp. 34):

$$\begin{aligned}\mathbb{E}[(\hat{f}(\boldsymbol{x}_\star) - y_\star)^2] &= \mathbb{E}[(\hat{f}(\boldsymbol{x}_\star) - f(\boldsymbol{x}_\star) - \epsilon)^2] \\ &= \mathrm{Var}(\hat{f}(\boldsymbol{x}_\star)) + [\mathrm{Bias}(\hat{f}(\boldsymbol{x}_\star)]^2 + \mathrm{Var}(\epsilon). \qquad (2)\end{aligned}$$

Formula (2) illustrates the *bias-variance tradeoff*. A model with low model complexity will generally lead to an estimator of $f(x_\star)$ with low variance, meaning that if we computed estimates $\hat{f}(\boldsymbol{x}_\star)$ several times by training the model on different data sets $\{\mathcal{T}_t,\ t = 1, 2, ...\}$, the predictions $\hat{f}(\boldsymbol{x}_\star; \mathcal{T}_t)$ would not vary that much. Moreover, a model with low model complexity will lead to an estimator of $f(x_\star)$ with high bias, meaning that the difference between the true mean $f(\boldsymbol{x}_\star)$ and the expected value of the prediction $\mathbb{E}[(\hat{f}(\boldsymbol{x}_\star)]$ is high. This leads to systematic errors in the predictions (James et al., 2013, pp. 34); the model underfits. In contrast, high model complexity is generally associated with high variance and low bias. Thus, an increase in model complexity leads to a decrease in bias and increase in variance, increasing the risk of overfitting. This is illustrated in Figure 1, taken from Hastie et al., 2017, pp. 38. Therefore, finding the optimal level of model complexity is a central task in training a machine learning model. This also illustrates why one can not evaluate and compare models based on their performance on training data, as this approach likely would lead to choosing an overfit model that would perform poorly on new data.

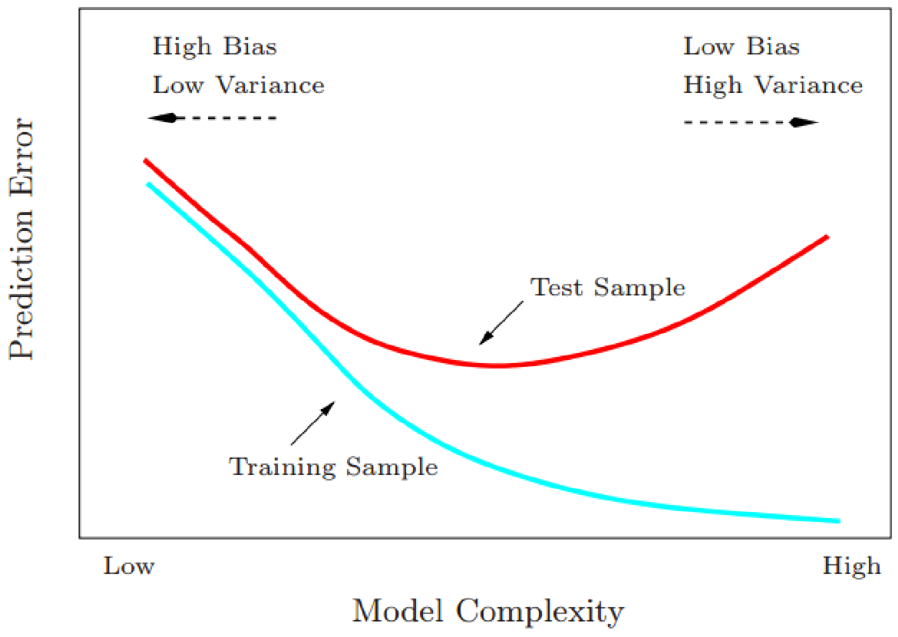

Figure 1: The bias-variance tradeoff. Underfitting occurs for low model complexity; the prediction error for the training sample and test sample is high. Overfitting occurs for high model complexity; prediction error is low on training sample and high on test sample. The optimal model complexity achieves the lowest prediction error for the test sample.

2.5 Hyperparameters

An important step in finding the optimal model complexity is optimizing, or tuning, one or several *hyperparameters*, also called tuning parameters. A hyperparameter is a parameter whose value is not learnt by the model itself but instead left to the user to choose (Lindholm et al., 2022, pp. 22). Hyperparameters control the complexity of the model and thus the tendency of the model to underfit or overfit to the training data (Hastie et al., 2017, pp. 222). The optimal value can be chosen by training the model on the training data set with different hyperparameter values and compute the prediction error on a *validation* data set for each model. The hyperparameter value that leads to the model with the smallest validation error is then chosen (Lindholm et al., 2022, pp. 129-130).

2.6 Learning setting in the present study

In this thesis, we will confine ourselves to supervised nonparametric binary classification where the predictors are continuous. Consequently, all components of $\boldsymbol{X}$ will henceforth be considered continuous and Y will be considered a binary variable. Depending on the context, Y will either take on the values -1 or 1 (Section 2.9), or 0 or 1 (Section 2.10), for ease of presentation. In the latter case, $f(\boldsymbol{X})$ is particularly easy to interpret. In this case, $f(\boldsymbol{X})$ is the conditional probability density function $p(Y|\boldsymbol{X})$, with $\mathbb{E}[Y|\boldsymbol{X}] = p(\boldsymbol{X})$ (Hastie et al., 2017, pp. 29). Furthermore, we have that $p(Y = 0|\boldsymbol{X} = \boldsymbol{x}) = 1 - p(Y = 1|\boldsymbol{X} = \boldsymbol{x})$. If $\hat{p}(Y = 1|\boldsymbol{X} = \boldsymbol{x}_\star) > 0.5$, we consequently predict $\hat{y_\star} = 1$, and $\hat{y_\star} = 0$ otherwise (Hastie et al., 2017, pp. 11).

2.7 Decision trees

This section is based on section 9.2 in Hastie et al. (2017), unless stated otherwise. Decision trees subdivide the predictor space into a set of disjoint subregions each having a hyperrectangular shape with boundaries parallel to the coordinate axes. The rules for splitting the predictor space can be visualized in a binary tree, hence the name decision tree. Decision trees are nonparametric since there is no underlying assumption about the form of $f(\boldsymbol{X})$. In this thesis we will look at a specific type of decision tree algorithm called CART (acronym for Classification And Regression Trees). Since this thesis covers classification, we will treat the classification case.

Before we describe how to grow a decision tree for classification, it is informative to look at a trained decision tree. Assume that we have a subdivision of the predictor space in K regions $R_1, ..., R_K$. In each region (also called leaf node or terminal node) R_K, we will predict the class c_k. The function approximation can be expressed as

$$t(\boldsymbol{X}) = \sum_{k=1}^{K} c_k \mathbb{I}(\boldsymbol{x} \in R_k),$$

where $\mathbb{I}$ is the indicator function. The best prediction $\hat{c}_k$ in each region R_k is the most frequently occurring class among the training observations in that region; a majority vote. In a region R_k containing n_k observations, the proportion of observations having observed class $m = 1, \ldots, M$ is

$$\hat{p}_{km} = \frac{1}{n_k} \sum_{i:\boldsymbol{x_i} \in R_k} \mathbb{I}(y_i = m),$$

where binary classification corresponds to $M = 2$. The predicted class $\hat{y}$ in region R_k is thus

$$\hat{y} = \arg \max_{1 \leq m \leq M} \hat{p}_{km}. \tag{3}$$

2.7.1 Recursive binary splitting

Determining the best partition, or split, of the predictor space by testing all possible partitions is generally not computationally attainable as the number of possible partitions is simply too large. We therefore resort to a greedy algorithm called *recursive binary splitting* when choosing how to split the predictor space. This algorithm is recursive and greedy since the tree is grown one split at a time and the algorithm chooses the split that results in the lowest training error after that one split, without taking influence of future splits into account (Lindholm et al., 2022, pp. 28). Let the integer l represent the index of predictor X_l and the real number ξ a value such that a split is performed at $X_l = \xi$. We have now created a *node* where the predictor space is split into the two regions

$$R_1(l,\xi) = \{\boldsymbol{X} | X_l \leq \xi\} \text{ and } R_2(l,\xi) = \{\boldsymbol{X} | X_l > \xi\}.$$

To find the optimal split, we want to identify the pair (l,ξ) that minimizes the loss, measured by some loss function $L(y_i, \hat{f}(\boldsymbol{x_i}))$:

$$\arg\min_{l,\xi} \left(\sum_{j:\boldsymbol{x_j} \in R_1} L(y_j, t(\boldsymbol{x_j})) + \sum_{h:\boldsymbol{x_h} \in R_2} L(y_h, t(\boldsymbol{x_h})) \right).$$

After this split, the splitting process is repeated in the two regions R_1 and R_2.

The partitioning of the predictor space in disjoint regions is illustrated in Figure 2, taken from Hastie et al., 2017, pp. 306. For illustrative purposes, we restrict our ourselves to a 2-dimensional predictor space consisting of the predictors X_1 and X_2.

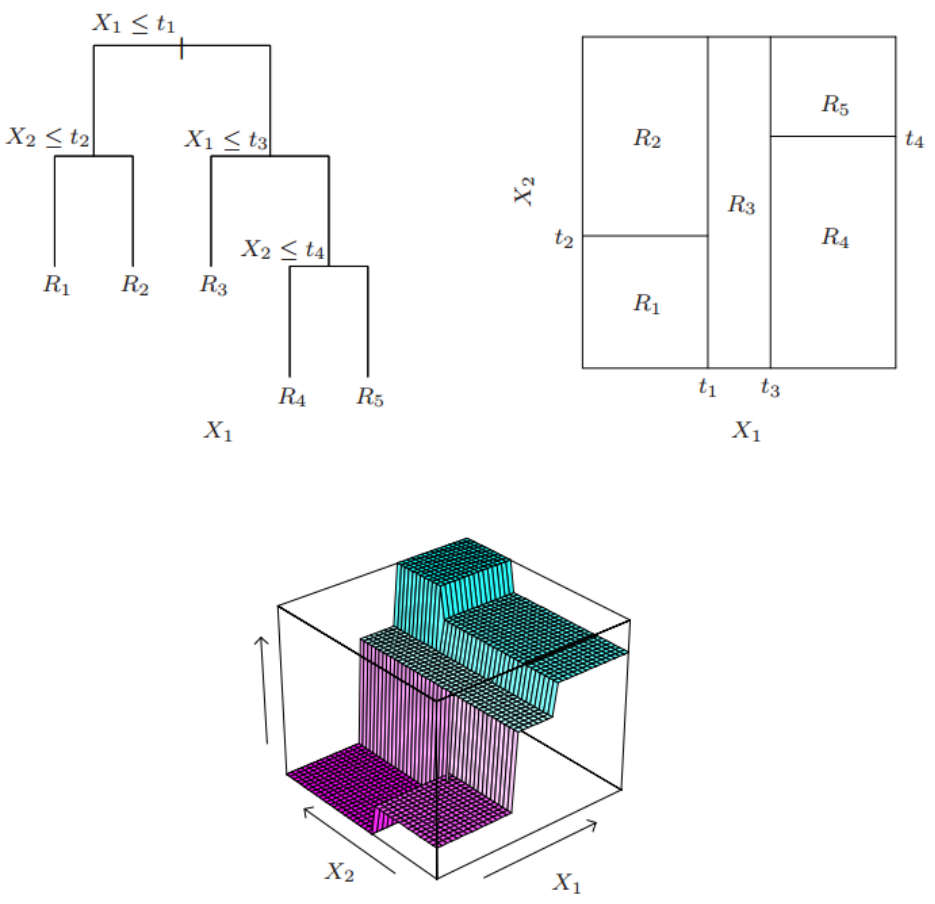

Figure 2: Illustration of a decision tree in the multiclass classification case for two predictors X_1 and X_2. In the upper left corner a binary tree is shown and in the upper right corner its corresponding partitioning of the predictor space. The bottom figure shows a perspective of the predictor space where different levels on the vertical axis represent different predicted classes.

2.7.2 Loss functions

We will now present some of the common loss functions when training decision trees. Perhaps the most intuitive loss function is the misclassification loss:

$$L_M = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i = \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{y}_i)$$

For a classification tree, minimization of L_M leads to a majority classifier (3) within each region of the tree. When we introduce AdaBoost we will show that the decision trees in this algorithm are trained by minimizing the *weighted misclassification loss:*

$$L_W = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq \hat{y}_i)}{\sum_{i=1}^{n} w_i}, \tag{4}$$

where $w_i > 0$ is a weight that is applied to observation i, reflecting how much influence that particular observation has on the loss. However, one of the downsides with these measures is that they are not sensitive to *node purity*; whether a majority of the observations in a certain leaf node belong to the same class. In general, choosing a loss function that is sensitive to node purity is an advantage when growing the tree with recursive binary splitting, since this tends to lead to fewer splits (Lindholm et al., 2022, pp. 33). Furthermore, the misclassification and weighted misclassification loss functions are not differentiable and can therefore not be minimized with common numerical optimization methods.

When training random forests we will instead use the *Gini index*, a loss function sensitive to node purity that also is differentiable. It is defined as

$$L_G = \sum_{m=1}^{M} \hat{p}_{km}(1 - \hat{p}_{km})$$

within region k. Another widely used loss function that also has these properties is *entropy*: $L_E = -\sum_{m=1}^{M} \hat{p}_{km} \ln \hat{p}_{km}$, where ln is the natural logarithm (Lindholm et al., 2022, pp. 34).

2.7.3 Hyperparameters

This section is based on section 2.3 in Lindholm et al. (2022). A common way to avoid growing the tree too deep, i.e. avoid making it overfit to training data, is to decide on a stopping criterion beforehand. There are several approaches to this. One approach is to decide on a maximum number of terminal nodes. Another approach is to decide on a minimum number of training observations in each terminal node. Fewer training observations in each terminal node leads to higher model complexity, the extreme case being when there is one training observation in each terminal node, leading to a perfect fit to training data and thus overfitting.

2.8 Ensemble methods

This section is based on section 16.1 in Hastie et al. (2017), unless stated otherwise. Single decision trees have a high variance and therefore run the risk of overfitting; even small changes in training data can result in a drastically different tree structure. One way to deal with this problem is to make sure not to grow the tree too deep. One can also use *pruning*, where sections of the fully grown tree are removed (Lindholm et al., 2022, pp. 36).

We will not use pruning in this thesis and will therefore not explore it further here. Nevertheless, even with optimization of hyperparameters and the use of pruning, the predictive performance of individual decision trees fall short compared to some of the other frequently used machine learning algoritms (James et al., 2013, pp. 340). This motivates the use of *ensemble methods*; train several *base learners* and choose the most frequently predicted class among the base learners as the prediction. As we shall see, in some ensemble methods this choice is weighted so that some base learners have a stronger influence on the prediction than others. Decision trees are common base learners in ensemble methods. In the following two subsections we will look at two popular types of ensemble methods and introduce the algorithms that will be used in the simulation study.

2.9 Boosting

This section is based on section 7.3 in Lindholm et al. (2022), unless stated otherwise. The idea behind *boosting* has at least historically been to use base learners with high bias and low variance, also called *weak learners*, that individually are only marginally better than chance. In the context of decisison trees, this means a shallow tree. Sometimes the decision tree is just a *decision stump*; a tree having depth one, which is what we will use as base learner in this thesis. The base learners are trained in sequence in which each learner aims to reduce the error of the previous learner. This is done by *re-weighting* the data so that more weight is given to the training observations that were misclassified by the previous learner. The manner in which this re-weighting is done differs between boosting algorithms. The prediction is then made by a weighted majority vote; the predictions from each learner are given a weight in such a way that more accurate learners have a stronger influence on the final prediction. Combining several weak learners in this manner leads to bias reduction. It should be noted that lately, the requirement of using a base learner with low model complexity has been disputed (Wyner et al., 2017).

2.9.1 Forward stagewise additive modeling

Unless stated otherwise, this section is based on section 10.3 in Hastie et al. (2017). For ease of presentation when reviewing the theory behind boosting and AdaBoost, the response Y will take on the values -1 or 1. Let B denote a hyperparameter describing the total number of decision trees to be trained by the boosting algorithm and let β_b, $\beta_b > 0$, denote the weight coefficient that assigns the weight to decision tree b according to its accuracy. Furthermore, let γ_b denote the set of pairs (l, ξ) that define the nodes as well as predicted classes in each terminal node in decision tree b. The function approximation for boosting can then be expressed as:

$$g(\boldsymbol{X}) = \sum_{b=1}^{B} \beta_b t(\boldsymbol{X}; \gamma_b). \tag{5}$$

Boosting can thus be thought of as a special case of an additive model where the basis functions are machine learning models (Lindholm et al., 2022, pp. 184). If we let $t_b(\boldsymbol{X})$ denote the b:th decision tree in the training sequence, the final prediction is given by:

$$\hat{Y} = \text{sign}\left[\sum_{b=1}^{B} \beta_b t_b(\boldsymbol{X})\right].$$

An additive model like (5) is usually fitted by minimizing some loss function $L(y, g(\boldsymbol{x}))$ averaged over training data:

$$\arg \min_{\{\beta_b, \gamma_b\}_{b=1}^{B}} \sum_{i=1}^{n} L\left(y_i, \sum_{b=1}^{B} \beta_b t(\boldsymbol{x}_i; \gamma_b)\right). \tag{6}$$

This type of optimization problem is often computationally demanding. *Forward Stagewise Additive Modeling* is a greedy approach that approximates the solution to (6) by iteratively computing and adding one basis function at a time without modifying the previous terms in the expansion. In our setting, this means that in each iteration b, a shallow decision tree $t(\boldsymbol{X}; \gamma_b)$ and its corresponding weight coefficient β_b are trained and chosen as to minimize the loss. The pseudo code for the algorithm is shown in Algorithm 1.

Algorithm 1 Forward stagewise additive modeling.

```
$g_0(\boldsymbol{X}) \leftarrow 0$
for $b = 1$ to $B$ do
    $(\beta_b, \gamma_b) = \arg \min_{\beta,\gamma} \sum_{i=1}^{n} L(y_i, g_{b-1}(\boldsymbol{x}_i) + \beta t(\boldsymbol{x}_i; \gamma))$
    $g_b(\boldsymbol{X}) \leftarrow g_{b-1}(\boldsymbol{X}) + \beta_b t(\boldsymbol{X}; \gamma_b)$
end for
```

2.9.2 AdaBoost

This section is based on section 10.4 in Hastie et al. (2017), unless stated otherwise. If we use the exponential loss function $L(y, g(\boldsymbol{x})) = e^{-yg(\boldsymbol{x})}$ in Algorithm 1, we end up with the *Adaboost* (Adaptive Boosting) algorithm. The product $yg(\boldsymbol{x})$ is known as the *margin.* Correct classifications leads to positive margins, while misclassifiations leads to negative margins, since $Y \in \{-1, 1\}$. We will now show how the individual base learners $t_b(\boldsymbol{x}_i)$

are trained and how their corresponding weight coefficients β_b are computed by the algorithm. As we will see below, working with the exponential loss function will prove convenient as it leads to a closed form expression that does not require numerical optimization. In each iteration b, the decision tree $t_b(\boldsymbol{X})$ is trained first and then the optimal value for β_b is computed. We therefore start with $t_b(\boldsymbol{X})$. In each iteration we need to solve

$$(\beta_b, t_b) = \arg\min_{\beta,t} \sum_{i=1}^{n} \exp\{-y_i(g_{b-1}(\boldsymbol{x}_i) + \beta t(\boldsymbol{x}_i))\}. \tag{7}$$

Now we make use of the fact that $e^{u+v} = e^u e^v$ and let $w_i^{(b)} = \exp\{-y_i g_{b-1}(\boldsymbol{x}_i)\}$. Expression (7) can then be rewritten as

$$(\beta_b, t_b) = \arg\min_{\beta,t} \sum_{i=1}^{n} w_i^{(b)} \exp\{(-\beta y_i t(\boldsymbol{x}_i))\}. \tag{8}$$

Since $w_i^{(b)}$ does not depend on β or $t(\boldsymbol{x}_i)$, it can be interpreted as a weight applied to training data point $\boldsymbol{x}_i$ that we can treat as a constant. Expression (8) can now be rewritten as

$$(\beta_b, t_b) = \arg\min_{\beta,t} \left(e^{-\beta} \cdot \sum_{i:y_i = t(\boldsymbol{x}_i)} w_i^{(b)} + e^{\beta} \cdot \sum_{i:y_i \neq t(\boldsymbol{x}_i)} w_i^{(b)} \right), \tag{9}$$

since correct classifications are associated with positive margins, leading to the factor $e^{-\beta}$ in the first term, while misclassifications lead to negative margins, giving rise to the factor e^{β} in the second term. Expression (9) can furthermore be rewritten as

$$(\beta_b, t_b) = \arg\min_{\beta,t} \left((e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i)) + e^{-\beta} \cdot \sum_{i=1}^{n} w_i^{(b)} \right). \tag{10}$$

Since $\beta_b > 0$, we have that $e^{\beta} - e^{-\beta} > 0$. Hence, the solution for $t_b(\boldsymbol{X})$ in each iteration b is

$$t_b(\boldsymbol{X}) = \arg\min_{t} \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i)).$$

Thus, in AdaBoost we train the individual decision trees using the weighted misclassification loss L_W (4) as loss function.

Now, it remains to show how to compute β in each iteration. This derivation is based on section 7.3 in Lindholm et al. (2022). In order to find the optimal value of β, we differentiate the expression in (10) with respect to β, set this derivate to zero and solve for β:

$$0 = \frac{\partial}{\partial \beta} \left((e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i)) + e^{-\beta} \cdot \sum_{i=1}^{n} w_i^{(b)} \right)$$

$$\begin{aligned}
&\iff 0 = (e^{\beta} + e^{-\beta}) \cdot \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i)) - e^{-\beta} \cdot \sum_{i=1}^{n} w_i^{(b)} \\
&\iff \sum_{i=1}^{n} w_i^{(b)} = \frac{(e^{\beta} + e^{-\beta}) \cdot \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i))}{e^{-\beta}} \\
&\iff \sum_{i=1}^{n} w_i^{(b)} = (e^{2\beta} + 1) \cdot \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i)) \\
&\iff e^{2\beta} = \frac{\sum_{i=1}^{n} w_i^{(b)}}{\sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i))} - 1 \\
&\iff \beta = \frac{1}{2} \ln \left(\frac{\sum_{i=1}^{n} w_i^{(b)}}{\sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(y_i \neq t(\boldsymbol{x}_i))} - 1 \right),
\end{aligned}$$

or equivalently,

$$\beta = \frac{1}{2} \ln \left(\frac{1 - L_W}{L_W} \right).$$

We conclude this section by presenting the AdaBoost algorithm as pseudo code in Algorithm 2 (taken from Hastie et al., 2017, pp. 339).

Algorithm 2 AdaBoost for classification.

```
for $i = 1$ to $n$ do                                  ▷ Initialize the weights.
    $w_i \leftarrow \frac{1}{n}$
end for
for $b = 1$ to $B$ do
    Train a decision tree $t_b(\boldsymbol{X})$ using weights $w_i$.
    Compute
        $L_W^{(b)} = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq t_b(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i}$.
    Compute
        $\beta_b = \frac{1}{2} \ln \left( \frac{1 - L_W^{(b)}}{L_W^{(b)}} \right)$.
    for $i = 1$ to $n$ do
        $w_i \leftarrow w_i \cdot \exp\{\beta_b [2 \cdot \mathbb{I}(y_i \neq t_b(x_i))]\}$.
    end for
end for
Output $g(\boldsymbol{X}) = \text{sign}\Big[\sum_{b=1}^{B} \beta_b t_b(\boldsymbol{X})\Big]$.
```

2.10 Bagging

This section follows section 7.1 in Lindholm et al. (2022). Another popular ensemble method is bootstrap aggregating, also called *bagging*. Like boosting, it can be implemented with different types of base learners. In contrast, the base learners in bagging have low bias and high variance, i.e. high model complexity. In the context of decision trees, this means trees that are grown deep. Such models can *individually* easily overfit to training data. The idea behind bagging is to reduce the variance of the model without increasing its bias. This is achieved by training the base learners on different but overlapping bootstrapped samples of the training data set $\mathcal{T}$, where each sample has the same size as the training data set. A bootstrapped sample is obtained by randomly drawing observations from the training data set with replacement, a procedure known as nonparametric bootstrapping (Held & Bové, 2014, pp. 66). Some observations will occur more than once in a given bootstrapped sample and will thus have more impact during training, whereas some observations will not occur at all. The final prediction is given by the unweighted majority vote of the base learner predictions. In summary, both boosting and bagging achieve improved accuracy but do so in different ways; boosting reduces bias while bagging reduces variance.

We will now show how bagging reduces variance and end with a motivation behind the second algorithm in this thesis. Let $z_1, ..., z_S$ be identically distributed random variables with mean μ and variance $\text{Var}(z_s) = \sigma^2$ for $s = 1, ..., S$. Moreover, let ρ be the average correlation coefficient between any pair of these variables:

$$\rho = \frac{\frac{1}{S(S-1)} \sum_{e \neq d} \mathbb{E}[(z_e - \mu)(z_d - \mu)]}{\sigma^2}.$$

We have that $\mathbb{E}\big[\frac{1}{S} \sum_{s=1} z_s\big] = \mu$, so the mean is unchanged by the averaging. Furthermore, we have that

$$\text{Var}\Big(\frac{1}{S} \sum_{s=1} z_s\Big) = \frac{1-\rho}{S}\sigma^2 + \rho\sigma^2. \tag{11}$$

In other words, if $\rho < 1$, the variance is reduced. As mentioned in Section 2.4, predictions from a machine learning model can be seen as random variables themselves. Since all base learner predictions are based on the same data $\mathcal{T}$ (through nonparametric bootstrapping), they are identically distributed but not independent, since they are correlated. When we average their predictions we decrease the variance of the prediction in accordance with (11). Since the mean is not altered, the bias remains low. We note that the first term in (11) can be made smaller by increasing S, but the second term equals $\rho\sigma^2$, independently of S. Thus, the size of the reduced variance obtained by bagging is bounded from below by the correlation between the base learner predictions. Therefore, a valid question is if it is possible to lower this correlation in order to get further variance reduction. The answer to this question leads us to our second machine learning algorithm.

2.10.1 Random forest

This section is based on section 7.2 in Lindholm et al. (2022) unless stated otherwise. A *random forest* is a modification of bagging that applies to CART decision trees. It turns out that decision trees in general are suitable for bagging since they have high variance. It has even been noted that having a base learner with high variance is essential in order for bagging to improve accuracy (Breiman, 1996, pp. 3). The purpose of the random forest algorithm is to get further variance reduction by decorrelating the bagged decision trees. This is achieved by putting a restraint on the number of predictors that are available as splitting variables in each split, rather than having the recursive binary splitting algorithm choose from all predictors. In each split, q predictors are selected at random after which the algorithm determines the optimal splitting variable and splitting criterion among these q predictors. Since q is not chosen by the algorithm itself, it is a hyperparameter. A recommended choice for the classification setting is to set $q = \sqrt{p}$ (rounded down to the nearest integer), which also is the default value that we will leave unchanged when training random forest models in this thesis.

We will now give an intuition as to why inducing randomness in the selection of splitting variables decorrelates the trees. As described in Section 2.7.1, the splitting is done greedily in a recursive manner. Imagine a scenario where one predictor explains the majority of the variation in the response. Even if we use bagging, it is very probable that all decision trees will select this predictor as the first splitting variable so that all decision trees end up with similar first splits, making them highly correlated. Random forest thus circumvents this problem by forcing some of the decision trees to use other predictors as splitting variables, making the decision trees less similar and thus less correlated. One could argue that this approach would degrade the performance of the individual decision trees, but since we are averaging a large number of decision trees, this is not a problem. As mentioned previously, we will use the Gini index L_G as loss function when growing the decision trees. Regarding stopping criteria for each tree there are several options. We will use the hyperparameter minimum node size n_{min}. The random forest algorithm is illustrated as pseudo code (taken from Hastie et al., 2017, pp. 588) in Algorithm 3 below.

Algorithm 3 Random forest for classification.

```
for $a = 1$ to $A$ do
    1. Draw a bootstrap sample $\mathcal{T}_a = \{(\boldsymbol{X}_{ai}, Y_{ai})\}_{i=1}^n$ of size $n$ from $\mathcal{T}$.
    2. Grow a random forest decision tree $T_a(\boldsymbol{X}_{ai})$ to the bootstrapped
       data by repeating the following steps recursively for each node
       until minimum node size $n_{min}$ is reached for each terminal node:
        (a). Select $q$ predictors at random from the $p$ predictors.
        (b). Select a predictor $l$ and value $\xi$ where the split will be
             performed such that the total loss $L_G$ within the node is
             minimized.
        (c). Split the node into two child nodes.
end for
Output majority vote$\big(\{T_a(\boldsymbol{X})\}_{a=1}^A\big)$ at each $\boldsymbol{X}$.
```

3 Simulation and modeling

In order to compare the predictive performance of AdaBoost and random forest, a simulation study will be conducted. We will in the following describe the simulation and modeling procedure.

3.1 Simulation of the predictors $\boldsymbol{X}$

The first step in the simulation process is to simulate the predictor values. For each observation $i = 1, ..., n$, a sample of predictor values $\boldsymbol{x_i} = (x_{i1}, ..., x_{ip})^T$ is simulated from the p-dimensional multivariate normal distribution with mean vector $\boldsymbol{\mu} = (0, ..., 0)^T$ and covariance matrix $\boldsymbol{\Lambda}$ equal to the p-dimensional identity matrix (Gut, 2009, pp. 125):

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \Big(\frac{1}{2\pi}\Big)^{p/2} \exp\big\{-\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\big\}, \ \boldsymbol{x} \in \mathbb{R}^p.$$

In the data sets described below, we will vary the number of predictors from 2 to 300.

3.2 Simulation of the response Y

The next step is to simulate the values of the response Y. Since both methods are highly flexible and able to capture complex and nonlinear relationships, the aim is to simulate data sets with nonlinear relationships between the predictors $\boldsymbol{X}$ and response Y. Our approach to obtain nonlinearity is to define geometrical shapes in the predictor space and assign observations lying inside and outside of the shapes to different classes. Observations that are enclosed by the shapes are assigned to the class $y = 0$ while the remaining observations lying outside the shapes are assigned to the class

$y = 1$. The idea of using geometrical shapes is inspired by a previous thesis (Basetti, 2019).

In all data sets, we will work with either equilateral triangles or circles when the data consists of two predictors and more generally, equilateral pyramids and n-spheres as the number of predictors grows. We will either have one shape or two smaller shapes in the data to assess how this affects the predictive performance of the methods. When the number of nonredundant predictors is greater than three, we will only have single n-spheres due to the challenges associated with programming n-simplexes (i.e. triangles in higher dimensions). In all data sets, the coordinates of the shape centroids are kept equal to ensure that all shapes are located in regions with similar density. In data sets with two triangles or pyramids, the second shape is rotated 180° to limit the extent of overlap between the two shapes. Moreover, the default sizes of the shapes are adjusted to ensure equal class distributions. In Figure 3 and 4 data sets with one and two shapes are illustrated with two predictors.

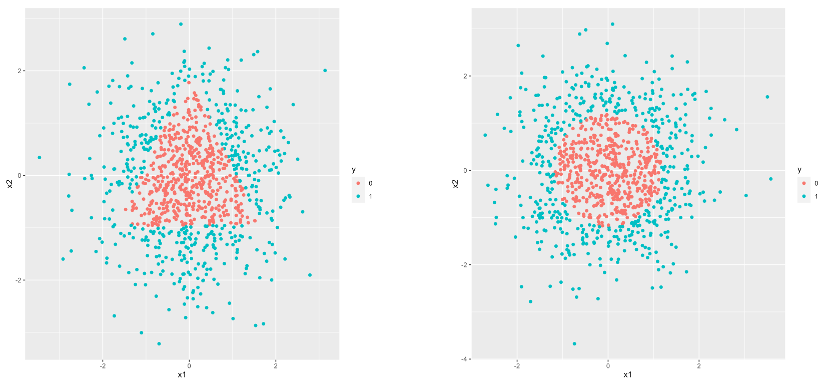

Figure 3: Simulated training data sets with two predictors where observations from one class (red) are enclosed in one shape.

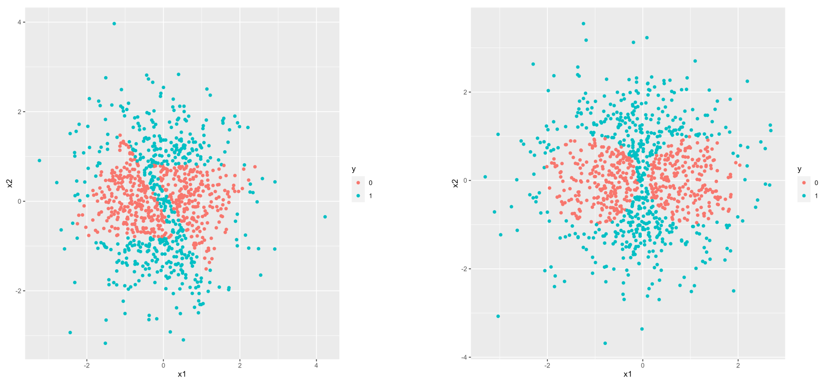

Figure 4: Simulated training data sets with two predictors where observations from one class (red) are enclosed in two shapes.

3.3 Noisy data

Noise in the data is obtained by specifying a probability ϵ that the regions contain the other class when assigning the classes. Thus, for observations enclosed by the shape we have that $P(Y = 1) = \epsilon$ and for observations outside the shape we have that $P(Y = 0) = \epsilon$. Data sets that have no noise correspond to setting $\epsilon = 0$. Throughout the simulation study, the data sets with noise will have $\epsilon = 0.2$. The effect of noise is illustrated with two predictors in Figure 5.

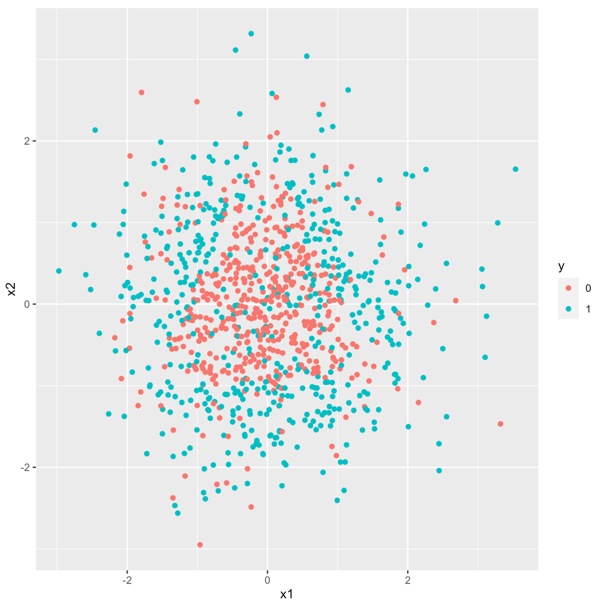

Figure 5: Simulated training data set with two predictors and noise where observations from one class (red) are enclosed in a triangle.

3.4 Skewed class distribution

Class distribution is controlled by varying the size of the shapes. An equal class distribution means that the data is evenly distributed between the two classes, i.e. 50 % of the observations belong to one class and the other 50 % belong to the other class. In the simulated data sets with skewed class distribution, about 20 % of the observations belong to class $y = 0$ while the remaining 80 % belong to class $y = 1$. The class distributions are allowed to differ 2 % from the intended class distribution, meaning that a training data set with 1000 observations and skewed class distribution has at least 180 observations and at the most 220 observations belonging to class $y = 0$. Skewed class distribution is illustrated in the case of a triangle with two predictors in Figure 6.

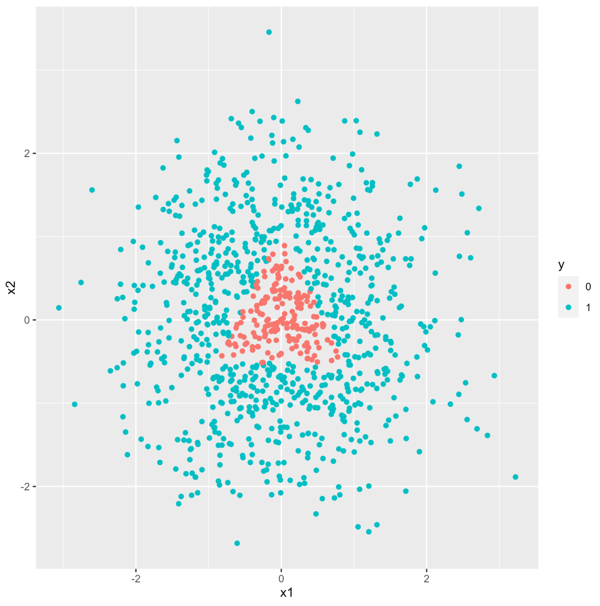

Figure 6: Simulated training data set with two predictors and skewed class distribution where observations from one class (red) are enclosed in a triangle.

3.5 Redundant predictors

Redundant predictors will only be present in data sets with more than three predictors. They are obtained by letting a smaller subset of predictors define the shapes that determine the classes. If we let p_n denote the number of predictors in this subset and p_r the remaining predictors, we have that $p = p_n + p_r$. The p_r predictors that do not define the shapes will be independent with respect to the response y and thus redundant. If the dimension of the predictor space is equal to the dimension of the shape, the number of redundant predictors is zero. In order to assess different proportions of nonredundant predictors, we will simulate several different scenarios. We will simulate an additional seven redundant predictors when the number of nonredundant predictors is three as well as five and ten when the number of nonredundant predictors is five. When the number of nonredundant predictors is 50, we will simulate an additional 50 and 100 redundant predictors. When the number of nonredundant predictors is 100, we simulate an additional 50, 100 and 200 redundant predictors.

3.6 Generation of data sets

To summarize, data sets with different characteristics are obtained by varying the number of nonredundant predictors (2, 3, 5, 10, 50, 100), number of shapes (one or two), type of shape (n-simplex or n-sphere), noise ($\epsilon = 0$ or $\epsilon = 0.2$), class distribution (50/50 or 20/80) and number of redundant predictors (0, 5, 7, 10, 50, 100 and 200). Noise, skewed class distribution and redundant predictors are mutually exclusive characteristics so that we can study the effect of each characteristic in isolation.

3.7 Modeling

For each type of data set described above, the models will be trained on training data sets each consisting of 1000 observations. While both methods (AdaBoost and random forest) are known to perform well without any hyperparameter tuning (Hastie et al., 2017, pp. 340, 590), we will try to do some elementary tuning for both methods. For optimization of hyperparameters, a separate validation data set of 250 observations will be used. When optimizing hyperparameters, we will limit ourselves to the number of weak learners B for AdaBoost and minimum node size n_{min} for random forest and we will start with the default values. For random forest, this corresponds to setting the node size n_{min} to 1, which corresponds to a fully grown tree. We will thus leave the number of trees A at the default value of 500. We will also leave the hyperparameter q that controls the number of available predictors in each split at its default value $q = \sqrt{p}$. For AdaBoost, we will use decision stumps (i.e., trees with depth one or put differently, two nodes) and we will start with the default number of weak learners $B = 50$. We will then try different values for B and n_{min} for Adaboost and random forest respectively and select the values that produce the highest accuracies on the validation data sets.

Due to time restraints and the vast number of data sets, we will not optimize the hyperparameters for each data set individually. However, we will tune the hyperparameters for data sets with greater than three and three or fewer nonredundant predictors separately, as we will see that the number of predictors will have a profound impact on the model fit. In addition, we will also increase the node size for random forest on the data sets with noise, as this will prove to improve the performance on these data sets. After the hyperparameters have been optimized, the models are tested repeatedly on 50 test data sets each consisting of 250 observations. The average accuracy is then computed for each method and type of data set.

3.8 Software

All data simulation and modeling is done in the programming language *R*. The *mvrnorm()* and *rbinom()* functions from the *MASS* package are used to simulate multivariate normally distributed predictor values and to produce noise in the data, respectively. For data manipulation and visualization, we use the *Tidyverse* package. For training and predictions of random forest and AdaBoost models, the packages *randomForest* and *ada* are used. The *caret* package is used to calculate evaluation metrics. For measuring runtimes, the *tictoc* package is used.

4 Results

Performance and results from hypothesis testing of performance differences are presented in two tables. Metrics on data sets with two and three nonredundant predictors is shown in Table 1 and metrics for more than three nonredundant predictors is shown in Table 2. When interpreting the t-tests, we have chosen a significance level of 0.05. Additional tables over classification accuracies across different hyperparameter settings and classification accuracies separated by class are located in the appendix. Below we will analyze the impact of different properties of the data sets.

4.1 Main findings

Overall, the accuracy of both methods is quite similar and observed differences in accuracy are moderate. As additional challenges such as noise, redundant predictors and especially, increased number of predictors are introduced in the data sets, some notable discrepancies emerge. The differences get more pronounced as the number of predictors grows.

On data with two nonredundant predictors, the methods have similar performance on data sets that do not contain noise, skewed class distribution or redundant predictors. From here on, we will refer to these data sets as *neutral* data sets. On data with three predictors, AdaBoost tends to outperform random forest on neutral data sets. When the data contains noise, random forest tends to outperform AdaBoost on data sets with two and three nonredundant predictors. This is reversed for data sets with skewed class distribution and redundant predictors.

When the number of nonredundant predictors is higher than three, AdaBoost, now trained with a higher number of weak learners, starts to outperform random forest on most data sets. The exception is a majority of the data sets that contain noise, where random forest still has a significantly better performance. Overall, AdaBoost tends to perform better than random forest on data sets with redundant predictors. However, when the predictor dimension is large, this is not always the case.

However, a further analysis of classification accuracy for each class separately, in Table 8 of the Appendix, uncovers an interesting discrepancy. In higher predictor dimensions, AdaBoost has notably higher classification accuracy than random forest on observations inside the shapes. This is reversed for observations outside the shapes for which random forest has a notably higher accuracy than AdaBoost. The advantage of AdaBoost in the former case is larger than the advantage of random forest in the latter case, making AdaBoost the best performing classifier overall in larger predictor spaces.

4.2 Impact of hyperparameters

After validating the performances with different values for number of weak learners B for AdaBoost and minimum node size n_{min} for random forest, a compromise was made when deciding on hyperparameter values for comparisons between the models, as there was no clear-cut hyperparameter value that outperformed the others on all types of data sets. The values that yielded good overall performance on a majority of the data sets were chosen. For AdaBoost, we ended up training the models with $B = 250$ on all data sets with two and three nonredundant predictors and $B = 1000$ on all data sets with more than three nonredundant predictors. For random forest, we trained the models with $n_{min} = 1$ on data sets with two and three nonredundant predictors, except for the data sets with noise where we instead ended up with $n_{min} = 15$. We trained all random forest models with $n_{min} = 20$ on data sets with more than three nonredundant predictors.

We refer the reader to Tables 4 - 7 in the Appendix for an overview of the accuracies of AdaBoost and random forest across different hyperparameter settings on all types of data sets. It should be noted that some modifications of the number of redundant predictors in data sets with 50 and 100 nonredundant predictors were made after selection of hyperparameters, so these data sets differ from the ones in Table 2 in this section.

Looking at how different number of weak learners B affect performance of AdaBoost in Table 4 in the Appendix, we see that the performance is not affected that much by altering B for data sets with two and three predictors. A more dramatic impact is seen for data sets with 50 and 100 nonredundant predictors in Table 5. We see that $B = 1000$ overall leads to worse performance compared to a lower values of B on data sets where the number of nonredundant predictors is five or less and superior performance on data sets with 50 and 100 nonredundant predictors. When training AdaBoost with $B = 50$, the performance is often worse than the other values regardless of data set, and this discrepancy is particularly pronounced on data sets with a higher number of predictors.

For random forest, the overall impact of altering the hyperparameter values is less pronounced, as seen in Table 6 and 7 in the Appendix, although we observe some patterns here as well. For smaller predictor dimensions, $n_{min} = 1$ outperforms higher values of n_{min} in a majority of the data sets. For larger predictor dimensions, higher values of n_{min} tend to perform better.

4.3 Impact of shape and number of shapes

In data sets with one shape containing one class, the performances are similar for spheres/circles and triangles/pyramids. When one class is enclosed by two shapes, both methods perform worse on triangles/pyramids compared

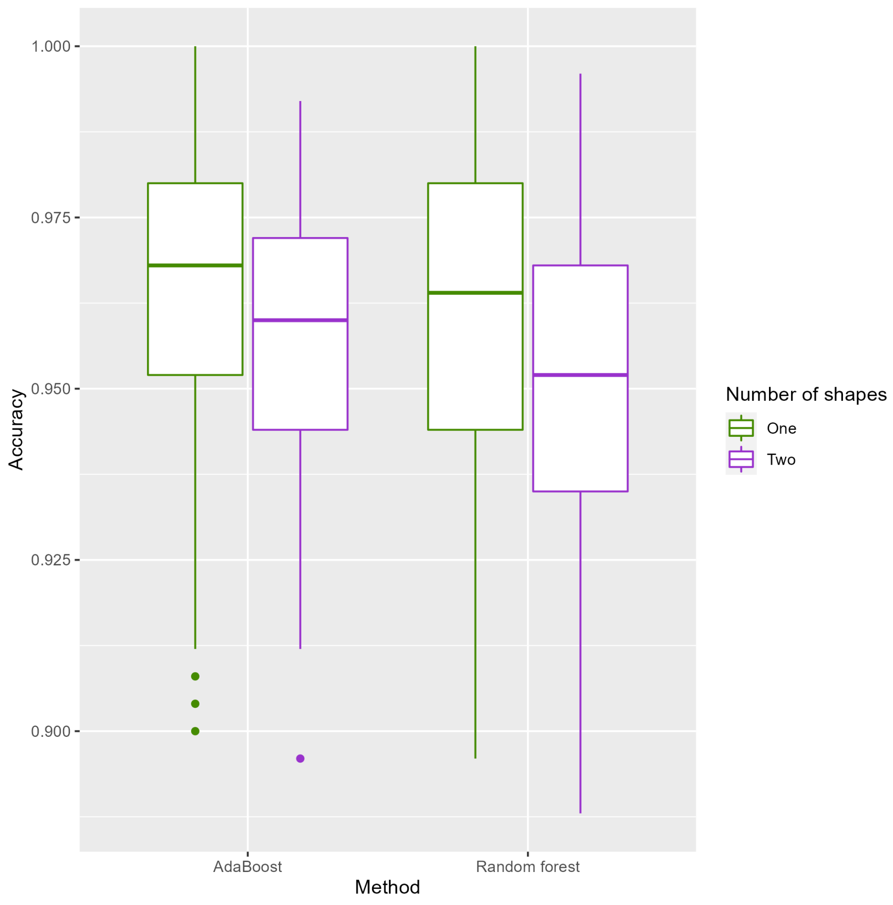

Figure 7: Box plots showing the effect of the number of shapes on accuracy for both methods on data sets with two and three nonredundant predictors.

to circles/spheres. This difference is small when the number of predictors is two and more pronounced when the number of predictors is three. The accuracy for both methods decreases as a second shape is introduced in the data set, the exception being when a second sphere is introduced with three predictors. In Figure 7, we have averaged the accuracy for one and two shapes over all data sets with two and three predictors. We see that the accuracy of random forest decreases a little more compared to AdaBoost when an additional shape is introduced. In Figure 8, we see that the dispersion of the accuracy is higher on data sets with circles/spheres compared to triangles/pyramids for both methods.

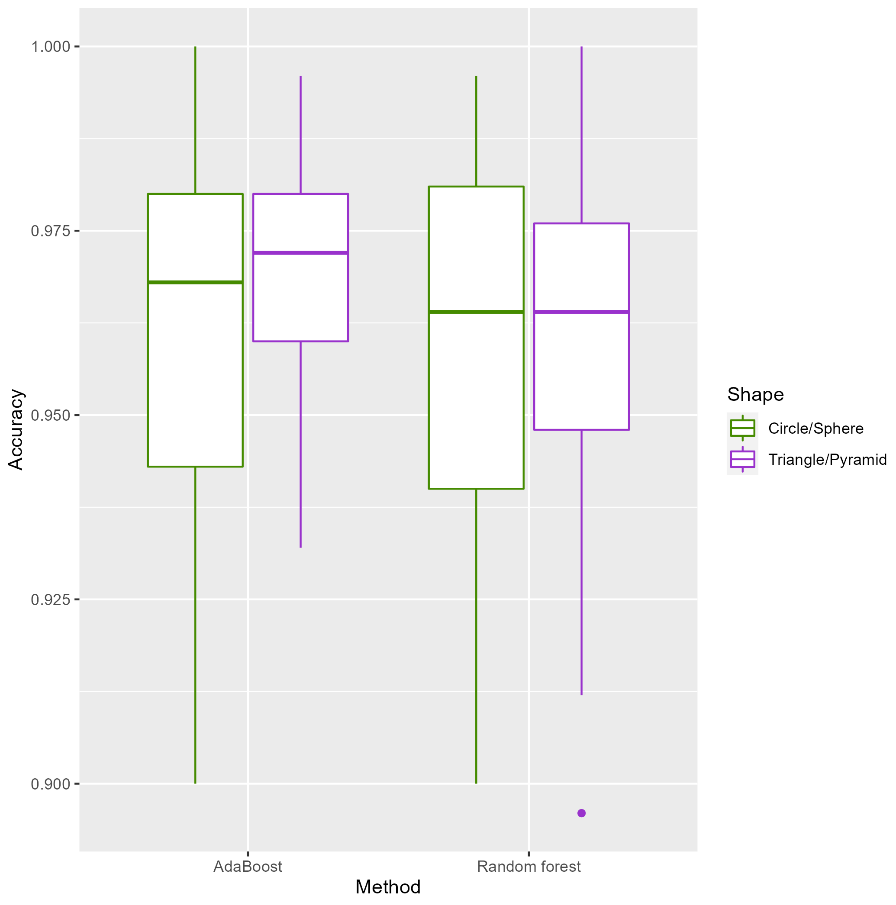

Figure 8: Box plots showing the effect of shape on accuracy for both methods on data sets with two and three nonredundant predictors.

4.4 Impact of predictor dimension

In table 2, we see that as the number of predictors grows, the accuracies drop for both methods. On these data sets, AdaBoost clearly outperforms random forest on a majority of the data sets.

4.5 Impact of noise

Since we have chosen $\epsilon = 0.2$, a perfect classifier would on average have an accuracy of 0.8 on the noisy data sets. Compared to the performance on data sets without noise, the performance drop is therefore at an expected level, although the performance of AdaBoost is more adversely impacted than that of random forest, with two, three and five nonredundant predictors. This can be seen in Figure 9, where we have averaged the accuracy on neutral and noisy data sets. Here, the advantage of random forest on noisy data sets is evident, although it is modest. Random forest performs significantly better compared to AdaBoost on all data sets with noise except for one. In higher predictor dimensions, the advantage of random forests is diminished overall, although it still has a significantly higher accuracy on two out of three noisy data sets, as seen in Table 2. The relative strength of random forest to handle noise is still illustrated by the fact that AdaBoost performs significantly better than random forest on the majority of the other data sets with higher predictor dimension that do not contain noise. In figure 10, we display the performance of the methods on data sets with noise on data sets with 50 and 100 nonredundant predictors. We observe that the averaged accuracy over all noisy data sets now is similar for both methods.

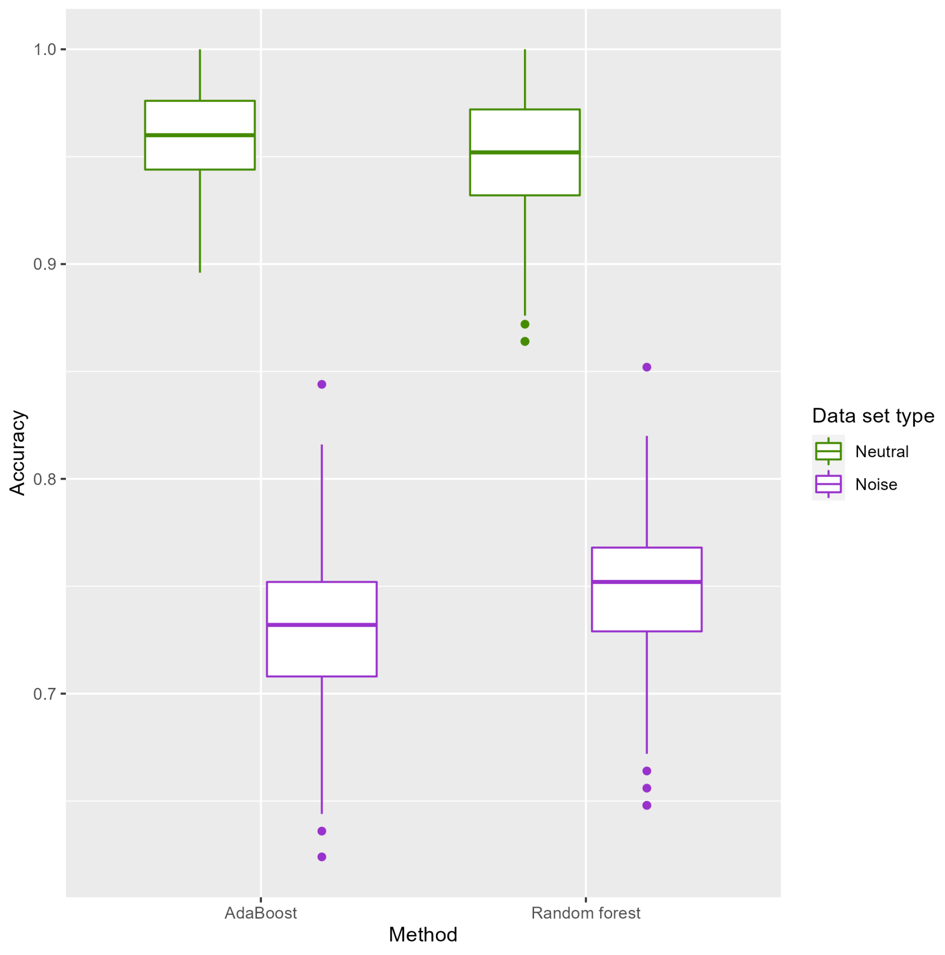

Figure 9: Box plots showing the effect of noise on accuracy for both methods on data sets with up to five nonredundant predictors.

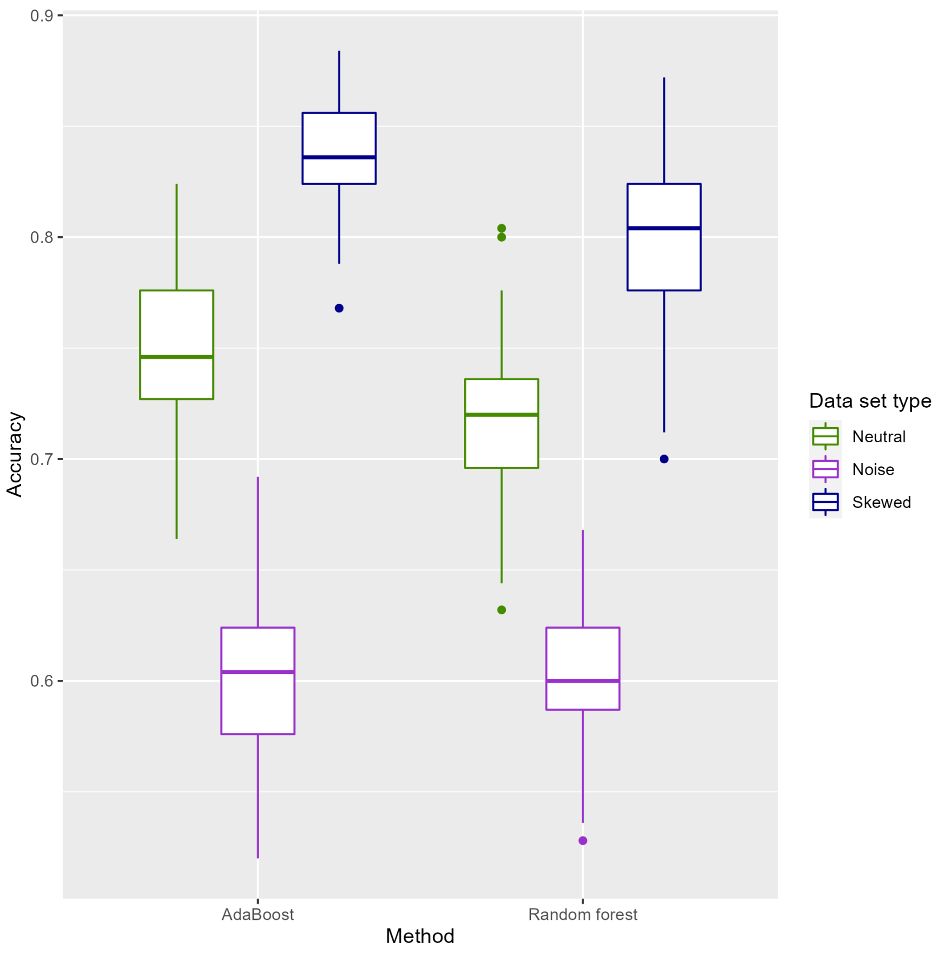

Figure 10: Box plots showing the effect of noise and skewed class distribution on accuracy for both methods on data sets with 50 and 100 nonredundant predictors.

4.6 Impact of skewed class distribution

The accuracy of both methods tend to improve slightly when the data sets have skewed compared to symmetric class distributions, regardless of type of data set. The gains in accuracy between symmetric and skewed class distributions vary between data sets but do not vary that much between methods. AdaBoost has a significantly higher accuracy compared to random forest on a majority of the skewed data sets, as illustrated in Figure 11 (low-dimensional data) and Figure 10 (high-dimensional data).

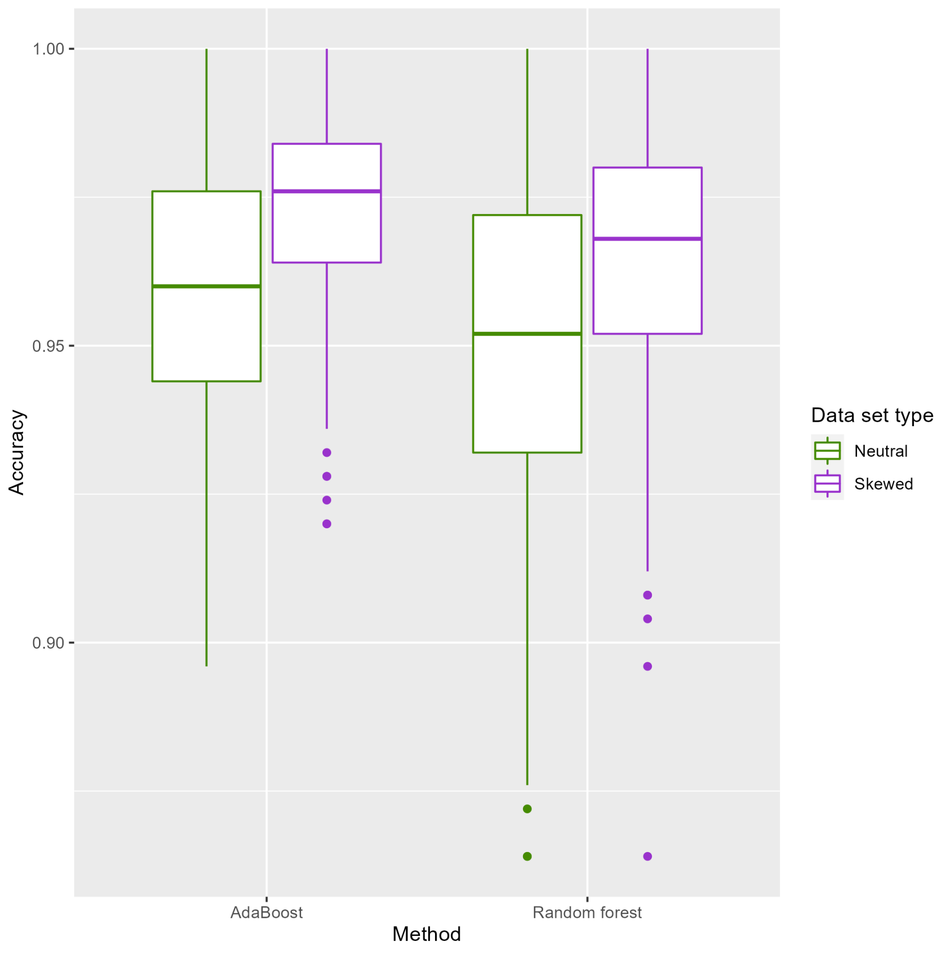

Figure 11: Box plots showing the effect of skewed class distribution on accuracy for both methods on data sets with up to five nonredundant predictors.

4.7 Impact of redundant predictors

The accuracy of both methods tend to drop as more redundant predictors are added. The performance of random forest is more sensitive to redundant predictors on data sets with two to five nonredundant predictors, as seen in Figure 12. Interestingly, in higher predictor dimensions this is reversed; on these data sets, the performance of AdaBoost is more heavily impacted by redundant predictors, as seen in figure 13. Another interesting observation is that when adding seven redundant predictors to a dataset with three nonredundant predictors, so that most predictors are redundant, the performance reduction is considerably larger on data sets with two pyramids compared to two spheres for both methods.

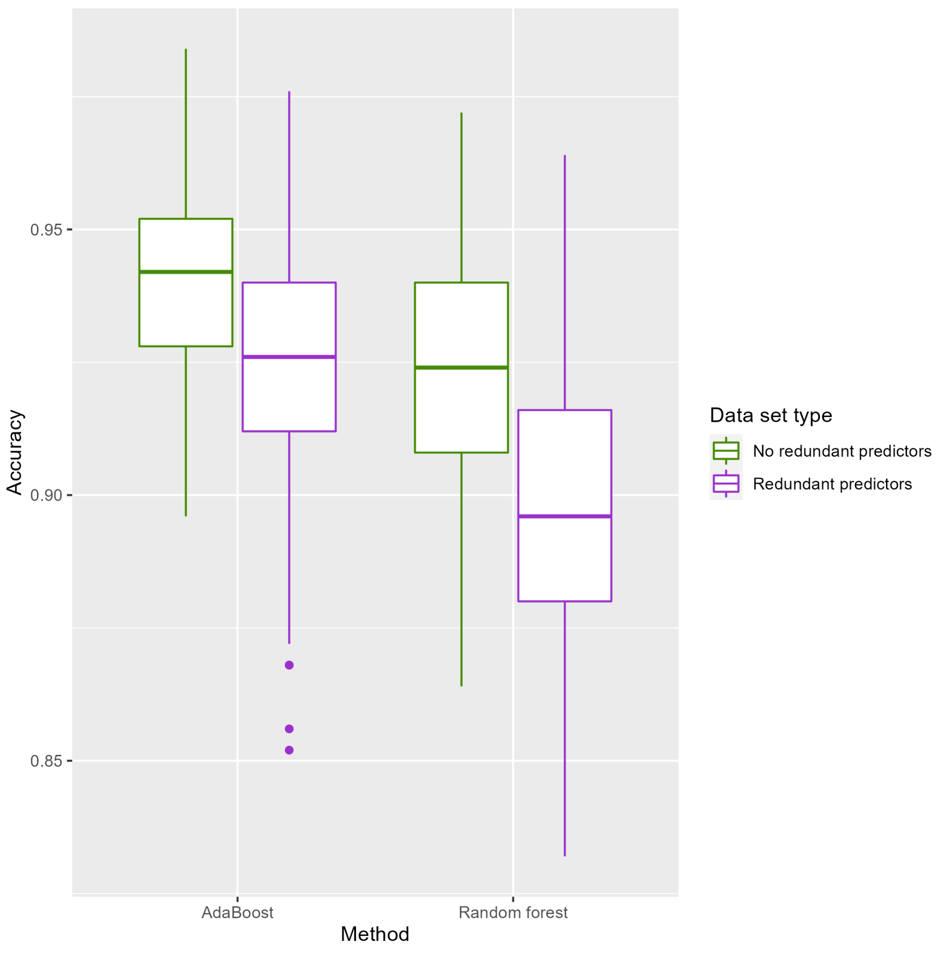

Figure 12: Box plots showing the effect of redundant predictors on data sets with two to five nonredundant predictors.

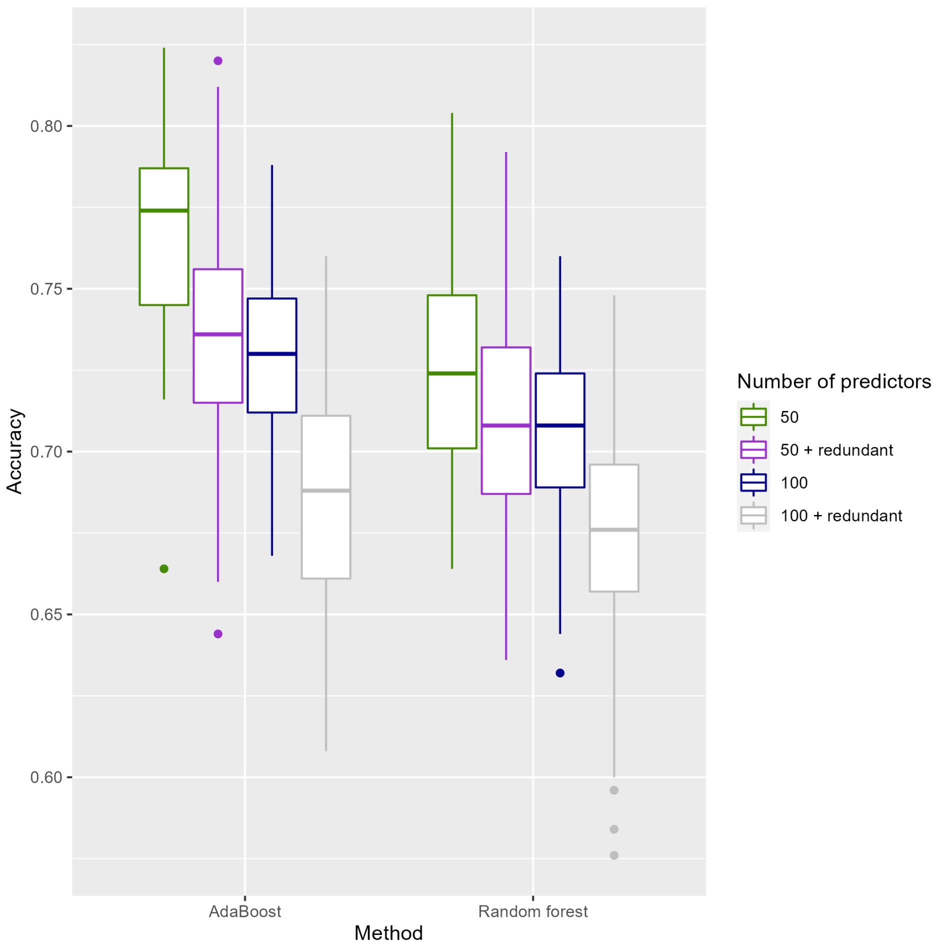

Figure 13: Box plots showing the effect of redundant predictors on data sets with 50 and 100 nonredundant predictors.

Table 1: Mean accuracy (proportion of correct classifications) and corresponding standard deviation on data sets with up to three nonredundant predictors. The accuracy is averaged over 50 test sets consisting of 250 observations. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80. The differences in accuracy are tested with two sample t-tests under the null hypothesis that the difference is zero. AdaBoost is trained with $B = 250$. Random forest is trained with $n_{min} = 1$ except for data sets with noise where n_{min} is set to 15. Whenever a method has significantly higher accuracy (at a 5 % significance level), the corresponding mean and standard deviations are given in bold.

Shape	p	**Noise**	**Skewed**	p_r	**AdaBoost**	**Random Forest**	**Difference**	
					M (SD)	M (SD)	t-statistic	p-value
One triangle	2	N	N	0	0.9795 (0.0077)	0.9790 (0.0102)	0.31	0.76
One circle	2	N	N	0	0.9814 (0.0088)	0.9802 (0.0080)	0.67	0.51
One triangle	2	Y	N	0	0.7590 (0.0272)	**0.7777 (0.0250)**	-3.57	<0.001
One circle	2	Y	N	0	0.7550 (0.0275)	**0.7678 (0.0276)**	-2.31	0.023
One triangle	2	N	Y	0	**0.9849 (0.0074)**	0.9801 (0.0087)	2.96	0.00381
One circle	2	N	Y	0	**0.9864 (0.0080)**	0.9821 (0.0074)	2.79	0.00629
Two triangles	2	N	N	0	**0.9692 (0.0103**	0.9633 (0.0121)	2.64	0.00955
Two circles	2	N	N	0	0.9724 (0.0102)	0.9690 (0.0106)	1.65	0.11
Two triangles	2	Y	N	0	0.7270 (0.0269)	**0.7412 (0.0297)**	-2.51	0.014
Two circles	2	Y	N	0	0.7495 (0.0265)	0.7582 (0.0255)	-1.67	0.097
Two triangles	2	N	Y	0	0.9785 (0.0099)	0.9753 (0.0116)	1.49	0.14
Two circles	2	N	Y	0	0.9801 (0.0092)	0.9766 (0.0089)	1.90	0.061
One pyramid	3	N	N	0	**0.9592 (0.0112)**	0.9452 (0.0141)	5.50	<0.001
One sphere	3	N	N	0	0.9406 (0.0168)	0.9397 (0.0160)	0.27	0.79
One pyramid	3	Y	N	0	0.7276 (0.0247)	**0.7463 (0.0265)**	-3.65	<0.001
One sphere	3	Y	N	0	0.7322 (0.0303)	**0.7552 (0.0295)**	-3.84	<0.001
One pyramid	3	N	Y	0	**0.9694 (0.0116)**	0.9545 (0.0135)	5.93	<0.001
One sphere	3	N	Y	0	**0.9769 (0.0108)**	0.9694 (0.0107)	3.45	<0.001
Two pyramids	3	N	N	0	**0.9370 (0.0163)**	0.9279 (0.0178)	2.72	0.00770
Two spheres	3	N	N	0	**0.9498 (0.0108)**	0.9382 (0.0134)	4.76	<0.001
Two pyramids	3	Y	N	0	0.7046 (0.0304)	**0.7255 (0.0279)**	-3.59	<0.001
Two spheres	3	Y	N	0	0.7143 (0.0269)	**0.7397 (0.0250)**	-4.88	<0.001
Two pyramids	3	N	Y	0	**0.9505 (0.0116)**	0.9417 (0.0125)	3.65	<0.001
Two spheres	3	N	Y	0	**0.9728 (0.0113)**	0.9665 (0.0137)	2.52	0.013
Two pyramids	6	N	N	3	**0.9155 (0.0150)**	0.9041 (0.0169)	3.58	<0.001
Two spheres	6	N	N	3	**0.9504 (0.0134)**	0.9333 (0.0148)	6.07	<0.001
Two pyramids	10	N	N	7	**0.8993 (0.0206)**	0.8790 (0.0226)	4.67	<0.001
Two spheres	10	N	N	7	**0.9455 (0.0142)**	0.9236 (0.0196)	6.42	<0.001

Table 2: Mean accuracy (proportion of correct classifications) and corresponding standard deviation on data sets with more than three nonredundant predictors. The accuracy is averaged over 50 test sets consisting of 250 observations. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80. The differences in accuracy are tested with two sample t-tests under the null hypothesis that the difference is zero. AdaBoost is trained with $B = 250$ when the number of predictors is less than 50 and $B = 1000$ when the number of predictors is 50 or greater. Random forest is trained with $n_{min} = 1$ when the number of predictors is less than 50 and $n_{min} = 20$ when the number of predictors is 50 or larger. Whenever a method has significantly higher accuracy (at a 5 % significance level), the corresponding mean and standard deviations are given in bold.

Shape	p	**Noise**	**Skewed**	p_r	**AdaBoost**	**Random Forest**	**Difference**	
					M (SD)	M (SD)	t-statistic	p-value
One sphere	5	N	N	0	**0.9371 (0.0150)**	0.8993 (0.0165)	12.00	<0.001
One sphere	5	Y	N	0	0.7093 (0.0302)	**0.7334 (0.0307)**	-3.97	<0.001
One sphere	5	N	Y	0	**0.9596 (0.0135)**	0.9321 (0.0177)	8.75	<0.001
One sphere	10	N	N	5	**0.9416 (0.0161)**	0.8880 (0.0222)	13.82	<0.001
One sphere	15	N	N	10	**0.9237 (0.0155)**	0.8867 (0.0219)	9.73	<0.001
One sphere	50	N	N	0	**0.7690 (0.0316)**	0.7269 (0.0325)	6.57	<0.001
One sphere	50	Y	N	0	**0.6247 (0.0247)**	0.6101 (0.0273)	2.81	0.00603
One sphere	50	N	Y	0	**0.8336 (0.0247)**	0.7798 (0.0277)	10.26	<0.001
One sphere	100	N	N	50	**0.7440 (0.0277)**	0.7146 (0.0323)	4.96	<0.001
One sphere	150	N	N	100	**0.7302 (0.0366)**	0.7042 (0.0329)	3.73	<0.001
One sphere	100	N	N	0	**0.7277 (0.0281)**	0.7058 (0.0288)	3.84	<0.001
One sphere	100	Y	N	0	0.5804 (0.0272)	**0.5979 (0.0303)**	-3.04	0.00300
One sphere	100	N	Y	0	**0.8408 (0.0213)**	0.8218 (0.0233)	4.25	<0.001
One sphere	150	N	N	50	0.7053 (0.0263)	0.6946 (0.0327)	1.80	0.074
One sphere	200	N	N	100	**0.6896 (0.0286)**	0.6786 (0.0324)	2.13	0.036
One sphere	300	N	N	200	0.6594 (0.0315)	0.6529 (0.0324)	1.03	0.31

4.8 Impact of class

In Table 8 of the Appendix, we can see the mean accuracy in classifying observations inside the shapes and outside the shapes for each method, for different types of data sets. It should be noted that with noise, there is an overlap such that both classes are represented inside and outside the shapes with probability ϵ. For ease of reasoning, we will nevertheless make the distinction between classes as inside and outside the shapes for noisy data sets as well.

When the number of predictors is small, the accuracies are similar for observations inside and outside the shapes for both methods, with two notable exceptions. The first exception is data sets with skewed class distribution. On these data sets, we see that both methods have a higher classification accuracy on observations lying outside the shapes. This accuracy discrepancy is the largest for random forest; 0.12 compared to 0.07 for AdaBoost. The second exception is the data sets with redundant predictors. On these data sets, random forest has a lower classification accuracy on observations lying inside the shapes whereas AdaBoost has similar performance.

In higher predictor dimensions ($p \geq 50$), the deterioration of classification accuracy on observations lying inside the shapes is heavily exacerbated for both methods on data sets with skewed class distribution. AdaBoost has a classification accuracy of 0.22 on observations lying inside the shapes whereas the corresponding value for random forest is 0.0017, i.e., both methods are performing worse than chance and random forest predicts the other class on nearly all of the test observations. In contrast, the accuracies on observations outside the shapes is 0.9882 and 1.0000 for AdaBoost and random forest respectively. Aside from this finding on data sets with skewed class distributions, the methods exhibit accuracy discrepancies that are each other's opposite; AdaBoost has higher accuracy on observations lying *inside* the shapes whereas random forest has higher accuracy on observations lying *outside* the shapes. This discrepancy is dramatic for AdaBoost and less pronounced for random forest.

4.9 Runtimes

Starting with one of the smallest data sets, one triangle and two predictors, the runtime is four seconds for random forest and 49 seconds for AdaBoost. Thus, already with the simplest data set it is evident that the training of AdaBoost takes more time (given its hyperparameter settings, which is $B = 250$ for these data sets). This discrepancy grows as the dimensionality of the predictor space grows and AdaBoost uses more weak learners, as shown in Table 3. For example, with 300 predictors, random forest has a runtime of 42 seconds and AdaBoost 5166 seconds (i.e., 86 minutes) - 123 times longer.

Table 3: Runtimes on data sets with high predictor dimensions. Number of predictors is denoted by p.

	Runtimes (s)	
p	AdaBoost	Random forest
100	1617	14
150	2335	20
200	3516	25
300	5166	42

5 Discussion

5.1 Main findings

In line with previous studies, random forest outperforms AdaBoost on noisy data sets while AdaBoost outperforms random forest on data sets with redundant predictors. In addition, except for the noisy data sets, AdaBoost tends to have an advantage overall. This advantage comes at quite a high cost computationally, as training and prediction takes vastly more time with the AdaBoost algorithm. The finding that AdaBoost tends to have higher accuracy while random forest is more computationally efficient has been observed previously (Miao & Heaton, 2010). Other studies have found that the methods have similar performance (Chan & Paelinckx, 2008; Tang et al., 2021).

As the predictor dimension grows, an interesting discrepancy emerges between the two methods. AdaBoost has superior classification accuracy on observations inside the shapes which contrasts starkly with observations outside the shapes, where instead random forest has superior accuracy.

The fact that AdaBoost takes longer time to run can be understood from how the algorithms are trained. In random forest, the training of trees can be parallelized (Tang et al., 2021), which is not possible in AdaBoost since the trees are grown sequentially, as shown in Algorithm 1. Thus, it should not come as a surprise that the runtime of random forest is shorter and less affected by predictor dimension compared to AdaBoost, as seen in Table 3.

5.2 Hyperparameter settings and model fit

As mentioned above, $B = 1000$ leads to worse performance compared to lower values of B for AdaBoost on data sets with two and three predictors. The most probable reason for this is that with 1000 weak learners, AdaBoost starts to overfit to the training data sets with two and three nonredundant predictors (in other words, its model complexity is too high). Indeed, the tendency of AdaBoost to overfit to low-dimensional data has been noted in previous simulation studies (Mease & Wyner, 2008, pp. 136). However, with a larger predictor dimension it is reasonable that more weak learners are needed to capture the patterns in the data. When training AdaBoost with $B = 50$, the performance is often worse than the other values regardless of type of data set, apart from a few exceptions. This is likely due to underfitting; having just 50 weak learners is not enough to capture the patterns in the training data even when the predictor dimension is small.

It should be noted that there has been a debate as to how AdaBoost should be trained. Some researchers argue that AdaBoost should be trained with early stopping for best results while others claim that AdaBoost should be trained with a large number of weak learners without any regularization since it has been observed that AdaBoost rarely overfits if the predictor dimension is large enough (Wyner et al., 2017). As mentioned previously, lately there has also been a debate as to whether a base learner needs to be weak (i.e., of low model complexity). In the past decade, we have seen successful implementations of AdaBoost with random forest as base learner (El Hamdaoui et al., 2021; Nayak et al., 2016; Rohan et al., 2019), which certainly challenges the validity of this assumption.

For random forest, the impact of different values of n_{min} is not as straightforward to analyze and understand. For noisy data, the performance is improved by having a larger node size in the trees, i.e. more shallow trees. This is to be expected since if we have more noise in the data, the averaging in each tree helps in mitigating the impact of noise. If we have just one training observation in each terminal node, the predictions get more sensitive to noise as the training observation in question can be a noisy observation. For data sets with a small number of predictors, $n_{min} = 1$ outperformed higher values in a lot of cases while increasing n_{min} to 20 proved to be better for data sets with a higher number of predictors. Since higher values of n_{min} creates shallower trees, leading to base learners with lower model complexity compared to a model with $n_{min} = 1$, it is somewhat surprising that $n_{min} = 1$ leads to better performance on data sets with a small number of predictors and $n_{min} = 20$ leads to better performance on data sets with a large number of predictors, as one would think that the former scenario could lead to overfitting and the latter scenario could lead to underfitting. One possible explanation is that the default value $n_{min} = 1$ actually is the best choice for all data sets, and it is the hyperparameter q that controls the number of predictors available as splitting variables in each split that should be optimized when the number of predictors is high. This can especially be the case when the data sets contain redundant predictors, as this has been noted in previous studies (Hastie et al., 2017, pp. 596).

5.3 Noisy data

The performance of AdaBoost is more negatively impacted by the presence of noise in the data sets, which is expected given previous findings (Dietterich, 2000, pp. 147; Li et al., 2017, pp. 5; Long & Servedio, 2008, pp. 1).

5.4 Class distributions

Interestingly, the accuracy on data sets with skewed class distributions are higher compared to data sets with symmetric class distributions for both methods. At first glance, this result is somewhat surprising. One possible explanation is that a larger proportion of the observations are lying outside the shapes in data sets with skewed class distributions, and that these observations are easy for the methods to correctly classify. To clarify our reasoning, remember that the class distribution is controlled by altering the size of the shape enclosing one of the classes. We can imagine that the methods could, due to the construction of regions parallel to the coordinate axes in decision trees, easily approximate a circle/sphere or triangle/pyramid by a square/hypercube. If the shape in question is small, it can be approximated by a smaller square/hypercube which leads to a lower error, since there are fewer observations lying between the boundary of the shape and the boundary of the approximated square/hypercube.

We can also note that in terms of overall classification accuracy, the classification problem gets simpler as the class distribution gets more skewed if we make a best guess solely based on knowledge about the class distribution. Suppose that the probabilities for the classes 0 and 1 are $1 - P$ and P, respectively. We can estimate P from the training data in advance and make the predictions by randomly guessing $Y = 1$ with probability P regardless of $\boldsymbol{X}$. This corresponds to $f(\boldsymbol{X}) = P$. The expected accuracy for random guessing is

$$P(Y=0)P(\text{correct prediction}|Y=0) + P(Y=1)P(\text{correct prediction}|Y=1)$$

$$= (1-P)(1-P) + P \cdot P = 1 - 2P + 2P^2 = 1 - 2P(1-P).$$

This expression grows as P gets closer to 0 or 1, i.e., as the class distribution gets more skewed. If we let $P = 0.2$ and make the prediction by random guessing as described above, this yields the expected accuracy

$$1 - 2 \cdot 0.2 \cdot (1 - 0.2) = 1 - 2 \cdot 0.2 \cdot 0.8 = 0.68.$$

An even class distribution corresponds to $P = 0.5$. This results in a lower expected accuracy if we use the same random guessing strategy:

$$1 - 2 \cdot 0.5 \cdot (1 - 0.5) = 1 - 2 \cdot 0.5 \cdot 0.5 = 0.5.$$

A further analysis of classification accuracy for each class shows that it is indeed easier to classify the observations outside the shapes on data sets with skewed class distributions (i.e., smaller shapes). In high predictor dimensions, the performance is worse than chance for both methods and the performance of random forest stands out as particularly poor. One probable explanation for this is the ratio of the number of predictors to the number of observations inside the shapes. For example, with 100 predictors, this ratio is about 1:2. Thus, it is quite understandable that the methods are not able to capture the relationships in the data as more observations are needed to explore the predictor space. In summary, one should be cautious about drawing conclusions about the performance of the methods on skewed data sets due to this observation.

5.5 Redundant predictors

The performance of random forest is more adversely impacted than that of AdaBoost by the presence of redundant predictors, which is in line with previous findings (Kubus, 2018, pp. 1; Nguyen et al., 2015, pp. 4). Interestingly, when the number of predictors is higher, the performance of random forest seems to be more robust against the influence of redundant predictors compared to AdaBoost. The robustness of random forest against nonredundant predictors when the number of predictors is fairly high has been noted in earlier studies (Hastie et al., 2017, pp. 596).

5.6 Predictor dimension

With a higher number of predictors and the same number of observations as all the other data sets, the aforementioned discrepancy in classification accuracy appears. The observed discrepancies in classification accuracy between the two classes for each method makes the overall classification accuracy a somewhat misguiding evaluation metric, as it does not unveil the full picture.

One possible, albeit speculative, explanation is that the averaging of several models in random forest makes it easier to capture cruder structures in the data whereas the sequential step-wise error reduction in AdaBoost makes it easier to capture finer structures. If we for a moment suppose that this explanation is true, the finding in this thesis can have important implications for method choice in other classification settings. These implications can be summarized as follows. If one is more concerned with capturing a crude structure in the data, random forest could be better suited for the problem. If one instead wants to capture finer structures in the data, AdaBoost could be a wiser choice. However, it is also important to note that it cannot be ruled out that hyperparameter settings have influenced the observed discrepancies in this thesis.

5.7 Runtimes

For both methods, the runtime increases as the dimensionality of the predictor space grows. Furthermore, the number of weak learners B has a dramatic impact on runtime for AdaBoost. This is obviously a disadvantage considering the energy and time consumption as well as the computational cost. It should be noted that runtimes depend on hardware, software as well as implementation. All simulation and modeling is conducted on an inexpensive PC laptop from 2016, which certainly contributes to the long runtimes in this thesis.

5.8 Suggestions for improvements and further studies

To facilitate a more fair comparison between AdaBoost and random forest, hyperparameter optimization can be made more rigorous, for instance by exploring different values for the hyperparameter q that represents the number of predictors to consider in each split in random forest. One can attempt to optimize q and n_{min} simultaneously with a *grid search*, where different pairs of values for these hyperparameters are validated (Lindholm et al., 2022, pp. 129). Further hyperparameter optimization is particularly relevant for data sets with redundant predictors, since setting q too low leads to a low probability of selecting relevant predictors if the number of redundant predictors is large (Hastie et al., 2017, pp. 596). Furthermore, previous findings indicate that lower values of q tend to improve performance on noisy data (Mentch & Zhou, 2020, pp. 16).

In order to further optimize the performance of AdaBoost, one can use regularization techniques such as early stopping (Hastie et al., 2017, pp. 365) to find a suitable number of weak learners B.

The runtimes of both methods differ greatly. One could argue that the comparison made in this thesis is not entirely fair if the runtimes are taken into account. Therefore, it would be interesting to compare the methods with hyperparameter settings that lead to similar runtimes for both methods. In particular, this would mean training AdaBoost with a fewer number of weak learners B. However, as pointed out above, random forest training can be parallelized in contrast to AdaBoost which makes it unlikely that the runtime of AdaBoost could match that of random forest without making a considerable sacrifice of accuracy for AdaBoost.

Furthermore, future studies investigating predictive performance on data sets with skewed class distributions should ensure that the number of observations in larger predictor dimensions is large enough so that the minority class contains a sufficient number of observations. Moreover, the class distribution could also be made more skewed than 20/80 to further elucidate performance differences between AdaBoost and random forest.

6 Conclusion

The aim of this thesis was to investigate and compare the predictive performance of AdaBoost and random forest, two popular machine learning algorithms for classification. Overall, we found that the two methods have similar performance, but some differences emerged. We showed that random forest had a higher accuracy on noisy data sets, while AdaBoost had a higher accuracy on data sets with skewed class distribution and redundant predictors. Moreover, AdaBoost tended to outperform random forest on data sets with larger predictor dimension. AdaBoost took a considerably longer time to run compared to random forest. These observations are in line with previous studies. However, in higher predictor dimensions, random forest had a notably higher classification accuracy than AdaBoost on one class whereas AdaBoost had an even bigger advantage compared to its counterpart on the other class, making AdaBoost the superior classifier overall. While further investigation of the cause of this discrepancy was out of scope of this thesis, we hypothesized that it could be due to how the models are trained. We pointed out that if this hypothesis is true, it could indicate that AdaBoost has an advantage over random forest in detecting finer structures in data while random forest has an advantage in detection of cruder structures. On data sets with larger predictor dimension and skewed class distribution, both methods had poor prediction accuracy on observations of the smaller class. One probable explanation to the poor accuracies is that the number of observations in the smaller class was too small.

7 References

Basetti, A. (2019). A comparison of logistic regression and neural networks for binary classification problems. Stockholm University.
Retrieved from https://www.math.su.se/publikationer/uppsatsarkiv/.

Breiman, L. (1996). Bias, variance, and arcing classifiers.
Retrieved from https://www.stat.berkeley.edu/ breiman/arcall96.pdf

Breiman, L. (2001). Random forests. *Machine learning*, 45, 5-32.
Retrieved from https://www.stat.berkeley.edu/ breiman/randomforest2001.pdf

Chan, J. C. W., & Paelinckx, D. (2008). Evaluation of random forest and Adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery. *Remote Sensing of Environment*, 112(6), 2999-3011.
Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0034425708000679

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40, 139-157.
Retrieved from https://link.springer.com/article/10.1023/A:1007607513941

Dudoit, S., & Fridlyand, J. (2003). Classification in microarray experiments. *Statistical analysis of gene expression microarray data*, 1, 93-158.
Retrieved from https://www.stat.berkeley.edu/ sandrine/Teaching/CMPBIO201.F13/DudoitFridlyandCRC03.pdf

El Hamdaoui, H., Boujraf, S., El Houda Chaoui, N., Alami, B., & Maaroufi, M. (2021). Improving heart disease prediction using random forest and AdaBoost Algorithms. *International Journal of Online & Biomedical Engineering*, 17(11).
Retrieved from https://pdfs.semanticscholar.org/2797/98203bdadd072430a57fec226e69714dc75d.pdf

Gut, A. (2009). *An Intermediate Course in Probability*. New York: Springer.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2017). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 12, pp. 1-764). New York: Springer.

Held, L., & Sabanés Bové, D. (2014). *Applied statistical inference*. Berlin Heidelberg: Springer, 10(978-3), 16.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. New York: Springer.

Kubus, M. (2018). The problem of redundant variables in random forests. Acta Universitatis Lodziensis. *Folia Oeconomica*, 6(339), 7-16.
Retrieved from https://www.researchgate.net/publication/331123315_The_Problem_of _Redundant_Variables_in_Random_Forests

Li, J., Liu, L., Liu, J., & Green, R. (2017). Building diversified multiple trees for classification in high dimensional noisy biomedical data. *Health Information Science and Systems*, 5, 1-10.
Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5634968/pdf/13755_2017_Article_25.pdf

Lindholm, A., Wahlström, N., Lindsten, F., & Schön, T. B. (2022). *Machine learning: A First Course for Engineers and Scientists*. Cambridge University Press.

Liaw, M. A. (2018). Package 'randomforest'. University of California, Berkeley: Berkeley, CA, USA.
Retrieved from https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

Long, P. M., & Servedio, R. A. (2008). Random classification noise defeats all convex potential boosters. In Proceedings of the 25th international conference on Machine learning (pp. 608-615).
Retrieved from http://www.machinelearning.org/archive/icml2008/papers/258.pdf

Mease, D., & Wyner, A. (2008). Evidence Contrary to the Statistical View of Boosting. *Journal of Machine Learning Research*, 9(2).
Retrieved from https://www.jmlr.org/papers/volume9/mease08a/mease08a.pdf

Mentch, L., & Zhou, S. (2020). Randomization as regularization: A degrees of freedom explanation for random forest success. *Journal of Machine Learning Research*, 21(171), 1-36.
Retrieved from https://jmlr.csail.mit.edu/papers/volume21/19-905/19-905.pdf

Miao, X., & Heaton, J. S. (2010). A comparison of random forest and Adaboost tree in ecosystem classification in east Mojave Desert. In 2010 18th International Conference on Geoinformatics (pp. 1-6). IEEE.
Retrieved from https://ieeexplore.ieee.org/document/5567504

Månsson, K. N., Frick, A., Boraxbekk, C. J., Marquand, A. F., Williams, S. C. R., Carlbring, P., ... & Furmark, T. (2015). Predicting long-term outcome of Internet-delivered cognitive behavior therapy for social anxiety disorder using fMRI and support vector machine learning. *Translational psychiatry*, 5(3), e530-e530.

Nayak, D. R., Dash, R., & Majhi, B. (2016). Brain MR image classification using two-dimensional discrete wavelet transform and AdaBoost with random forests. *Neurocomputing*, 177, 188-197.
Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0925231215017531

Nguyen, T. T., Huang, J. Z., & Nguyen, T. T. (2015). Unbiased feature selection in learning random forests for high-dimensional data. *The Scientific World Journal*, 2015.
Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4387916/pdf/TSWJ2015-471371.pdf

Pandiri, D. K., Murugan, R., & Goel, T. (2024). Smart soil image classification system using lightweight convolutional neural network. *Expert Systems with Applications*, 238, 122185.
Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0957417423026878

Rohan, T. I., Siddik, A. B., Islam, M., & Yusuf, M. S. U. (2019, July). A precise breast cancer detection approach using ensemble of random forest with AdaBoost. In 2019 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2) (pp. 1-4). IEEE.
Retrieved from https://ieeexplore.ieee.org/document/9036697

Tang, J., Henderson, A., & Gardner, P. (2021). Exploring AdaBoost and random Forests machine learning approaches for infrared pathology on unbalanced data sets. *Analyst*, 146(19), 5880-5891.
Retrieved from https://pubs.rsc.org/en/content/articlepdf/2021/an/d0an02155e

Wyner, A. J., Olson, M., Bleich, J., & Mease, D. (2017). Explaining the success of adaboost and random forests as interpolating classifiers. *Journal of Machine Learning Research*, 18(48), 1-33.
Retrieved from https://www.jmlr.org/papers/volume18/15-240/15-240.pdf

Zhu, T. (2020). Analysis on the applicability of the random forest. In *Journal of Physics: Conference Series* (Vol. 1607, No. 1, p. 012123). IOP Publishing.
Retrieved from https://iopscience.iop.org/article/10.1088/1742-6596/1607/1/012123

A Appendix

A.1 Hyperparameter settings

In Table 4 and 5, we display the accuracy for AdaBoost for different number of weak learners B. In table 6 and 7, we provide corresponding results for random forest and different values of the minimum node size n_{min}.

Table 4: Mean accuracies for AdaBoost on data sets with two and three nonredundant predictors for different numbers of weak learners B. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80.

					Accuracy				
Shape	p	**Noise**	**Skewed**	p_r	$B = 50$	$B = 100$	$B = 250$	$B = 500$	$B = 1000$
One triangle	2	N	N	0	**0.9822**	0.979	0.9794	0.9795	0.9798
One circle	2	N	N	0	0.9804	0.981	0.9806	**0.9814**	0.981
One triangle	2	Y	N	0	0.7715	**0.7734**	0.7591	0.7514	0.7382
One circle	2	Y	N	0	**0.7704**	0.764	0.7568	0.7402	0.728
One triangle	2	N	Y	0	0.9811	0.9847	**0.9856**	0.9854	0.985
One circle	2	N	Y	0	0.9813	0.9858	0.9854	0.9862	**0.9865**
Two triangles	2	N	N	0	0.9532	0.963	0.9666	**0.9679**	0.9666
Two circles	2	N	N	0	0.9681	0.9698	**0.973**	0.9714	0.9713
Two triangles	2	Y	N	0	0.7332	**0.734**	0.7263	0.7158	0.7094
Two circles	2	Y	N	0	**0.7564**	**0.7564**	0.751	0.741	0.7344
Two triangles	2	N	Y	0	0.9778	0.9779	**0.9782**	0.9781	0.9766
Two circles	2	N	Y	0	0.98	0.9799	0.9787	**0.9801**	0.9785
One pyramid	3	N	N	0	0.9518	0.9593	**0.96**	0.9586	0.9573
One sphere	3	N	N	0	**0.9367**	0.928	0.9316	0.9235	0.9354
One pyramid	3	Y	N	0	**0.7498**	0.7374	0.727	0.7126	0.7071
One sphere	3	Y	N	0	**0.7495**	0.7436	0.7367	0.7252	0.7124
One pyramid	3	N	Y	0	0.9635	**0.9699**	0.9698	**0.9692**	0.9662
One sphere	3	N	Y	0	0.9732	**0.9776**	0.9778	0.9774	0.9771
Two pyramids	3	N	N	0	0.9197	0.9328	0.9389	**0.9402**	0.9393
Two spheres	3	N	N	0	0.9454	0.9492	**0.9515**	0.9492	0.9485
Two pyramids	3	Y	N	0	**0.7161**	0.7132	0.7028	0.6939	0.6829
Two spheres	3	Y	N	0	**0.7342**	0.7336	0.723	0.7056	0.6964
Two pyramids	3	N	Y	0	0.9421	0.9494	0.9518	**0.9546**	0.9543
Two spheres	3	N	Y	0	0.9691	0.9702	**0.973**	0.9724	0.9715
Two pyramids	10	N	N	7	0.8882	0.8981	**0.9016**	0.8975	0.8996
Two spheres	10	N	N	7	0.9434	0.9474	**0.9484**	0.9439	0.9406

Table 5: Mean accuracies for AdaBoost on data sets with more than three nonredundant predictors for different numbers of weak learners B. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80.

					Accuracy				
Shape	p	**Noise**	**Skewed**	p_r	B **= 50**	B **= 100**	B **= 250**	B **= 500**	B **= 1000**
One sphere	5	N	N	0	0.9246	0.9347	0.9379	**0.9386**	0.9373
One sphere	5	Y	N	0	**0.72**	0.7132	0.6968	0.6908	0.6828
One sphere	5	N	Y	0	0.9509	0.954	0.957	0.9558	**0.9576**
One sphere	10	N	N	5	0.9184	0.9322	**0.9388**	**0.9388**	0.935
One sphere	50	N	N	0	0.6941	0.7164	0.7614	0.7673	**0.7901**
One sphere	50	Y	N	0	0.5922	0.6038	0.6157	0.6278	**0.6256**
One sphere	50	N	Y	0	0.7962	0.813	0.8324	0.8461	**0.8462**
One sphere	100	N	N	50	0.6734	0.6969	0.7306	0.746	**0.7615**
One sphere	150	N	N	100	0.6561	0.6791	0.7134	0.7242	**0.7331**
One sphere	100	N	N	0	0.6351	0.6574	0.691	0.7094	**0.7266**
One sphere	100	Y	N	0	0.5564	0.5694	0.5806	0.5903	**0.5987**
One sphere	100	N	Y	0	0.8227	0.8255	0.8323	0.839	**0.8429**
One sphere	120	N	N	20	0.6307	0.6574	0.6845	0.702	**0.7096**

Table 6: Mean accuracies for random forest on data sets with two and three nonredundant predictors for different values of the minimum node size n_{min}. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80.

Shape	p	**Noise**	**Skewed**	p_r	**Accuracy**					
					$n_{min} = 1$	$n_{min} = 5$	$n_{min} = 10$	$n_{min} = 15$	$n_{min} = 20$	$n_{min} = 30$
One triangle	2	N	N	0	**0.98**	0.9774	0.9752	0.9742	0.9732	0.9702
One circle	2	N	N	0	**0.9804**	0.9776	0.976	0.9712	0.961	0.9466
One triangle	2	Y	N	0	0.7458	0.7591	0.7725	0.7743	**0.7778**	0.7742
One circle	2	Y	N	0	0.7404	0.7525	0.7631	0.7663	**0.7692**	0.7682
One triangle	2	N	Y	0	**0.9794**	0.9791	0.9772	0.9748	0.9728	0.9622
One circle	2	N	Y	0	**0.9812**	0.981	0.9761	0.974	0.9722	0.969
Two triangles	2	N	N	0	**0.9646**	0.9613	0.9578	0.9513	0.9486	0.9336
Two circles	2	N	N	0	0.9682	0.9661	0.9652	0.9625	0.9578	0.9533
Two triangles	2	Y	N	0	0.7178	0.7274	0.7402	**0.7414**	0.74	0.7343
Two circles	2	Y	N	0	0.7478	0.7511	0.7575	**0.7587**	0.7569	0.7566
Two triangles	2	N	Y	0	**0.9766**	0.9728	0.965	0.9623	0.9567	0.946
Two circles	2	N	Y	0	**0.977**	0.9762	0.9735	0.9705	0.9686	0.9681
One triangle	3	N	N	0	**0.945**	0.9411	0.9376	0.933	0.9302	0.9174
One circle	3	N	N	0	**0.938**	0.9375	0.9344	0.9315	0.9281	0.924
One triangle	3	Y	N	0	0.7334	0.741	0.7413	0.7469	**0.7505**	0.7482
One circle	3	Y	N	0	0.7425	0.75	0.7535	**0.7565**	0.7547	0.7523
One triangle	3	N	Y	0	**0.9562**	0.9526	0.9467	0.9416	0.9382	0.9315
One circle	3	N	Y	0	**0.9694**	0.968	0.9654	0.962	0.9614	0.9589
Two triangles	3	N	N	0	**0.9262**	0.9222	0.9132	0.909	0.9053	0.899
Two circles	3	N	N	0	**0.9382**	0.9366	0.9353	0.9305	0.9254	0.9201
Two triangles	3	Y	N	0	0.7145	0.7163	0.7219	0.7238	0.7231	**0.7242**
Two circles	3	Y	N	0	0.7285	0.7326	0.7376	0.7402	**0.7409**	0.7398
Two triangles	3	N	Y	0	**0.9438**	0.9389	0.9352	0.9322	0.9273	0.9194
Two circles	3	N	Y	0	**0.9664**	0.9649	0.9628	0.9598	0.957	0.9503
Two triangles	10	N	N	7	**0.8756**	0.8727	0.874	0.8694	0.8666	0.863
Two circles	10	N	N	7	**0.9176**	0.9155	0.9154	0.911	0.9112	0.9046

Table 7: Mean accuracies for random forest on data sets with more than three nonredundant predictors for different values of the minimum node size n_{min}. The number of predictors is denoted by p and number of redundant predictors p_r. Noise = Y corresponds to $\epsilon = 0.2$. Skewed = Y corresponds to a class distribution of 20/80.

Shape	p	**Noise**	**Skewed**	p_r	**Accuracy**					
					$n_{min} = 1$	$n_{min} = 5$	$n_{min} = 10$	$n_{min} = 15$	$n_{min} = 20$	$n_{min} = 30$
One sphere	5	N	N	0	**0.8958**	0.8938	0.8885	0.8831	0.8786	0.8742
One sphere	5	Y	N	0	0.7126	0.7184	0.7207	0.7278	0.7261	**0.7289**
One sphere	5	N	Y	0	**0.9253**	0.9234	0.9224	0.9215	0.9193	0.9163
One sphere	10	N	N	5	0.9	**0.9012**	0.8979	0.895	0.8884	0.8859
One sphere	50	N	N	0	0.7008	0.7045	0.7264	0.7447	0.7454	**0.7465**
One sphere	50	Y	N	0	0.6174	0.6128	0.6201	**0.621**	0.6189	0.6199
One sphere	50	N	Y	0	0.7818	**0.7821**	0.7817	0.7818	0.7814	0.7815
One sphere	100	N	N	50	0.7082	0.7094	0.7149	0.7236	**0.7245**	0.7046
One sphere	150	N	N	100	0.6954	0.6892	0.7006	0.715	0.7232	**0.7252**
One sphere	100	N	N	0	0.6702	0.6714	0.6767	0.6892	**0.691**	0.6894
One sphere	100	Y	N	0	0.5864	0.5883	0.5863	0.6045	0.5992	0.5962
One sphere	100	N	Y	0	0.8224	0.8212	0.8104	0.8341	0.8274	0.8263
One sphere	120	N	N	20	0.6683	0.6631	0.679	0.6846	**0.6935**	**0.6874**

A.2 Accuracy per class

Table 8 shows mean accuracy separated by class for each type of data set and method.

Table 8: Mean accuracies separated by class for different types of data sets.

Accuracy					
		Inside shape		**Outside shape**	
Data set type	p	**AdaBoost**	**Random forest**	**AdaBoost**	**Random forest**
Neutral, triangle/pyramid	2 & 3	0.9689	0.9588	0.9699	0.9658
Neutral, circle/sphere	2 & 3	0.9615	0.9593	0.9605	0.9607
Neutral, one shape	2 & 3	0.9652	0.959	0.9652	0.9633
Neutral, Two shapes	2 & 3	0.9563	0.95	0.9576	0.9483
Neutral	2, 3 & 5	0.9571	0.9458	0.9597	0.9521
Noise	2, 3 & 5	0.7327	0.7408	0.7284	0.757
Skewed	2, 3 & 5	0.9149	0.8646	**0.9874**	**0.9884**
Redundant predictors	2, 3 & 5	0.9208	0.8737	0.929	**0.9185**
Neutral	50 & 100	**0.8198**	0.6874	0.668	**0.749**
Noise	50 & 100	**0.6358**	0.5256	0.5689	**0.6873**
Skewed	50 & 100	0.2228	0.0017	**0.9882**	**1**
Neutral	50	**0.8399**	0.7022	0.688	**0.7561**
Redundant predictors	50	**0.8046**	0.7076	0.6633	0.7116
Neutral	100	**0.7998**	0.6725	0.648	**0.7418**
Redundant predictors	100	**0.7429**	0.6481	0.6212	**0.705**