Advances in Random Forests with Application to Classification

by

Arnu Pretorius

Thesis presented in partial fulfilment of the requirements for the degree of Master of Commerce in Mathematical Statistics in the Faculty of Economic and Management Sciences at Stellenbosch University

Supervisor: Dr. S. Bierman
Co-supervisor: Prof. S.J. Steel

December 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Abstract

Advances in Random Forests with Application to Classification

A. Pretorius

Thesis: MComm (Mathematical Statistics)

December 2016

Since their introduction, random forests have successfully been employed in a vast array of application areas. Fairly recently, a number of algorithms that adhere to Leo Breiman's definition of a random forest have been proposed in the literature. Breiman's popular random forest algorithm (Forest-RI), and related ensemble classification algorithms which followed, form the focus of this study. A review of random forest algorithms that were developed since the introduction of Forest-RI is given. This includes a novel taxonomy of random forest classification algorithms, which is based on their sources of randomisation, and on deterministic modifications. Also, a visual conceptualisation of contributions to random forest algorithms in the literature is provided by means of multidimensional scaling.

Towards an analysis of advances in random forest algorithms, decomposition of the expected prediction error into bias and variance components is considered. In classification, such decompositions are not as straightforward as in the case of using squared-error loss for regression. Hence various definitions of bias and variance for classification can be found in the literature. Using a particular bias-variance decomposition, an empirical study of ensemble learners, including bagging, boosting and Forest-RI, is presented. From the empirical results and insights into the way in which certain mechanisms of random forests affect bias and variance, a novel random forest framework, *viz.* oblique random rotation forests, is proposed. Although not entirely satisfactory, the framework serves as an example of a heuristic approach towards novel proposals based on bias-variance analyses, instead of an ad hoc approach, as is often found in the literature.

The analysis of comparative studies regarding advances in random forest algo-

rithms is also considered. It is of interest to critically evaluate the conclusions that can be drawn from these studies, and to infer whether novel random forest algorithms are found to significantly outperform Forest-RI. For this purpose, a meta-analysis is conducted in which an evaluation is given of the state of research on random forests based on all (34) papers that could be found in which a novel random forest algorithm was proposed and compared to already existing random forest algorithms. Using the reported performances in each paper, a novel two-step procedure is proposed, which allows for multiple algorithms to be compared over multiple data sets, and across different papers. The meta-analysis results indicate weighted voting strategies and variable weighting in high-dimensional settings to provide significantly improved performances over the performance of Breiman's popular Forest-RI algorithm.

Uittreksel

Ontwikkelings rakende 'random forests' met klassifikasie as toepassing

A. Pretorius

Tesis: MComm (Wiskundige Statistiek)

Desember 2016

Sedert hulle bekendstelling is *random forests* met groot sukses in 'n wye verskeidenheid toepassings geïmplementeer. 'n Aantal algoritmes wat aan Leo Breiman se definisie van 'n *random forest* voldoen, is redelik onlangs in die literatuur voorgestel. Breiman se gewilde *random forest (Forest-RI)* algoritme en verwante ensemble klassifikasie algoritmes wat daaruit ontwikkel is, vorm die fokus van die studie. 'n Oorsig van nuut ontwikkelde *random forest* algoritmes wat sedert die bekendstellig van *Forest-RI* voorgestel is, word gegee. Dit sluit 'n nuwe kategoriseringsraamwerk van random forest algoritmes in, wat gebaseer is op hulle bron van ewekansigheid, asook op hulle tipe deterministiese wysigings. Met behulp van meerdimensionele skalering word 'n visuele voorstelling van bydraes in die literatuur ten opsigte van *random forest* algoritmes ook gegee.

Met die oog op 'n analise van ontwikkelings rondom *random forest* algoritmes, word die opdeling van die verwagte vooruitskattingsfout in 'n sydigheid- en variansie komponent beskou. In vergelyking met regressie wanneer die gekwadreerde-fout verliesfunksie gebruik word, is hierdie opdeling in klassifikasie minder voor-die-hand-liggend. Derhalwe kom verskeie definisies van sydigheid en variansie vir klassifikasie in die literatuur voor. Deur gebruik te maak van 'n spesifieke sydigheid-variansie opdeling word 'n empiriese studie van ensemble algoritmes, ingesluit *bagging, boosting* en *Forest-RI*, uitgevoer. Uit die empiriese resultate en insigte rakende die manier waarop sekere meganismes van *random forests* sydigheid en variansie beinvloed, word 'n nuwe *random forest* raamwerk voorgestel, *nl. oblique random rotation forests*. Hoewel nie in geheel bevredigend nie, dien die raamwerk as 'n voorbeeld van 'n heuristiese benadering tot nuwe voorstelle gebaseer op sydigheid-variansie analises in plaas van 'n ad hoc benadering, soos wat dikwels gevind word in die

literatuur.

Verder word vergelykende studies met betrekking tot *random forests* geanaliseer. Hier is dit van belang om gevolgtrekkings wat uit vergelykende studies gemaak is, krities te evalueer, en om te verifieer of nuwe *random forest* algoritmes betekenisvol verbeter op *Forest-RI*. Met bogaande doelwitte in gedagte is 'n meta-analise uitgevoer waarin die stand van *random forest* navorsing geëvalueer is. Die analise is gebaseer op al (34) artikels waarin 'n nuwe *random forest* algoritme voorgestel is en vergelyk word met reeds bestaande *random forest* algoritmes. Deur gebruik te maak van die gerapporteerde prestasie-maatstawwe in elke artikel, is 'n nuwe prosedure voorgestel waarvolgens 'n aantal algoritmes oor 'n aantal datastelle en oor verskillende artikels vergelyk kan word. Die resultate van die meta-analise toon aan dat geweegde stem-strategieë en die weging van veranderlikes in hoë-dimensionele data 'n betekenisvolle verbetering lewer op die akkuraatheid van Breiman se gewilde *Forest-RI* algoritme.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- Dr. S. Bierman for her expert guidance, support, understanding and motivation over the entire duration of this thesis;
- Prof. S.J. Steel for his never wavering meticulous attention to detail, his advice and mentorship;
- The MIH Media Lab at Stellenbosch University for financial support.
- The National Research Foundation (NRF) for financial support under grant number: UID 94615. *Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.*
- All my friends at the Media Lab who provided an endless source of interesting conversation, new insights, wonderful ideas and general support and companionship;
- My loving parents, for all their hard work, emotional and financial support through the years that enabled me to be where I am today;
- Last, but definitely not least, my fiancée for her constant support, love and understanding.

Contents

List of Figures

List of Tables

Frequently used notation

Inputs (predictors)

X: Random input variable, could either be a scalar or an $N \times 1$ vector.

x: Observed value of the random input variable X, either a scalar or an $N \times 1$ vector depending on the nature of X.

$\boldsymbol{X}$: Random $p \times 1$ vector of inputs, *i.e.* $\boldsymbol{X}^T = [X_1, ..., X_p]$. Note the difference between X and $\boldsymbol{X}$: the former refers to a single input variable whereas the latter refers to an observation, *i.e.* a data point in p-dimensional space.

$\boldsymbol{x}$: Observed $p \times 1$ vector of inputs, *i.e.* $\boldsymbol{x}^T = [x_1, ..., x_p]$, *i.e.* an observed data point in p-dimensional space.

Outputs (responses)

Y: Quantitative output variable (in regression).

y: Observed value of the output variable Y, where usually $y \in \mathbb{R}$.

C: Qualitative output variable (in classification).

c: Observed value (group or class) of the quantitative output variable C, where usually $c \in \{1, ..., K\}$, a set consisting of K possible classes.

Data

Ω: A generic data set consisting of N observations of input-output pairs where there are p input variables and a single output variable.

Ω_{TR}: Random training data represented as a set $\{(\boldsymbol{X}_i, C_i), i = 1, ..., N\}$.

Ω_{tr}: A particular (observed) training data set $\{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$.

Ω_{TE}: Random test data (unseen by a learning algorithm), represented by the set $\{(\boldsymbol{X}_{0i}, C_{0i}), i = 1, ..., N_0\}$.

Ω_{te}: A particular (observed) test data set $\{(\boldsymbol{x}_{0i}, c_{0i}), i = 1, ..., N_0\}$.

Ω^*: A bootstrap data set (size N) obtained from sampling with replacement from Ω_{tr}.

Functions

$g(\cdot), f(\cdot), t(\cdot)$: Estimated functions mapping inputs to outputs (sometimes abbreviated as g or f). In the case of regression, $f(\boldsymbol{x})$ will be used to indicate that the inputs are mapped to a numerical quantity. In classification, $g(\boldsymbol{x})$ will indicate that the inputs are mapped to a categorical quantity. The function t will be used specifically for tree based classification algorithms.

$g_B(\cdot), f_B(\cdot)$: The Bayes (model) function, which is the theoretically optimal function producible by a learning algorithm.

$g_{\Omega_{tr}}(\cdot)$: An estimated function obtained from a learning algorithm which was trained on a particular training set Ω_{tr}.

$\bar{g}_{\Omega_{TR}}(\cdot)$, or $\bar{g}(\cdot)$: The majority vote classifier defined at a point $\boldsymbol{x}$ by

$$\bar{g}(\boldsymbol{x}) = \arg\max_k E_{\Omega_{TR}}\big\{I(g(\boldsymbol{x}) = k)\big\},$$

where $I(\cdot)$ is the indicator function which is equal to 1 when the argument is true and equal to 0 otherwise.

Probability

$P(X)$: The probability distribution of the random variable X.

$P(X, C)$: The joint distribution of X and C.

$P(X = x)$, or $P(x)$: The probability that the random variable X takes on the value x.

$P(C = k|\boldsymbol{X} = \boldsymbol{x})$, or $P(k|\boldsymbol{x})$: The conditional probability that the random variable C takes on the qualitative value k given that the random vector $\boldsymbol{X}$ has taken on the realisation $\boldsymbol{x}$.

$\hat{P}(C = k|\boldsymbol{X} = \boldsymbol{x})$, or $\hat{P}(k|\boldsymbol{x})$: The estimated conditional (posterior) probability that the random variable C takes on the qualitative value k given that the random vector $\boldsymbol{X}$ has taken on the realisation $\boldsymbol{x}$, as estimated by a classification algorithm trained on Ω_{tr}.

$P_{\Omega_{tr}}(\cdot)$: Response class probability distribution as estimated by a learning algorithm on a particular training set Ω_{tr}.

$P_{\Omega_{TR}}(\cdot)$: Response class probability distribution as estimated by repeated sampling from Ω_{TR}, in other words, the probability distribution as estimated by the majority vote model.

Chapter 1

Introduction

Statistical learning theory has contributed extensively to the development of highly accurate and interpretable supervised classification and regression models. In particular, *random forests* have been shown to perform extremely well when compared to other models, and can provide highly accurate predictions using minimal tuning time (Caruana and Niculescu-Mizil, 2006). The areas of application for random forests stretch across a wide range of academic and industry domains such as Ecology (Cutler *et al.*, 2007), Medicine (Klassen *et al.*, 2008), Astronomy (Gao *et al.*, 2009), Business (Larivière and van Den Poel, 2005), Bioinformatics (Boulesteix *et al.*, 2012), Transport Planning (Zaklouta *et al.*, 2011), and more recently the domain of Expert Systems (Braida *et al.*, 2015). Furthermore, recent novel proposals have been demonstrated to sometimes remarkably outperform current state-of-the-art random forest algorithms as well as other popular supervised learning approaches (Rodriguez *et al.*, 2006; Menze *et al.*, 2011; Seyedhosseini and Tasdizen, 2015). The popularity and widespread use of random forests, as well as ongoing attention to novel random forest proposals in the literature indicate the impact and influence of random forest algorithms.

1.1 Motivation and Thesis Objectives

The aim of this study is an investigation of both the earlier and more recent random forest algorithms in a supervised classification setting. More specifically, a study with regard to their construction, properties pertaining to bias and variance, as well as with regard to their performance in various artificial and real world scenarios, is of interest.

1.1.1 Motivation

The novelty of recent random forest proposals leads to a number of research questions and avenues that could possibly lead to contributions in this field.

First, there is scope for a comprehensive review and proper conceptualisation of novel random forest proposals in the literature. In order to facilitate a good overview and understanding of the various proposals, integration of developments into a conceptual framework seems needed.

A second research question relates to which of the developments in random forest algorithms lead to significant improvements in terms of accuracy. In order to recommend some proposals above others, an investigation of comparative studies for random forest algorithms is required. Such an investigation involves a critical evaluation of the validity of reported results, and may in turn lead to a framework for best practices in comparative studies involving novel classification algorithms. For example, appropriate experimental design and methodology pertaining to comparative studies based on simulation and benchmark data sets may create a framework in which novel proposals can be evaluated. Moreover, a study of possible bias-variance decompositions in classification may facilitate a deeper level of comparison amongst algorithms. In the current literature, aspects pertaining to the bias and variance associated with random forests are yet to be fully explored.

Many random forest proposals in the literature seem to be fairly ad hoc. Hence a third research question is whether a heuristic motivation for novel random forest proposals may follow from an analysis of comparative studies.

1.1.2 Thesis Objectives

The objectives of this study may be summarised as follows: To gain and facilitate a comprehensive understanding of random forests; to obtain and illustrate insights into their design, and into their bias and variance characteristics; and to summarise and further analyse their comparative performances. These main objectives may be expanded upon as follows:

- To provide a review of classification trees, ensemble classifiers and the history and development of the earlier random forest contributions;
- To provide a review of more recent random forest algorithms that were proposed with a view to potentially improve upon the performance of earlier contributions;
- To propose a framework in order to conceptualise, structure and integrate the more recent proposals;
- To study possible bias-variance decompositions in classification in order to facilitate a deeper level of comparison amongst ensemble learners and amongst random forest algorithms;
- To investigate the use of an appropriate bias-variance decomposition in an empirical comparison of ensemble learners, and also of random forest algorithms;

- To investigate the possibility of using insights gained from the above empirical study to motivate a novel random forest proposal;
- To conduct a meta-analysis of reported results in comparative studies on random forest algorithms, and following this analysis, to recommend best practices in order to provide a framework in which novel algorithms can more easily be compared;
- To use insights gained from empirical investigations and from the meta-analysis to recommend some recent random forest algorithms above others.

Finally with regard to the objectives of this study, an important overall objective of the work presented here is that it should be as transparent and reproducible as possible. This aspect is further discussed in the next section.

1.2 Data, Code and Reproducibility

Given a scientific article or dissertation that reports findings from an analysis, Peng (2015) partitions the reproducibility of results into three levels, as shown in Figure 1.1.

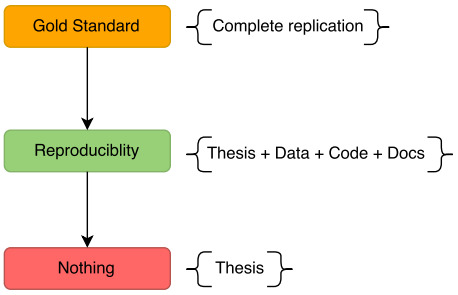

Figure 1.1: Levels of replication.

The "Gold Standard" refers to perfect replication. This means that all the necessary resources in terms of measurement mechanisms, computational hardware and software as well as the steps taken in the analysis are available to such an extent that an exact copy of the original study can be conducted. At the other end of the spectrum lies a study that allows no replication at all. Here only the information regarding the findings provided in the thesis are given.

Reproducible research lies somewhere in the middle. The idea is to make available all of the data, code and associated documentation in such a way

that a researcher will be able to *reproduce* the study. This accessibility is crucial since typically the journey for the reader of scientific research starts at the opposite end to that of the author, as depicted in Figure 1.2 (Peng, 2015).

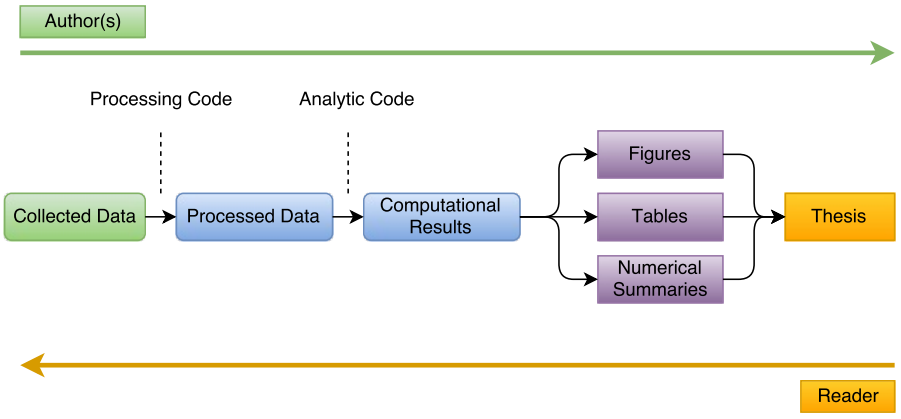

Figure 1.2: Research Pipeline.

The author moves from left to right in Figure 1.2, first collecting the data, processing it and then obtaining results through computational analysis. Between each step are code segments that perform tasks to either transform raw data into tidy data, or tidy data into results. Finally, all the findings are summarized and condensed into an article or thesis consisting of figures, tables and/or numerical summaries. In contrast, the reader who is interested in reproducing the research starts from the right and moves to the left. Without access to data and code, a reader must decipher what the author did, given only the information presented in the report. Therefore, the goal of reproducible research is to essentially give the reader the ability to start from the same position as the author, while at the same time adding the missing pieces between each transformation in the form of code. Therefore, in order to consider research reproducible from the perspective of the reader, four things are required:

1. The collected data.
2. Processing and analytic code.
3. Data and code documentation.
4. Public access to a distribution platform.

In this regard, all of the relevant data, processing and analytic code as well as related documentation for this thesis can be found at the following public

on-line repository:

- https://github.com/arnupretorius/MastersThesisCode.

This includes the data collected and used for the meta-analysis conducted in Chapter 7. For quick reference, the code is also provided in Appendix D.

An R package, which provides for the implementation of the novel random forest framework presented in the thesis, can be downloaded and sourced from within R using the following commands:

R Code 1.1: Random Rotation Forest R Package

```
# download and load random rotation forests package
if("devtools" %in% installed.packages()[,"Package"] == FALSE){
      install.packages("devtools")
}
library(devtools)

# Github profile: Arnu Pretorius
install_github("arnupretorius/RRotF")
library(RRotF)
```

For more details on the internal functions and code used in the package, the reader is referred to Appendix D.1.

In order to sketch the background to this study, in the remainder of this chapter, the most important concepts pertaining to random forest algorithms for classification are discussed.

1.3 Important Concepts and Terminology

One of the uses for data is to gain insight into the inner workings of a system or phenomenon. Another possibly more common use is to obtain accurate predictions. This task usually takes the form of using a set of inputs $X_1, ..., X_p$ to predict an unknown output value Y. An approach often considered when faced with such a setting is *supervised learning*. The basic idea of supervised learning is to monitor the system of interest over a period of time and to collect data of both the inputs and the corresponding outputs. Once the data have been collected, both the recorded input and output values can be used to extract a set of rules. Through these rules knowledge of the input values facilitates prediction of the corresponding output values. The above rule extraction is typically performed by a predefined algorithm known as a *learning* algorithm. The algorithm is "trained" using a data set at hand in order to approximate

the unknown mechanism governing the system of interest. It is for this reason that the recorded data consisting of input-output pairs is named the *training* data. The architecture of a learning algorithm often depends on the nature of the output value of interest. The main distinction is between quantitative outputs Y which lead to the development of *regression* algorithms and qualitative outputs C which lead to the development of *classification* algorithms. In this study the focus will be on classification.

1.3.1 Supervised Learning for Classification

In classification the output C takes on values $k \in \{1, ..., K\}$ representing different groups or classes to which inputs of the system may belong. Most of the examples and theory in this text will focus on the (common) case in which $K = 2$, however some of the theory will cover the case where $K \geq 3$. Mostly, it is possible to intuitively extrapolate ideas from binary classification to multi-class classification. For the time being, without loss of generality, let $C \in \{0, 1\}$. The aim of classification is to assign an observed input $\boldsymbol{x}$, which is not part of the training data, to the correct output category C. In practice, a given input to a system often does not belong to a unique class. As a consequence, classification often involves the estimation of class probabilities. For example in binary classification, suppose $\boldsymbol{x} \in \mathbb{R}^p$, then $\boldsymbol{x}$ belongs to Class 1 with probability $P(C = 1|\boldsymbol{x}) = 1 - P(C = 0|\boldsymbol{x})$. In this thesis the goal of classification is to find a *classifier* $g(\boldsymbol{x})$, via a learning algorithm using the training data, capable of estimating these probabilities such that $\boldsymbol{x}$ is correctly classified.

1.3.2 Expected Loss and the Bayes Classifier

Naturally, misclassification of a point $\boldsymbol{x}$ by a classifier $g(\boldsymbol{x})$ will have associated with it some form of cost or loss. Suppose ℓ_k is the loss incurred for misclassifying $\boldsymbol{x}$ as belonging to class $k, k \in \{0, 1\}$. Then the expected loss is

$$E\big[L(C, g(\boldsymbol{x}))\big] = \ell_0 I(g(\boldsymbol{x}) = 0)P(C = 1|\boldsymbol{x}) + \ell_1 I(g(\boldsymbol{x}) = 1)P(C = 0|\boldsymbol{x}), \tag{1.3.1}$$

where $I(\cdot)$ is the indicator function which assumes a value 1 if its argument is true and 0 otherwise, while $L(\cdot, \cdot)$ is a loss function. If $\ell_0 = \ell_1 = 1$, the loss function becomes $L_{0\text{-}1}(C, g(\boldsymbol{x})) = I(g(\boldsymbol{x}) \neq C)$, which is known as the *0-1 loss* with an expected value

$$E\big[L_{0\text{-}1}(C, g(\boldsymbol{x}))\big] = I(g(\boldsymbol{x}) \neq 1)P(C = 1|\boldsymbol{x}) + I(g(\boldsymbol{x}) \neq 0)P(C = 0|\boldsymbol{x}). \tag{1.3.2}$$

Note that (1.3.2) implies that if $P(C = 1|\boldsymbol{x}) > 0.5$, the expected loss incurred for misclassifying $\boldsymbol{x}$ as belonging to Class 1 is $P(C = 0|\boldsymbol{x}) = 1 - P(C = 1|\boldsymbol{x}) < P(C = 1|\boldsymbol{x})$, the loss for the opposite mistake. The optimal classifier which will minimise the expected loss in this situation will therefore be a rule classifying to the most probable class,

$$g_B(\boldsymbol{x}) = I\left(P(C = 1|\boldsymbol{x}) \geq \frac{\ell_1}{\ell_0 + \ell_1} = \frac{1}{2}\right), \tag{1.3.3}$$

called the *Bayes* classifier. It is important to realise that even the optimal (Bayes) classifier will rarely achieve perfect classification due to the intrinsic probabilistic nature underlying observable systems. In addition, although it is often the case that $\ell_0 = \ell_1$, it is also not uncommon that $\ell_0 \neq \ell_1$. Asymmetry related to the two types of misclassification departs from the 0-1 loss framework and classification by way of the most probable class. However (1.3.3) still holds, and the threshold will simply be adjusted appropriately (away from 0.5). In this text most of the examples and theory will be for the case where $\ell_0 = \ell_1$, and the loss function under study is the 0-1 loss.

1.3.3 Generalisation Error

A related quantity to that of expected loss is the *generalisation* or *test* error of a classifier which depends on a particular training set. Let the set of input-output pairs forming a training set (of size N) be denoted by $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$. The generalisation error associated with $g(\boldsymbol{x})$ is then

$$Err^* = E\big[L(C, g(\boldsymbol{x}))\big|\Omega_{tr}\big]. \tag{1.3.4}$$

One is typically more interested in obtaining an estimate of (1.3.4) than obtaining an estimate of the expected loss in (1.3.2). This is because the generalisation error more closely resembles the error that is made in practice. The expectation in (1.3.2) is taken over all possible training data sets, which is clearly a quantity further from reality. Furthermore, it makes sense to be more interested in the question: "given a particular training data set, how accurate will a trained classifier be in the future?", than: "given training data taken *repeatedly* from the system, how accurate on *average* will a trained classifier be in the future?" Hence supervised learning for classification seeks to find a function $g(\boldsymbol{x})$ by way of a learning algorithm trained using a particular training set Ω_{tr} having an associated generalisation which is as small as possible.

1.3.4 Trees for Classification

Suppose two input variables X_1 and X_2 are believed to be related to a qualitative binary output variable $C \in \{+, -\}$. The two-dimensional *input space*, depicting the locations of training data points, is presented in Figure 1.3 (Fuchs, 2014).

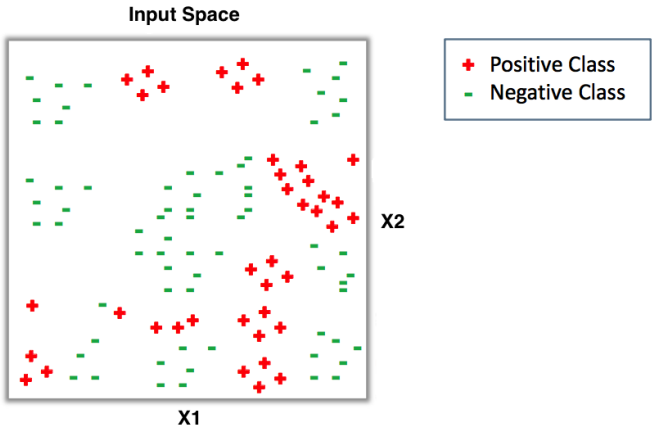

Figure 1.3: Input space for a binary classification problem.

In binary classification, the objective is to use these points to learn a rule capable of separating as well as possible the positive cases illustrated by the red plus signs in Figure 1.3 from the negative cases presented in green. (In the domain of medicine, the colouring makes sense: the positive class may indicate that a patient has a certain disease, while the negative class indicates the disease to be absent). A simple strategy is to a find an appropriate straight line to separate the two classes using a linear combination of the inputs. That is, one may use $\hat{\beta}_0 + \hat{\boldsymbol{\beta}}_1^T \boldsymbol{x}$, where $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}_1$ are estimated *coefficient* vectors (respective "weights" for each input). This is the approach taken by popular linear classifiers such as *linear* and *logistic regression*. For example, in linear regression each class is modelled separately and a hyperplane (strictly an affine set) $\{\boldsymbol{x} : (\hat{\beta}_{0+} - \hat{\beta}_{0-}) + (\hat{\boldsymbol{\beta}}_+ - \hat{\boldsymbol{\beta}}_-)^T \boldsymbol{x} = 0\}$ defines a linear *decision boundary* between each class (Hastie *et al.*, 2009). Estimated class probabilities are then produced as a function of the distance to the decision boundary and subsequently used for prediction. However for this data example, Figure 1.4 shows the difficulty of obtaining good separation using a simple linear combination of the inputs (Fuchs, 2014).

A common way in which linear classifiers amend this situation is by augmenting the input space with higher orders of the original inputs and/or with interaction terms and by then constructing a linear separating hyperplane in this

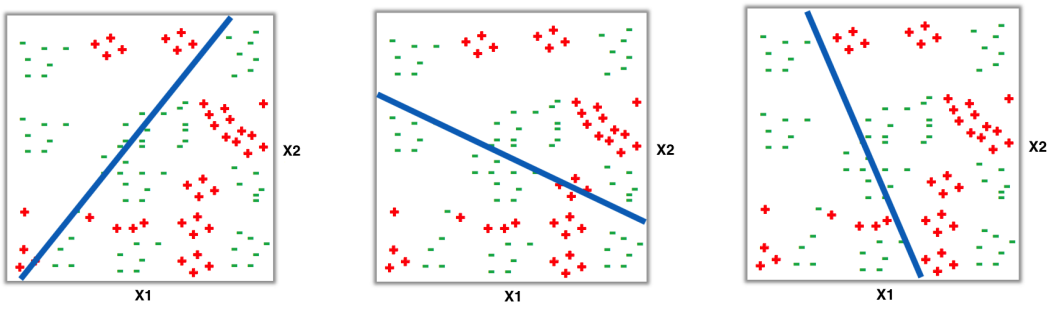

Figure 1.4: Class separation using linear combinations of the input variables: *Each panel shows a different linear combination of the input variables to create a boundary that attempts to separate the two classes from each other.*

augmented space. Once the decision boundary is projected onto the original input space it will be non-linear and better able to separate the two classes. Examples of popular classifiers in this framework include *flexible discriminant analysis* and *support vector machines*. But consider as alternative the following strategy: take the input space and split it into two rectangular regions achieving a reasonable degree of separation. Next, treat each partition as the original input space and split those into two smaller rectangular regions. Continue in this way until each region only contains points belonging to a single class. These steps are illustrated on the left side of each panel in Figure 1.5 where the input space is split into regions $A, B, ..., L$ (Fuchs, 2014).

The right side of each panel is an isomorphic representation of each recursive binary partitioning. Each picture resembles the shape of an upside down tree. Therefore the name given to this type of classifier is a *classification tree*. (A more formal treatment of classification trees is given in Chapter 2.) Each of the eleven numbered circles in the tree in the bottom right panel of Figure 1.5 is called a *node*. Circle 1 at the top of the tree is called the *root* node, while all other circles represent *internal* nodes of the tree. Each node represents a rule specified by both a variable and a split-point, mapping out the recursive partitions made in order to obtain each region. The regions $A, ..., L$ are called the *terminal* nodes of the tree. For each of these terminal regions, $P(C = +|\boldsymbol{x}^0) = 1 - P(C = -|\boldsymbol{x}^0)$ can be estimated using the proportion of positive training data observations in that region.

At prediction time, a new unseen observation $\boldsymbol{x}_0$ is "dropped" from the root at the top of the tree. Based on which rules are satisfied, $\boldsymbol{x}_0$ then follows a specific path down to a terminal node as shown in Figure 1.6 (Fuchs, 2014). Suppose that in this way $\boldsymbol{x}_0$ ends up in the terminal node F. With symmetric loss $\boldsymbol{x}_0$ is assigned to the positive class if $\hat{P}(C = +|\boldsymbol{x})$ associated with the terminal node F is larger than $\hat{P}(C = -|\boldsymbol{x})$, and vice versa.

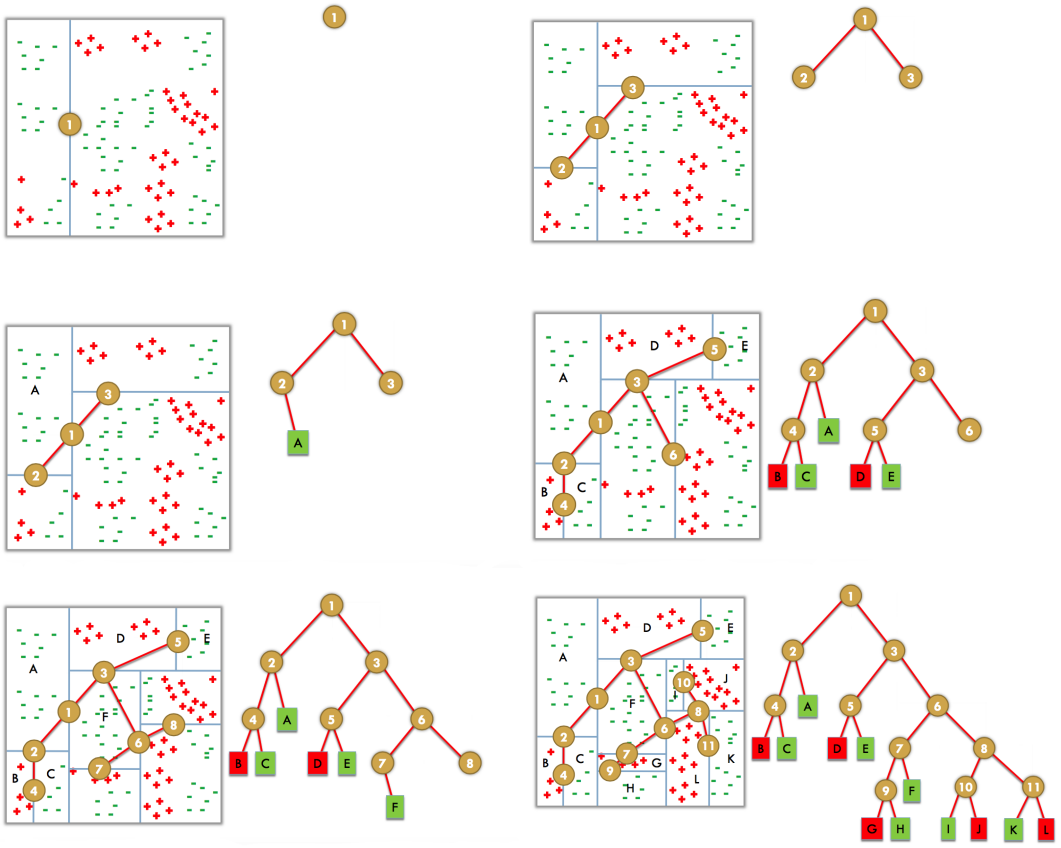

Figure 1.5: Stage by stage construction of a binary classification tree: *Moving from the top left to the bottom right, each panel shows a partitioned region and corresponding tree representation during different stages of constructing a classification tree.*

Note that in Figure 1.6, for any terminal node in the tree

$$\max\{\hat{P}(C = +|\boldsymbol{x}), \hat{P}(C = -|\boldsymbol{x})\} = 1.$$

This means the tree is *fully grown*, containing only *pure* terminal nodes in which all training data observations belong to the same class.

1.3.5 Training Error and the Bias-Variance Trade-off

The *training error* of a classifier, *viz.*

$$\bar{Err} = \frac{1}{N}\sum_{i=1}^{N} L(c_i, g(\boldsymbol{x}_i)) \tag{1.3.5}$$

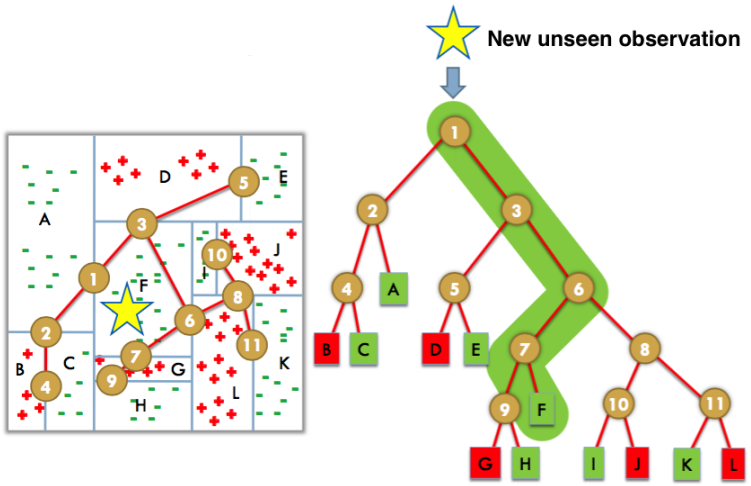

Figure 1.6: Prediction using a classification tree.

measures the average misclassification loss over the training set. An approach towards constructing a classifier g might be to find g that minimises $\bar{Err}$. For fully grown trees, the training error $\bar{Err}$ is always equal to zero (trees need not be fully grown — see Section 2.4). However, this by no means guarantees that the tree classifier will generalise well to data presented to it in the future.

The drawback of the maximum sized tree is that the measure of variability dependent on the locations of the sampled training points, *i.e.* the *variance* of the classifier, will be high. In other words, it can be imagined that if a new round of data recording took place from the same system under study, the layout of the partitioned space might change considerably from the one presented in Figure 1.5. This is in contrast with a simple linear separating boundary that might only change slightly when new data is sampled. Less complex approaches such as linear regression are "rigid" in this sense and have low variance. However, they rely heavily on the rather strict assumption that the separating hyperplane appropriate for the data is a $(p-1)$-dimensional flat surface. If this assumption is incorrect, the incurring high bias will cause the performance of linear classifiers to suffer. Trees, on the other hand, are more complex, with few assumptions regarding the shape of the decision boundary. Therefore trees generally have higher variance and lower bias than linear classifiers. This trade-off based on model complexity is referred to as the *bias-variance trade-off*. In fact, using $L_{SE}(Y, f) = (Y - f)^2$, it can be shown that for any quantitative response (which include probability estimates) the expected loss at a point can be decomposed into an additive formula containing three terms: irreducible (Bayes) error, bias and variance. More detail in this regard is provided in Section 5.1. Nothing can be done about the irreducible error stemming from the nature of a system or phenomenon. The ideal is an

algorithm designed to balance the trade-off between bias and variance in an optimal way. The bias-variance decomposition is more complicated for 0-1 loss. This is discussed in Chapter 5.

1.3.6 Beyond a Single Tree

Much of the recent success in the development of statistical learning algorithms have gone the way of using multiple classification trees as *base learners*, combined in clever ways to produce a better performing *ensemble classifier*. To loosely motivate this approach, from a bias-variance perspective, consider the following general argument (Hastie *et al.*, 2009). Let $X_1, ..., X_B$ be identically distributed variables, not necessarily independent, with $Var(X_i) = \sigma^2$ and $Cov(X_i, X_j) = \rho\sigma^2, \forall i, j, i \neq j$. Also let $\bar{X} = \frac{1}{B}\sum_{i=1}^{B} X_i$. Then $Var(\bar{X})$ may be written as

$$\begin{aligned} Var(\bar{X}) &= Var\left(\frac{1}{B}\sum_{i=1}^{B} X_i\right) \\ &= \frac{1}{B^2}\sum_{i=1}^{B} Var(X_i) + \frac{1}{B^2}\sum_{\substack{i=1 \\ i\neq j}}^{B}\sum_{j=1}^{B} Cov(X_i, X_j) \\ &= \frac{1}{B^2}\sum_{i=1}^{B} \sigma^2 + \frac{1}{B^2}\sum_{\substack{i=1 \\ i\neq j}}^{B}\sum_{j=1}^{B} \rho\sigma^2 \\ &= \frac{1}{B}\sigma^2 + \frac{1}{B^2}(B^2\rho\sigma^2 - B\rho\sigma^2) \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \end{aligned} \tag{1.3.6}$$

By increasing B, the second term in (1.3.6) can be made arbitrarily small. Hence from (1.3.6) it is clear that, taking the average over a large number of random variables reduces the variance of $\bar{X}$. The above serves to motivate aggregation. However, the first term remains unaffected by the size of B and only interacts with the magnitude of the correlation between the variables. Proceeding one step further, many proposals have been made that in addition to aggregation, induce some degree of artificial randomness into the algorithm. This can actually have an effect of reducing the correlation ρ and thus further reduce the variance.

Random forests was the catch-all name proposed for these types of algorithms, but the title has since been slightly reserved to refer to only a well known special case (Breiman, 2001*a*). As mentioned before, it is the investigation of

these randomised tree ensemble classifiers and later developments that followed, which form the main focus of this study.

1.4 Outline

In this section an outline of the thesis is presented. Figure 1.7 provides a display of the organisation of each chapter in relation to the remainder of the thesis. This may be used as a quick reference and "road map" throughout the text.

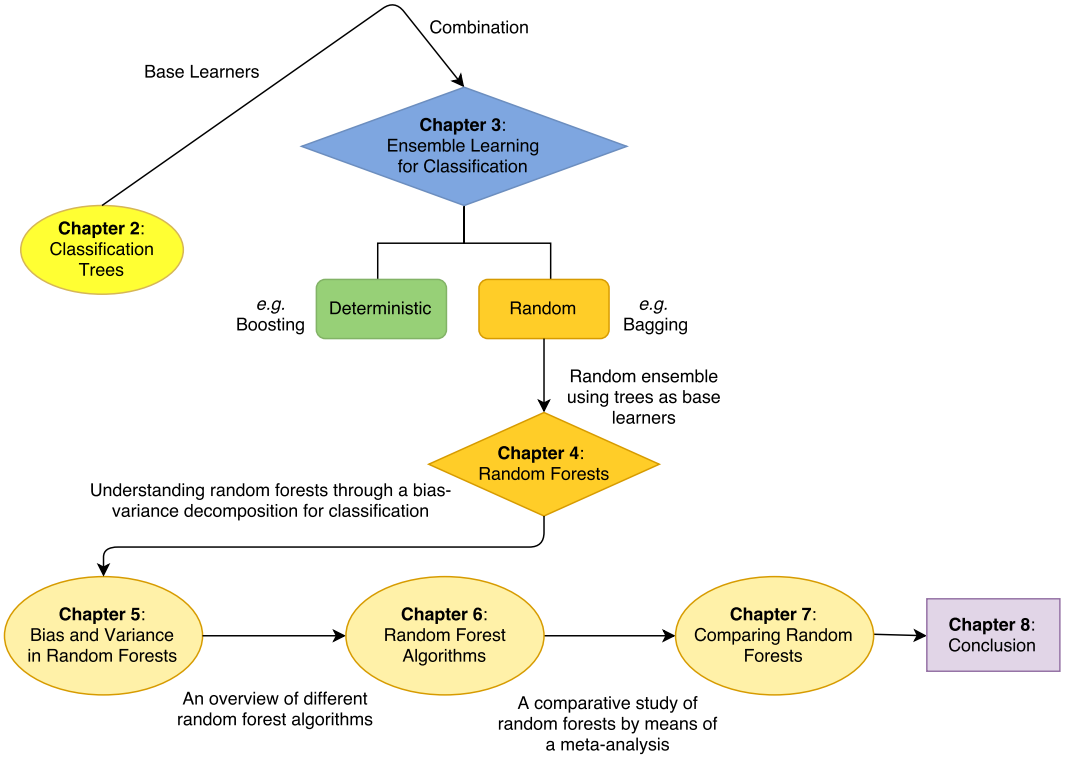

Figure 1.7: Thesis outline.

In **Chapter 2**, classification trees are discussed in more detail. As previously mentioned, trees form the foundation for many ensemble learning algorithms, serving as base learners which are combined to create powerful classifiers that exploit the wisdom of crowds. The construction of ensemble classifiers are then discussed in **Chapter 3**, with specific emphasis on boosting and bagging. In **Chapter 4**, an introduction to random forests as random ensemble learning algorithms that exclusively use trees as base classifiers, is given. The most popular random forest algorithm (*viz.* Breiman's Forest-RI) is discussed,

along with important aspects such as the generalisation error of a random forest classifier, overfitting in random forests, out-of-bag error estimation, and interpretation. Towards a deeper understanding and comparison of ensemble classifiers, **Chapter 5** is devoted to a study of bias and variance in random forests. The differences between regression and classification are emphasised, leading to a discussion of the Friedman effect in classification, and of various bias-variance decompositions that may be found in the literature. The bias and variance (and their respective effects) of ensemble classifiers are investigated by means of a simulation study. This allows a comparison of a single classification tree with bagging, Forest-RI, and boosting. A novel taxonomy of random forest algorithms based on possible traits pertaining to an algorithm's deterministic modifications and/or sources of randomisation is given in **Chapter 6**. Four sources of randomisation are identified and discussed. Possible deterministic modifications are also divided into four categories and discussed. A visual perspective of later contributions in the random forests literature is provided by means of multi-dimensional scaling. This is followed by an empirical investigation of bias and variance (and their respective effects) in the case of several random forest algorithms, *viz.* Forest-RI, extremely randomised forests, rotation forests and oblique random forests. The focus in **Chapter 7** is on a meta-analysis of all results reported in research on random forests. An evaluation of the selection of performance measures, of performance estimation, of methods to compare algorithms, and of the reproducibility of research is discussed. The two main outcomes of the meta-analysis are the following. The results facilitate recommendation of some random forest algorithms above others, leading to the proposal of a novel random forest framework. Also, a novel two-step procedure is proposed for comparing a novel algorithm to established algorithms. This is then used to evaluate the proposed random forest framework. Finally, a summary of main findings and some concluding remarks are given in **Chapter 8**.

Chapter 2

Classification Trees

In this chapter, classification trees are discussed. A revisit to the two ways of representing a binary classification tree together with more formal terminology is presented in Section 2.2. An approach towards finding the regions of the partitioned input space is outlined in Section 2.3. This is followed by a discussion of the way in which the optimal tree size is typically determined in Section 2.4. Finally, Section 2.5 provides an example of fitting a classification tree to simulated data, with concluding remarks given in Section 2.6.

2.1 Introduction

Classification trees have been used in many different domains, including Medicine (Nair *et al.*, 2002), Ecology (Vayssières *et al.*, 2000; Smith *et al.*, 1997), Clinical Psychology (Ostrander *et al.*, 1998), and the study of natural language (Kuhn and De Mori, 1995). James *et al.* (2013) provide the following advantages of classification trees over other classification approaches:

- They are very easy to explain.
- They can be seen as more closely mirroring human decision-making than other classification approaches.
- They can be displayed graphically for easy interpretation.

However, as mentioned in Section 1.3, a disadvantage of trees is that they have high variance. Therefore, although they are easy to interpret, it should be borne in mind that these interpretations stem from an unstable source. As a consequence, any conclusions that are drawn must be done with caution (Hastie *et al.*, 2009).

The basic algorithm for a classification tree consists of the following steps (James *et al.*, 2013):

1. Partition the input space into M non-overlapping regions, $R_1, R_2, ..., R_M$.
2. Predict the response of every observation falling in Region R_m as the response of the majority class in R_m.

Through the above algorithm, the original input space which is typically heterogeneous with respect to the training output values, is divided into more homogeneous regions. The underlying rationale is that prediction is simplified in settings where the observed output values vary less. The resulting regions $R_1, R_2, ...$ in a sense define "local" neighbourhoods, each containing points with certain input values. Subsequently, new unseen observations are associated with a local neighbourhood and classified using knowledge of the class distribution within that neighbourhood.

2.2 Tree Representation and Terminology

Consider two continuous input variables X_1 and X_2 and a two-dimensional input space defined by their joint values. An illustration of a possible recursive binary partitioning of such a space is given in Figure 2.1.

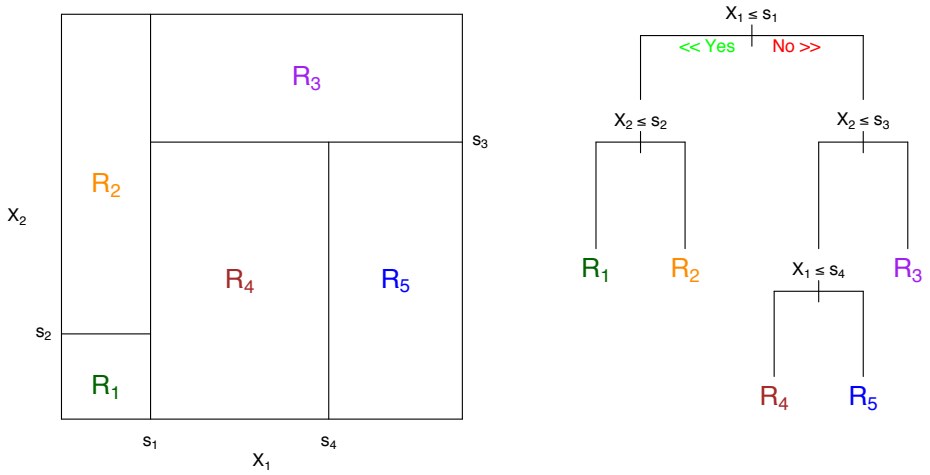

Figure 2.1: Recursive binary partitioning: *The left panel shows a partition of a two-dimensional input space and the right panel displays the corresponding tree obtained from recursive partitioning.*

As seen before, each of the regions $R_1, ..., R_5$ plays a dual role: each region represents a rectangular subspace of the original input space, as well as a terminal node of the corresponding tree. In this text both will be denoted by $R_m(\boldsymbol{j}, \boldsymbol{s})$ (the context will distinguish between a subspace or a node), where the arguments $\boldsymbol{j} = \{j_1, j_2, ...\}$ and $\boldsymbol{s} = \{s_1, s_2, ...\}$ signify the specific region R_m to depend on a vector of variable indices and corresponding split-points on those variables. A partitioning boundary is given by a pair (j, s), where j indicates the variable on which to split and where s denotes the value at which to split the variable X_j. The above notation is needed to annotate rules such as "$X_1 \leq s_1$" at each internal node. If a rule is satisfied at a particular node, the next rule is tested at the left node a level further down the tree, referred to as the left *child* node of the current node. Otherwise, the rule at the right child node is tested. A sequence of such rules ultimately defines a region.

Note that the representation provided in the left panel of Figure 2.1 is restricted to at most three dimensions. In contrast, the tree diagram depicted in the right panel can be visualised and interpreted even for large dimensions of the input space. Therefore, even though the terminology presented here is largely interchangeable, the fact that the latter representation is more commonly adopted, the discussion will be more often phrased in this setting (*i.e.* referring to nodes as opposed to hyper-rectangular subspaces).

Suppose the response C can take on values $k \in \{1, ..., K\}$ and a classification tree is fit to the training set $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$. The proportion of observations belonging to class k inside node m is

$$\hat{P}_m(k) = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} I(c_i = k), \tag{2.2.1}$$

where N_m is the total number of observations belonging to node m. An important property of a node is its impurity, which is the degree to which a node deviates from representing a homogeneous region. Node impurity can be computed in different ways. Popular measures include:

- Misclassification rate: $\frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} I(c_i \neq k) = 1 - \hat{P}_m(k)$,
- Gini index: $\sum_{k=1}^{K} \hat{P}_m(k)(1 - \hat{P}_m(k))$,
- Deviance: $-\sum_{k=1}^{K} \hat{P}_m(k) log(\hat{P}_m(k))$.

The misclassification rate is less sensitive to changes in node probabilities and is

not everywhere differentiable, therefore the Gini index and Deviance measures are more regularly used (Hastie *et al.*, 2009).

2.3 The CART Algorithm

Classification and Regression Trees ($CART$) is a well known tree induction algorithm by means of recursive binary partitioning proposed by Breiman *et al.* (1984). The first step in binary partitioning is to divide the input space into two non-overlapping regions. This is achieved by iteratively splitting over the range of different values for each input variable and selecting the optimal variable and split-point based on the sum over node impurities for both subregions. The subregions obtained from this split are then in turn partitioned into two non-overlapping regions. The above procedure is repeated until some stopping criterion is reached (more details are given in Section 2.4).

There is a sequential dependency implicit in the algorithm: each partitioning is dependent on the current state of the partitioned space. Since it is infeasible to consider all possible combinations of variable and split-point pairs, CART only selects the optimal partition *given* the current sequence of splits that have already taken place. In other words, CART is said to take a *greedy* approach towards finding homogeneous subregions (James *et al.*, 2013).

Concretely, suppose a data set consists of N observations and p input variables $X_1, ..., X_p$, together with a categorical response C form the training set $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, 2, ..., N\}$. Starting with the entire input space, consider a split variable X_j and a split-point s defining the two subregions

$$R_1(j, s) = \{X : X_j \leq s\} \text{ and } R_2(j, s) = \{X : X_j > s\}. \tag{2.3.1}$$

The optimal variable and split-point is found by solving

$$\min_{j,s} \left\{ \sum_{m=1}^{2} Q_m(j, s) \right\}, \tag{2.3.2}$$

where Q_m, denotes the impurity of node m, and $m = 1, 2$. Repetition of the above procedure on each subregion produces the tree. A new observation in node m is then classified as belonging to class $\hat{k} \in \{1, ..., K\}$, where $\hat{k} = \arg\max_k \hat{P}_m(k)$.

It is important to note that classification trees need not always use orthogonal splits to partition the input space. Oblique classification trees can be

constructed using a multivariate model to describe the split boundary at each tree node (Heath *et al.*, 1993). Furthermore, CART is not the only approach to tree induction. One of the earliest proposed strategies, called Automatic Interaction Detection (AID) was used to analyse survey data (Morgan and Sonquist, 1963). Another proposal is $CHAID$, which builds on AID employs a strategy based on subdividing cross-tabulations of the predictors and the response using χ^2 tests for significance (Kass, 1980). Furthermore, Quinlan (1986) introduced the $ID3$ tree, with subsequent improvements $C4.5$ and $C5.0$ (Quinlan, 1993). The later algorithm has become quite similar to CART (Hastie *et al.*, 2009). More recently, Hothorn *et al.* (2006) proposed a conditional inference framework aimed at relieving bias at splits. A bias can exist, for example, when a predictor has a very large range of values over which the algorithm can search compared to other variables, and is therefore split on more often. The approach has a CHAID-like flavour where statistical tests for significance are incorporated at each split.

2.4 Pruning the Tree

To fully grow a classification tree, the binary partitioning strategy presented in Section 2.3 can be continued until every terminal node of the tree only contains observations belonging to a single class. However, this corresponds to the most complex version of the tree, which from a bias-variance perspective corresponds to the tree with the highest variance. The maximum-sized tree is said to "fit the training data too closely", or to *overfit* the data if higher classification accuracy on future observations can be achieved using a smaller tree. It is unlikely that the maximum-sized tree classifier will generalise well to new unseen observations.

In this regard it is important to have an appropriate strategy for selecting the optimal size of a tree. A simple approach is to specify a threshold for the sum of node impurities, and to stop splitting once the decrease in impurity is negligible (Hastie *et al.*, 2009). Stopping early in this way is rather myopic in the sense that a suboptimal split during the initial stages of the procedure might lead to an extremely beneficial split later on, which will not be found. An alternative to stopping early by way of an impurity threshold, is to stop early by specifying a maximum terminal node size. Thus the tree is split until all nodes have at most (say) ten observations irrespective of the class distribution within each node. Specifying the maximum terminal node size to be high implies (shallow) small trees with fewer terminal nodes (small M), while specifying it to be low implies (deep) large trees with many terminal nodes (large M). The problem is that this simply substitutes the issue of finding an appropriate value for M with the issue of finding an appropriate value for the maximum node size.

The strategy which is by far the most generally used in practice is *cost-complexity pruning*. The basic idea underlying tree pruning is to start with a fully grown tree and then to "prune" off some branches in order to obtain a tree of the right size. More specifically, let t_0 represent a fully grown tree. Also let $t \subseteq t_0$ denote any subtree of t_0, or t_0 itself. Then for a specified parameter value $\alpha \geq 0$, pruning proceeds by minimising the cost-complexity criterion

$$\varsigma_\alpha(t) = \sum_{m=1}^{M_t} N_m Q_m(t) + \alpha M_t, \tag{2.4.1}$$

where M_t is the number of terminal nodes of t and $Q_m(t)$ is the impurity of node m belonging to tree t. Note that in (2.4.1), α interacts with the number of terminal nodes. Hence large values of α translate to a heavy penalty on trees with many terminal nodes, whereas a small value of α allows for larger trees to be selected. In this way α controls the bias-variance trade-off, causing it to be an important *tuning parameter* in trees.

Let t_α denote the tree minimising (2.4.1) for a specified α, then t_α can be found using the concept of a *weakest link*. The weakest link of a tree is the internal node resulting in a split having the lowest decrease in the sum of node impurities compared to all other node splits. To produce a finite sequence of different sized trees, starting with the full tree the current weakest link is removed. The procedure is continued for each successive weakest link. It can be showed that t_α lies within this sequence and that this solution is in fact unique (Breiman *et al.*, 1984).

The optimal value for α can be found using a k-fold cross-validated *grid search*.[1] For a specific value of α, k-fold cross-validation (CV) splits the training data into k equally sized folds. Letting each fold in turn act as unseen data (called a *validation* set), the remaining folds act as the training data used to fit the tree t_0 and to find t_α. Next, the t_α from every turn is used to obtain an estimate of the average misclassification error over the validation sets. By specifying a grid of values for α and performing k-fold CV, the optimal value for α is selected as the one corresponding to the minimum average CV misclassification error.

2.5 A Simulated Data Example

For the purpose of further exploring the properties of trees, in this section a classification tree is fit to simulated data. Following Hastie *et al.* (2009), the

[1] The k here does not refer to a class for the output, but to the number of folds used in k-fold cross-validation.

simulated data were created by first generating ten means $\boldsymbol{m}_{b1}, ..., \boldsymbol{m}_{b10}$ from a bivariate normal distribution $N((1,0)^T, I)$, where I is the identity matrix. Similarly, an additional ten means $\boldsymbol{m}_{o1}, ..., \boldsymbol{m}_{o10}$ were generated from $N((0,1)^T, I)$. A hundred observations from the blue class were generated as follows: for each observation, a mean $\boldsymbol{m}_b$ was selected at random from $\{\boldsymbol{m}_{bi}, i = 1, ..., 10\}$, and an observation was drawn from $N(\boldsymbol{m}_b, I/5)$. The above data sampling mechanism leads to a mixture of Gaussian clusters for the blue class (Hastie *et al.*, 2009). A similar procedure was then followed for the orange class (*i.e.* sampling $\boldsymbol{m}_o$ from $\{\boldsymbol{m}_{oj}, j = 1, ..., 10\}$, and an observation from $N(\boldsymbol{m}_o, I/5)$). Figure 2.2 displays a plot of the simulated data together with the corresponding Bayes decision boundary.

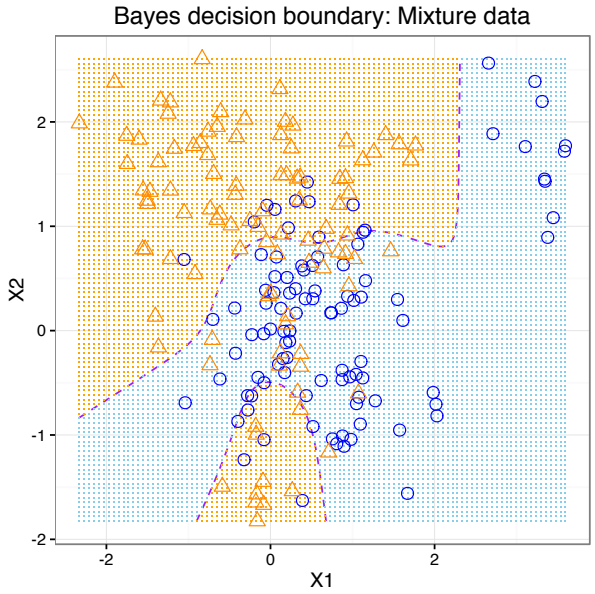

Figure 2.2: Simulated mixture data: *The dashed purple line represents the Bayes decision boundary.*

The boundary is non-linear and rather "smooth", as opposed to having sharp edges. It is clear from Figure 2.2 that even the optimal (Bayes) boundary does require perfect separation of the training data to achieve good generalisation error. The decision boundary of a classification tree is presented in the left panel of Figure 2.3.

An additional *test* set (new unseen data) of 10,000 observations were used to estimate the test error of the fitted tree. The tree decision boundary approximation of the Bayes boundary is quite rigid, which is to be expected using only orthogonal splits. The Bayes error rate on the test set is 21.5%, whereas the classification tree achieved an error rate of 26.2%. The corresponding tree

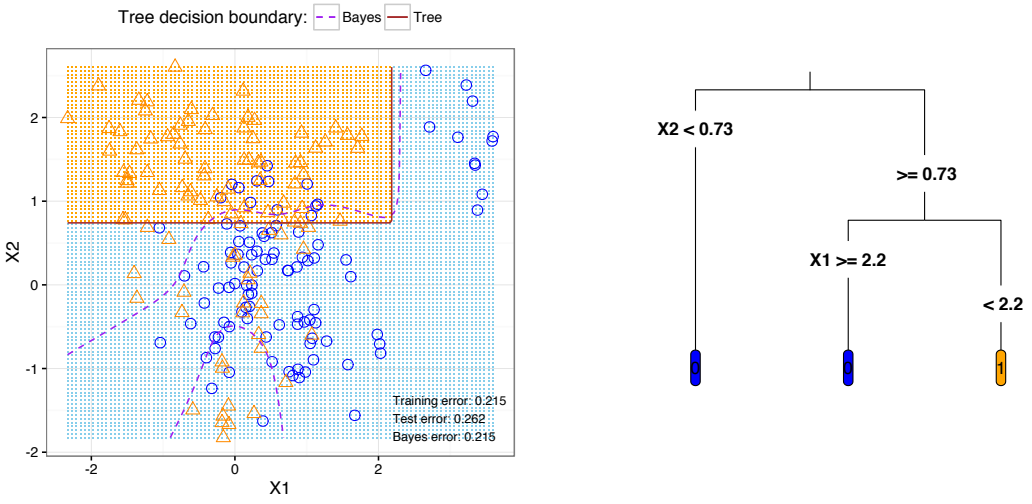

Figure 2.3: Classification tree fitted to the mixture data: *The decision boundary is represented by the solid brown line in the left panel.*

diagram showing the rules to obtain the decision boundary is displayed in the right panel of Figure 2.3. Any data point having a value lower than 0.73 for X_2 is classified as blue, otherwise the value of X_1 is inspected. Observations with X_1 values higher than 2.2 are again classified to the blue class, otherwise they are classified to the orange class.

In Section 1.3.4 it was mentioned that trees generally have high variance. In other words, trees are fairly sensitive to changes in the data. In contrast, linear models such as logistic regression tend to be more stable. To illustrate this, Figure 2.4 shows the decision boundary for fully grown as well as optimally pruned trees compared to logistic regression in the case of three training data sets sampled as described in the beginning of this section. The top row of Figure 2.4 corresponds to fully grown trees, the middle row to pruned trees and the bottom row to logistic regression. From left to right the columns of 2.4 represent the different training data sets. As previously alluded to, the slope of the decision boundary of the logistic regression model in the bottom row of Figure 2.4 changes only slightly as changes in the data are observed. In contrast, both fully grown and pruned trees (in the top and middle rows) have a high level of variability with respect to the shape of their respective decision boundaries across the training data samples. However, with the former being much more variable than the latter, it is clear that cost-complexity pruning helps to alleviate some of the variance.

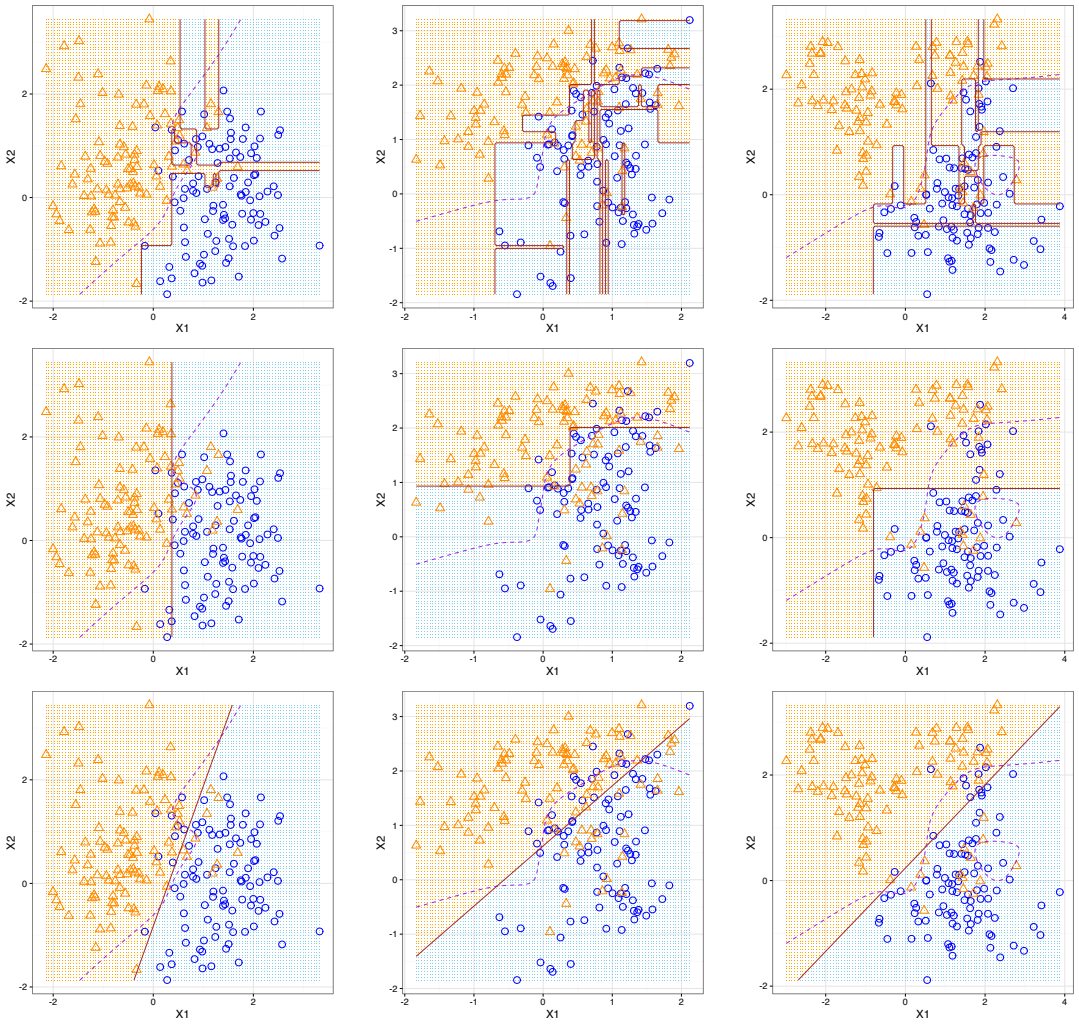

Figure 2.4: Changes in decision boundary as a result of changes in the data: top row: *fully grown classification trees*; middle row: *optimally pruned classification trees*; bottom row: *logistic regression classifier*.

2.6 Concluding Remarks

In this chapter classification trees were discussed using the CART approach of recursive binary partitioning of the input space. A tree is constructed by recursively splitting the input space into two regions based on a search over different split-points for each variable and by finding the optimal variable and split-point pair based on the sum of child node impurities.

The size of the tree controls the bias-variance trade-off. In order to avoid overfitting, a possible strategy is to start with fully grown trees and to then find the optimal tree size using cost-complexity pruning.

However, both maximum-sized and pruned trees tend to suffer more from variance than from bias. Furthermore, by using orthogonal splits the decision boundary of a tree is limited in its ability to approximate the Bayes boundary due to its rigidity.

Although limiting the performance of a single tree, the aforementioned aspects cannot merely be circumvented by combining multiple tree classifiers, but can even serve to improve the performance of the ensemble as a whole. In the next chapter, ensemble learning is discussed together with the characteristics that make trees ideal candidates for ensemble learning algorithms. In this regard, the road map forward is presented in Figure 2.5.

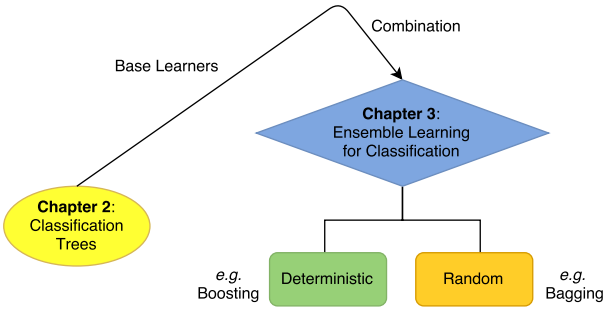

Figure 2.5: Road map to Chapter 3: Using classification trees as base learners to create ensemble classification algorithms.

Chapter 3

Ensemble Learning for Classification

The focus of this chapter is the idea of combining several models to create an ensemble classifier. Ensemble learning can be subdivided into deterministic ensembles and non-deterministic ensembles. The former subdivision is discussed in Section 3.2 with an example, namely boosting. Non-deterministic ensembles and an example known as bagging are the topic of Section 3.3. Finally, Section 3.4 discusses the appropriateness of trees as base learners for an ensemble with Section 3.5 providing some concluding remarks regarding ensemble learning.

3.1 Introduction

The idea of ensemble learning for classification is to combine multiple classifiers (base learners) in a clever way in order to create a more powerful ensemble classification algorithm. Each approach towards creating such an ensemble can be motivated as either attempting to reduce the bias or the variance of the final classifier, or to reduce both the bias and variance. For example, since a common aspect of ensemble classifiers is aggregation, many of them result in an appreciable reduction in variance (as loosely motivated in 1.3.6). Ensemble methods are occasionally referred to as *dictionary* methods since they involve creating linear combinations of a set of base learners selected from a large "dictionary". A generic ensemble classification algorithm is given in Algorithm 1 (Friedman and Popescu, 2008). The relevant notation pertaining to the algorithm is as follows:

- $b(\boldsymbol{x}, \Theta)$: a base learner characterised by the argument Θ. For example, in classification trees Θ represents the terminal nodes, *i.e.* $\Theta =$

$\{R_m, \boldsymbol{j}_m, \boldsymbol{s}_m, m = 1, ..., M\}$, the terminal nodes of the tree with their corresponding splitting variables and split-points.

- $S_b(\eta)$: the b^{th} sample (taken with or without replacement) of size $\eta \cdot N$ from the training data, where $\eta \in (0, 1]$.
- ν: a *memory* parameter, controlling the amount of past information that is incorporated at each step, where $\nu \in [0, 1]$.

Algorithm 1 Ensemble Learning for Classification

1. Let $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$, where C can take on values $k \in \{1, ..., K\}$ and initialise $g_0(\boldsymbol{x}) = 0$.
2. For $b = 1$ to B:
   a) Sample $S_b(\eta)$ from Ω_{tr}.
   b) Find $\Theta_b = \arg\min_\Theta \sum_{\boldsymbol{x}_i \in S_b(\eta)} L(c_i, g^\nu_{b-1}(\boldsymbol{x}_i) + b(\boldsymbol{x}_i, \Theta))$.
   c) Set $g_b(\boldsymbol{x}) = b(\boldsymbol{x}, \Theta_b)$.
   d) Update $g^\nu_b(\boldsymbol{x}) = g^\nu_{b-1}(\boldsymbol{x}) + \nu \cdot b(\boldsymbol{x}, \Theta_b)$.
3. Let $g^*_b(\boldsymbol{x}) = g_b(\boldsymbol{x})$ if $\nu = 0$. Otherwise let $g^*_b(\boldsymbol{x}) = g^\nu_b(\boldsymbol{x})$. The ensemble classifier is

$$\bar{g}_{EL}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} I(g^*_b(\boldsymbol{x}) = k).$$

At each step $b = 1, ..., B$, ensemble learning proceeds by finding the base learner characterised by the argument Θ that minimises the loss over all the points in the sample $S_b(\eta)$. The next addition to the ensemble is then either just the current base learner, or a fraction of all the base learners that have been fitted up until the current step. It is important to clarify that the addition operator for classification acts on the estimated posterior probabilities $\hat{P}_{S_b(\eta)}(C = k|\boldsymbol{x})$, where $k \in \{1, ..., K\}$. At Step 3 each member of the ensemble casts a vote for a class $k = \arg\max_k \hat{P}_{S_b(\eta)}(k)$, and the final classifier is the majority vote of all the members. The size of the fraction ν essentially controls the amount by which the algorithm avoids base learners at the current step that are similar to those that have come before. For example, if ν is large (for example $\nu = 1$), the base learner most likely to be selected (line $2(a)$) at Step b is one that does not contain the information that is already contained

in $g_{b-1}^{\nu}(\boldsymbol{x})$. In contrast, note that if $\nu = 0$ it implies that $g_b^{\nu}(\boldsymbol{x}) = 0$ for all $b = 1, ..., B$. Therefore at each step, the fitted base learner is found completely independent of all other previous base learners.

Implicit in Algorithm 1 is a mechanism for adjusting the extent to which the ensemble learner is obtained in a deterministic way. Roughly speaking in this text an ensemble is said to be a *deterministic* ensemble if no randomness is induced during the creation of the ensemble. In other words, if the algorithm were to be run twice on the same training data, the two resulting deterministic classifiers will produce the same predictions. On the opposite side lies a *non-deterministic* or *random* ensemble, which has randomness introduced at some stage of the ensemble creation process.

3.2 Deterministic Ensembles

Deterministic ensembles are devoid of any sources of randomness. At each stage the same training data is used and the base learner is fitted using an adaptive but deterministic strategy. An adaptive step is crucial for the success of the ensemble, since otherwise the same classifier is produced at each step and no new information is obtained. In Algorithm 1 this translates to an implicit adaptive strategy within the fitted base learner $b(\boldsymbol{x}, \Theta)$, as well as to restrictions on the parameter η and on the sampling strategy. Specifically, to obtain the same sample size it must be that $\eta = 1$ and that the sampling is performed without replacement. A prime example of a classifier built from a deterministic ensemble of base learners is boosting.

3.2.1 Boosting

Boosting is a powerful learning technique that uses deterministic ensemble learning to produce highly accurate prediction models. The base learners in boosting are often "weak" classifiers characterised by their prediction accuracy being slightly better than random guessing. An example of a weak classifier is a classification tree with a single split at the root node, referred to as a *stump*. Using stumps as base learners, the original proposal for a boosted model (called *AdaBoost*, short for adaptive boosting), constructs a sequence of classifiers trained on data that are iteratively "adapted" (Freund and Schapire, 1995). The adaptive step at each iteration causes the current stump to be fitted to focus more on the observations that were misclassified by the stump in the previous iteration. The final prediction model then takes the form of a weighted sum of the above adapted weak learners.

In more detail, consider the training set $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$, where the output $C \in \{-1, +1\}$ and where each observation receives an initial weight

$w_i = 1/N$. The first step of the AdaBoost procedure is to obtain a classifier $t(\boldsymbol{x}, \Theta)$ by fitting a stump to the weighted data[1] (Hastie *et al.*, 2009). Here b indexes the current number of iterations starting at $b = 1$ and ending at some specified total number of boosting iterations B. The contribution from the b^{th} fitted stump towards the final classifier is based on a weighted version of the training error and is computed as

$$\alpha_b = log\left(\frac{1 - Error_b}{Error_b}\right), \tag{3.2.1}$$

where the weighted training error is

$$Error_b = \frac{\sum_{i=1}^{N} w_i I(t(\boldsymbol{x}_i, \Theta_b) \neq c_i)}{\sum_{i=1}^{N} w_i}. \tag{3.2.2}$$

The relationship between α_b and $Error_b$ is given in the left panel of Figure 3.1.

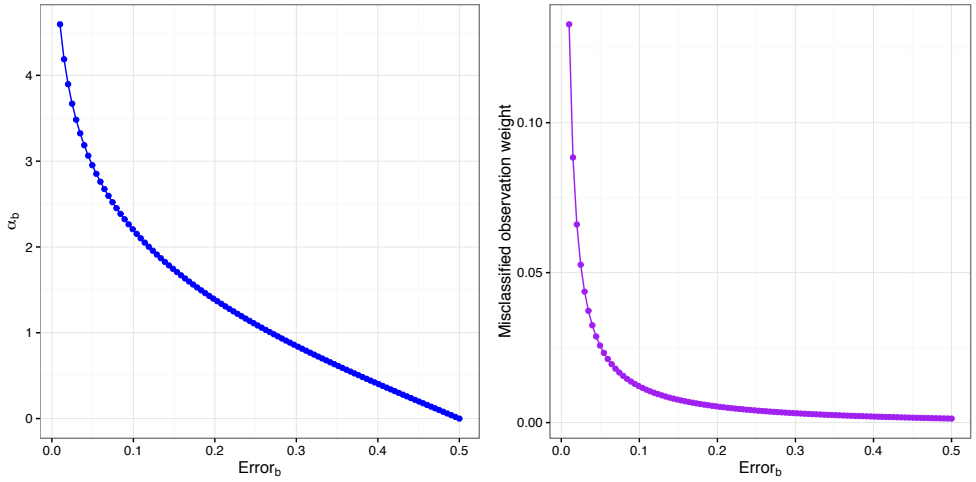

Figure 3.1: Improving the accuracy of trees with the AdaBoost algorithm.

It is clear that each weak learner's contribution to the final classifier is a monotone decreasing function of the weighted training error. Therefore fitted stumps that perform well on the training data play a bigger part in determining the final ensemble classifier. In the second step of AdaBoost the observation

[1]The weights have an effect on the construction of the tree through weighting the impurity measures computed when searching for an optimal split.

weights are updated as follows:

$$w_i^b = w_i^{b-1} \cdot e^{\alpha_b I(t(\boldsymbol{x}_i, \Theta_b) \neq c_i)}, \tag{3.2.3}$$

where $I(t(\boldsymbol{x}_i, \Theta_b) \neq c_i)$ is an indicator function ensuring that only the weights of the misclassified observations are increased by a factor e^{α_b}. The right panel of Figure 3.1 shows that there is a negative relationship between the increase in observation weight and classification error. This might seem counter intuitive. However consider a classifier with a training error rate of 0.5. In this case the error is more likely to be attributed to a poor classifier than to the data being intrinsically difficult to classify. Hence each observation receives only a slight increase in weight. On the other hand, if a classifier obtained an error rate close to zero, with for example only one or two misclassified points, it makes intuitive sense to force the classifier in the next iteration to concentrate more on correctly classifying these few points. This is done by increasing the weights of these few misclassified points by a large factor. After B iterations of the aforementioned steps, the final AdaBoost classifier takes the form

$$\bar{g}_{ada}(\boldsymbol{x}) = sign\left[\sum_{b=1}^{B} \alpha_b t(\boldsymbol{x}, \Theta_b)\right]. \tag{3.2.4}$$

In Figure 3.2 AdaBoost is compared to a single tree stump and to a fully grown tree in terms of their test error on simulated data (in this text referred to as the *elemStat* data).[2]

From Figure 3.2 it is seen that a single stump achieved an error rate only slightly better than random guessing, whereas the maximum sized tree achieved an error rate just below 30%. In stark contrast AdaBoost managed to reach a test error rate as low as 10% after only 100 iterations and kept improving to a test error of about 5% at 600 iterations.

With the aim of showing AdaBoost to be a special case of a larger boosting family, as well as pointing out its connection with the ensemble learning framework presented in Section 3.1, the procedure is examined from an alternative view point. In general, boosting can be described as a method for function estimation (Friedman, 2001). In this framework boosting estimates the population minimiser of a specified convex loss function within a function

[2]The simulated data is from independent variables $X_1, ..., X_{10} \sim N(0, 1)$ with $Y = 1$ if $\sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5) = 9.34$, otherwise $Y = -1$ (Hastie *et al.*, 2009). The classes have equal sizes, with 2000 observations used for training and $10,000$ for testing.

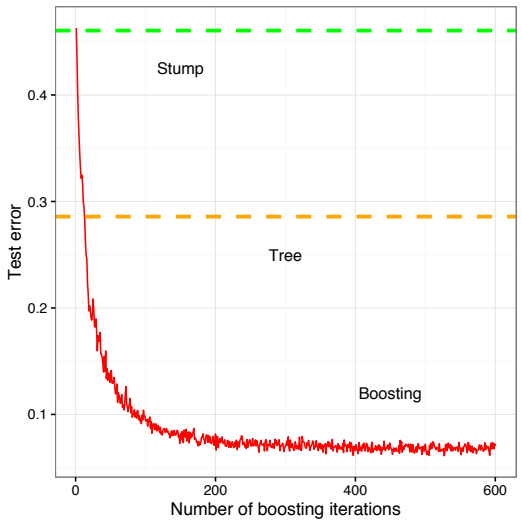

Figure 3.2: Test Error rates on elemStat data for a stump, for a fully grown tree and for AdaBoost.

space constrained by the chosen base learner. This is known as *functional gradient boosting*. In more detail, consider the problem of estimating the function

$$\bar{g}^*_{boost}(\boldsymbol{x}) = \arg\min_{g(\boldsymbol{x})} E\big[L(c_i, g(\boldsymbol{x}))\big], \tag{3.2.5}$$

where $L(\cdot,\cdot)$ is a convex and differentiable loss function with respect to $g(\boldsymbol{x})$. A well known approach towards finding a parameter vector minimising a convex function is called *steepest descent*. Starting with an initial guess, steepest descent aims to take small "steps" in the direction of the negative gradient. Here each step updates the estimate of the parameters by adding a fraction of the negative gradient evaluated at the current estimate. Since the function is convex, each step will ensure that the new estimate is an improvement over the last. The procedure is continued until a negligible change in the slope indicates a global optimum to have been reached. The notation $g(\boldsymbol{x})$ in (3.2.5) refers to a function that needs to estimated and not a numeric parameter vector. The clever idea that underlines functional gradient boosting is to use a base learner to approximate the negative gradient computed at each step. More specifically, the negative gradient components at step b are

$$u_i = -\frac{\partial}{\partial f} L(c, g(\boldsymbol{x})), i = 1, ..., N, \tag{3.2.6}$$

each evaluated at the current estimate $g_{b-1}(\boldsymbol{x}_i)$. An approximation of the negative gradient is obtained by fitting a base learner $b(\boldsymbol{x}, \Theta_b)$ to the set

$\{(\boldsymbol{x}_i, u_i), i = 1, ..., N\}$. The update is then of the form

$$g_b(\boldsymbol{x}) = g_{b-1}(\boldsymbol{x}) + \nu \cdot b(\boldsymbol{x}, \Theta_b), \tag{3.2.7}$$

where in the boosting literature, $0 < \nu \leq 1$ is called the *step-length factor* or the *learning rate* (Bühlmann and Hothorn, 2007). After a specified number of steps (B), the final classifier is

$$\bar{g}_{boost}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} I(g_b(\boldsymbol{x}) = k). \tag{3.2.8}$$

Note that the classifier in (3.2.8) is constrained to the function class of the chosen base learner.

Functional gradient boosting allows for several different boosting algorithms to be constructed by simply specifying a different loss function and/or base learner. For example, selecting stumps as base learners and using exponential loss results in the AdaBoost procedure. In the regression setting, using the squared-error loss $L_{SE}(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$, results in the negative gradient at each step to be twice the current residuals $u_i = 2(y_i - f_{b-1}(\boldsymbol{x}_i)), i = 1, ..., N$. From here the base learner is fit to the residuals and the update takes the usual form $f_b(\boldsymbol{x}) = f_{b-1}(\boldsymbol{x}) + \nu \cdot b(\boldsymbol{x}, \Theta_b)$. The final estimate is then given as the average over of the members of the ensemble, and not a majority vote. The above procedure is known as $L_2 Boosting$. Hence by simply changing the loss function, different boosting algorithms for both regression and classification can be constructed.

Finally, any gradient boosting approach can be viewed as a special case of the ensemble framework in Algorithm 1. Specifying a loss function, sampling without replacement with $\eta = 1$ and obtaining base learners by way of the negative gradient fully defines a specific boosting procedure.[3]

3.3 Random Ensembles

There are two options regarding the introduction of randomness into an ensemble learning procedure. The first is to choose a sample strategy such that the sample drawn at each step is either a sample of size N taken *with* replacement, or a sample drawn without replacement where $\eta < 1$. This means that at each

[3]Stochastic gradient boosting is a form of boosting that removes the restrictions mentioned here. However this of course still remains a special case of Algorithm 1.

step, the sample used to fit the base learner will be different from previous samples. In addition, a second source of randomness implicit in the ensemble creation can stem from the way in which the base learner is constructed. Therefore, random ensembles are created via Algorithm 1 by either:

1. Selecting a deterministic base learner with $\eta = 1$ and sampling with replacement;
2. Selecting a deterministic base learner with $\eta < 1$ and sampling with or without replacement;
3. Selecting a randomised base learner with $\eta = 1$ and sampling with replacement;
4. Selecting a randomised base learner with $\eta < 1$ and sampling with or without replacement.

An example of a random ensemble learning algorithm is *bagging*.

3.3.1 Bagging

In an attempt to reduce the variance of a model and thereby to potentially improve its accuracy, Breiman (1996*a*) developed a procedure to combine multiple base learners using bootstrap aggregation, or "bagging". Bagging essentially fits many base learners by using bootstrap samples of the training data to produce a large number of model estimates. A prediction for a new case is then simply the average of all the outputs produced by the base learners, or in the case of classification, the majority vote.

In more detail, consider the training set $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i, i = 1, 2, ..., N\}$ and some learning algorithm to construct a classifier $g(\boldsymbol{x})$. A set $\{\Omega_1^*, \Omega_2^*, ..., \Omega_B^*\}$ can be generated using bootstrap sampling. Subsequently, the set of classifiers $\{g_{\Omega_1^*}(\boldsymbol{x}), g_{\Omega_2^*}(\boldsymbol{x}), ..., g_{\Omega_B^*}(\boldsymbol{x})\}$ can each be trained on a bootstrap replicate. Now suppose $\boldsymbol{x}$ can be classified as belonging to one of K classes and let

$$\hat{P}_{bag}(k|\boldsymbol{x}) = \sum_{b=1}^{B} I(g_{\Omega_b^*}(\boldsymbol{x}) = k). \tag{3.3.1}$$

Then bagging classifies $\boldsymbol{x}$ to the class for which $\hat{P}_{bag}(k|\boldsymbol{x})$ is a maximum, where $k = 1, ..., K$, and the corresponding aggregated (majority vote) classifier is

$$\bar{g}_{bag}(\boldsymbol{x}) = \arg\max_k \hat{P}_{bag}(k|\boldsymbol{x}). \tag{3.3.2}$$

To illustrate the effect of bagging classification trees on the obtained decision boundary, a bagged model consisting of 100 trees is fitted to the mixture data and shown in the top right panel of Figure 3.3.

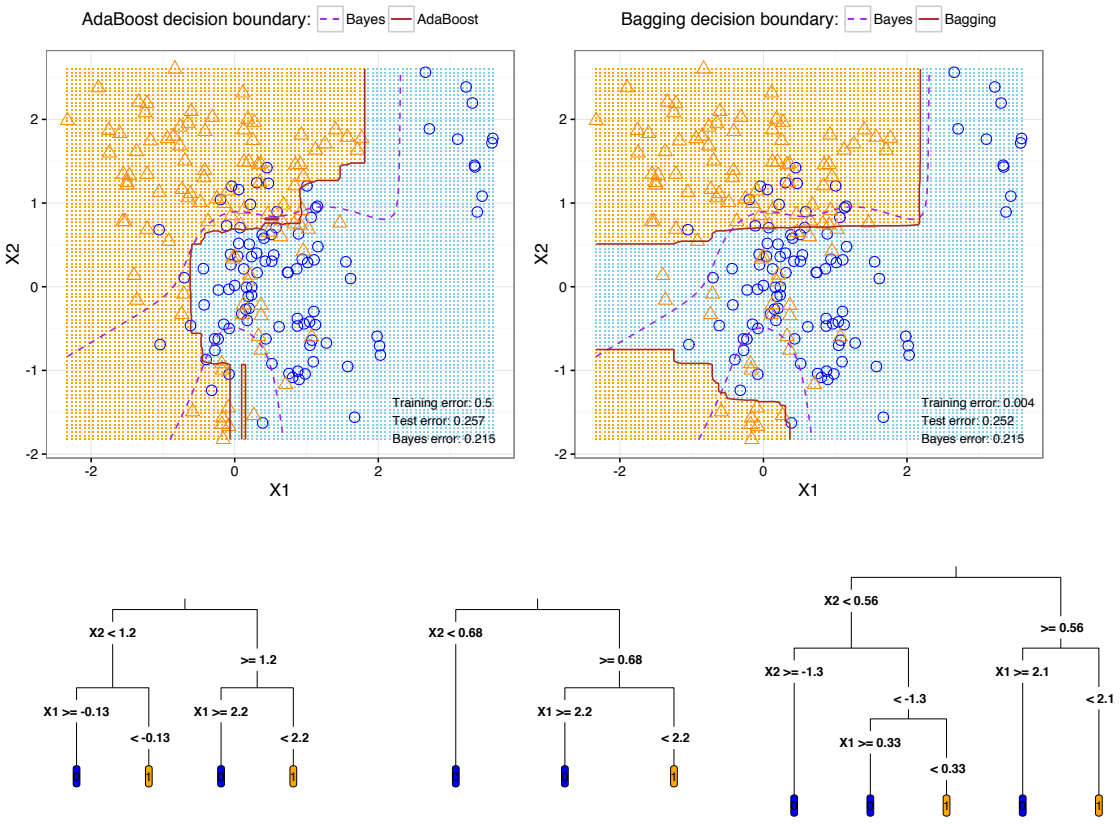

Figure 3.3: Top: *AdaBoost compared to bagging using* 100 *classification trees fitted to the mixture data: the decision boundary is represented by the solid brown line.* Bottom: *A random sample of three classification trees from the bagged ensemble*

This boundary is compared to the boundary produced by AdaBoost (also using 100 trees) as shown in the left panel. Both algorithms result in a departure from the axis-parallel decision rules produced by a single classification tree. This is of course induced by the aggregation of decision rules from several trees. (Strictly speaking the decision boundary produced by both procedures is still axis-parallel, however it is much more finely grained.) Boosting achieves a test error rate of 25.7%, whereas bagging achieves a test error rate of 25.2%, which constitutes an average improvement of less than one percent (0.7%) for

a single tree. Although in this example the accuracy of the ensemble learners are not remarkably higher than that of a single tree, the situation is likely to change given a more complex data set (refer to Figure 3.2).

Interestingly, the boosted classifier obtained an error rate of 50% on the training data, yet the test error as well as the shape of the decision boundary is reasonable. A possible explanation for this pertains to the previously discussed view of boosting as a functional gradient descent algorithm. Boosting has as main objective the minimisation of a convex loss function and not minimising the training error. This might explain why, even though the training error remained high, the test error did not suffer.

The bottom panel of Figure 3.3 shows three randomly drawn classification trees from the ensemble used to create the bagged classifier. Each tree is different, although the tree growing algorithm used to create each tree is intrinsically deterministic given a particular data set. This illustrates the effect of bootstrap sampling and its ability to create diversity among base learners.

The bagging algorithm presented above is (along with boosting) another example of a special case of Algorithm 1. Besides the addition of randomness in bagging, boosting and bagging differ fundamentally in the sense that bagging constructs base learners independently at each step (*i.e.* $\nu = 0$), whereas boosting is an iterative procedure that aims to learn from previous iterations ($\nu > 0$). Therefore, bagging implements Algorithm 1 where sampling is done with replacement, with $\eta = 1$ and $\nu = 0$.

3.4 Trees: Popular Base Learners for Ensembles

Both boosting and bagging as examples of ensemble learners use trees as base learners for constructing the ensemble, although there are no formal restrictions regarding this choice. Furthermore, this is not an arbitrary selection specific to this text, but the preferred base learner at the inception of each algorithm. The following question arises: why are trees good base learners?

The answer can be divided into two parts. Certain characteristics of a tree advocate its use in terms of being a good individual learner, while other characteristics are important to enable good performance of the ensemble. Starting with the former, trees have the following properties that render them attractive at the level of an individual base learner (Hastie *et al.*, 2009):

- Natural handling of both quantitative and qualitative data types.

- Natural handling of missing values.
- Robustness to outliers.
- Insensitivity to monotone transformations of the input variables.
- Ability to handle large data sets (large N).
- Implicit variable selection, enabling one to deal with many irrelevant input variables.

This is an impressive list. However, although trees have low bias, they suffer from high variance and as a consequence typically have poor generalisation performance.

It is this high variance that render trees appropriate base learners for ensembles. In the case of deterministic ensembles, starting out with already low biased trees, the variance can be reduced (sometimes dramatically) through aggregation. It is however in random ensembles that the instability of trees really come into play. By being very sensitive to changes in the data as well as to algorithmically induced randomness, trees are capable of producing a *diverse* ensemble of base learners. The diversity serves as a way to decorrelate the set of trees from each other, such that the variance of the ensemble is reduced even further. In fact, "memoryless" random ensembles where the base learner is restricted to be some sort of tree has become such an active area of research, that this algorithm class has received its own name, *random forests* (RFs).

3.5 Concluding Remarks

A way of improving classification procedures is offered through aggregation of single classifiers into an ensemble learner. In this way a substantial reduction in variance generally leads to improved generalisation.

Furthermore, a general ensemble framework allows for the construction of several different ensemble classification procedures through the choice of an appropriate loss function, base learner, type of sampling procedure and the degree of dependence between iterations. However, with respect to determinism there exists a dichotomy within the ensemble framework. Deterministic ensembles are invariable across multiple runs and work through adaptation, whereas random ensembles change from one run to the next due to different sources of artificially induced randomness.

Random ensembles use trees as base learners, which are highly sensitive to

changes in the data and to randomness injected through the design of an algorithm. The effect is base learners that tend to be less correlated and which facilitates a further reduction in variance. This is the approach followed by random forests.

In the next chapter, random forests are discussed within this general framework of ensemble learning algorithms that make use of independently constructed randomised trees as base learners. The road map forward is presented in Figure 3.4.

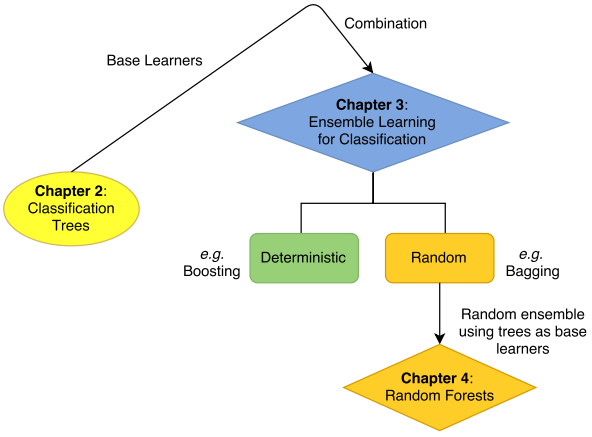

Figure 3.4: Road map to Chapter 4: Random forests as ensemble learning algorithms using independently constructed randomised trees as base learners.

Chapter 4

Random Forests

Random forests reside under the umbrella of random ensemble classifiers, however they exclusively use trees as base learners. In this chapter, a general perspective on random forests is given in Section 4.1, with early developments discussed in Section 4.2. The generalisation error of a random forest and its resistance to overfitting is explored in Sections 4.3 and 4.4 respectively. A popular example of a random forest is Breiman's Forest-RI which is the focus of Section 4.5, with the use of out-of-bag samples for error estimates and variable importance discussed in Section 4.6. Finally in this chapter, concluding remarks are given in Section 4.7.

4.1 Introduction

In his seminal paper on random forests, Breiman (2001*a*) pointed out the common element in most tree based random ensemble learning algorithms. Specifically, for the b^{th} tree to be constructed, each algorithm generates a random vector Θ_b, independent and identically distributed from the past random vectors $\Theta_1, ..., \Theta_{b-1}$, and produces a classifier $t(\boldsymbol{x}, \Theta_b)$. After a large number of trees have been constructed in this way, a majority vote is used to classify a new observation. Breiman dubbed all the procedures characterised by the aforementioned steps "random forests" and provided the following formal definition (notation slightly changed).

Definition 1.1 A random forest is a classifier consisting of a collection of tree-structured classifiers $\{t(\boldsymbol{x}, \Theta_b), b = 1, ..., B\}$ where the $\{\Theta_b\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\boldsymbol{x}$.

The caveat in Definition 1.1 is that it requires the argument vector Θ_b to be independently distributed, hence in addition to the base learner restriction, random forests have no memory parameter (*i.e.* $\nu = 0$) when cast in the ensemble learning framework. The restriction might not feel justified, however in the case of an algorithm such as stochastic gradient boosting (a version of boosting with random sampling), the definition will ensure that this type of boosting still resides under the boosting umbrella and not under that of random forests.

4.2 Early Developments

Even prior to the bagging proposal, in a paper by Ho (1995) called "Random Decision Forests", a method was proposed to address the poor generalisation performances of classification trees which are grown to arbitrary levels of complexity. The proposed method uses classification trees with splits on linear combinations of the input variables and constructs multiple trees using only a random subset of the input variables for each tree. Using a classifier based on the aggregation of the posterior probability estimates from each tree, the complementary generalisations from the trees are utilised to produce a more powerful classifier.

In more detail, suppose B trees are constructed, each tree only using a random subset of the input variables and let $R_b(\boldsymbol{x})$ denote the terminal node assigned to $\boldsymbol{x}$ by tree $t_b, b = 1, ..., B$. The posterior probability that $\boldsymbol{x}$ belongs to the class k, where $k = 1, ..., K$, is given by

$$P(k|R_b(\boldsymbol{x})) = \frac{P(k, R_b(\boldsymbol{x}))}{\sum_{l=1}^{K} P(l, R_b(\boldsymbol{x}))}. \tag{4.2.1}$$

This probability can be estimated as the fraction of the observations in $R_b(\boldsymbol{x})$ that are assigned to class k. Then using the obtained probability estimates from all the trees gives the aggregated estimated posterior probability as

$$\hat{P}_{RS}(k|\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{P}(k|R_b(\boldsymbol{x})), \tag{4.2.2}$$

where $\hat{P}(k|R_b(\boldsymbol{x}))$ is the estimated posterior probability of k given $\boldsymbol{x}$ for the b^{th} tree. The aggregated classifier is then $\bar{t}_{RS}(\boldsymbol{x}) = \arg\max_k \hat{P}_{RS}(k|\boldsymbol{x})$.

Ho (1995) provides a geometric interpretation of the classifier $\bar{t}_{RS}(\boldsymbol{x})$. Each

terminal node can be seen as defining some neighbourhood around the observations assigned to that node, inside a random subspace of the input variables. Aggregating the posterior probabilities for a given observation $\boldsymbol{x}$ over all the neighbourhoods may then be viewed as an approximation of the posterior probabilities in the entire input space.

Independently, Amit and Geman (1997) also proposed a method to estimate posterior probabilities using random subspaces of the input variable space and aggregating over the different subspace specific probabilities. Their aim in the paper was to improve the performance of a classifier in recognising a handwritten digit. In their proposal, an observation is classified by taking the mode of the posterior distribution over "shape" classes. However, the fundamental difference between their approach and that by Ho (1995), is that Amit and Geman (1997) considered a new random subspace at each node split while constructing a classification tree instead of selecting a single random subspace for the entire tree construction. The classifier produced in this way is essentially identical to the one in (4.2.2), but with different posterior probability estimates $\hat{P}(k|R_b(\boldsymbol{x})), b = 1, ..., B$ used in the aggregation. It was this paper that Breiman would later mention as having been influential in his thinking when writing his paper on random forests (Breiman, 2001*a*).

In a later paper by Ho (1998), a comparison was made between his random subspace method proposed earlier (Ho, 1995) and constructing classification trees using training set resampling methods such as the bootstrap. This was in an attempt to find the best method to construct a tree based classifier that maintains high accuracy on the training set while monotonically decreasing the generalisation error as it grows in complexity. Another approach is the one proposed by Dietterich (1998), which is closely related to a proposal by Kwok and Carter (1990). The idea is to replace the optimal variable and split point with a random selection among the top 20 ranked variables and corresponding split points at each node when growing trees (the top 20 might include the same variable more than once, but with a different split point).

4.3 Generalisation Error of a Random Forest

Following Breiman (2001*a*), in this section a derivation of the upper bound for the generalisation error of a random forest is presented. First, define the margin function of a random forest obtained from a training set to be

$$\frac{1}{B}\sum_{b=1}^{B} I(t(X,\Theta_b) = c) - \max_{c \neq k} \frac{1}{B}\sum_{b=1}^{B} I(t(X,\Theta_b) = k), \tag{4.3.1}$$

where $\{t(X,\Theta_b), b = 1, ..., B\}$ denotes an ensemble of classification trees. The

margin can be seen as a measure of confidence in the classification, since it represents the amount by which the average number of votes for the correct class exceeds the average number of votes for the most likely class other than the correct class. A larger margin increases the confidence in an observation being classified to the correct class and a smaller margin reduces this confidence. Note that a classifier unable to classify to the correct class more often than any other class will have a negative margin. From the Strong Law of Large Numbers, as the number of trees increases, for almost surely the margin converges to

$$mg(X,C) = P_{\Theta}(t(X,\Theta) = C) - \max_{C \neq k} P_{\Theta}(t(X,\Theta) = k). \tag{4.3.2}$$

(A proof of 4.3.2 is presented in Section 4.4). Breiman (2001*a*) defines the strength of a set of tree classifiers $\{t(\boldsymbol{x},\Theta)\}$ as

$$S = E_{X,C}\big[mg(X,C)\big]. \tag{4.3.3}$$

The generalisation error of a random forest is then given by

$$Err^* = P_{X,C}(mg(X,C) < 0), \tag{4.3.4}$$

which is simply the probability of a negative margin. Restricting the derivation to the case where S is greater than zero, it is possible to write

$$\begin{aligned} Err^* &= P_{X,C}(mg(X,C) - E_{X,C}\big[mg(X,C)\big] + E_{X,C}\big[mg(X,C)\big] < 0) \\ &= P_{X,C}(S - mg(X,C) \geq S) \\ &= P_{X,C}\big((mg(X,C) - S)^2 \geq S^2\big) \\ &= P_{X,C}\big(\sqrt{(mg(X,C) - S)^2} \geq \sqrt{S^2}\big) \\ &= P_{X,C}\big(|mg(X,C) - S| \geq S\big), \end{aligned}$$

since $S \geq 0$ implies $|S| = S$. Next consider a random variable Z with $E(Z) = \mu > 0$, Chebychev's inequality states that for any $\alpha > 0$,

$$P(|Z - \mu| \geq \alpha) \leq \frac{Var(Z)}{\alpha^2}. \tag{4.3.5}$$

Plugging $Z = mg(X, C)$ and $\mu = \alpha = S$ into (4.3.5), the following upper bound for the generalisation error is obtained:

$$Err^* \leq \frac{Var\big[mg(X, C)\big]}{S^2}. \tag{4.3.6}$$

To gain more insight into the variance of the margin let

$$\neg C = \arg\max_{C \neq k} P_\Theta(t(X, \Theta) = k) \tag{4.3.7}$$

such that

$$\begin{aligned} Var\big[mg(X, C)\big] &= Var\Big[P_\Theta(t(X, \Theta) = C) - P_\Theta(t(X, \Theta) = \neg C\Big] \\ &= Var\Big\{E_\Theta\big[I(t(X, \Theta) = C) - I(t(X, \Theta) = \neg C\big]\Big\}. \end{aligned}$$

Breiman (2001*a*) denotes $I(t(X, \Theta) = C) - I(t(X, \Theta) = \neg C)$, the so-called raw margin function, by $rmg(X, C, \Theta)$. The raw margin can be interpreted as the margin computed over some finite data sample and the margin is then simply the expected value of the raw margin. Hence,

$$Var\big[mg(X, C)\big] = Var\Big\{E_\Theta\big[rmg(X, C, \Theta)\big]\Big\}. \tag{4.3.8}$$

Moreover, consider two independent and identically distributed random vectors Θ and Θ', then

$$mg(X, C)^2 = E_{\Theta,\Theta'}\big[rmg(X, C, \Theta)rmg(X, C, \Theta')\big], \tag{4.3.9}$$

which is a result of the identity that for any function h, it holds that

$$E_\Theta\big[h(\Theta)\big]^2 = E_{\Theta,\Theta'}\big[h(\Theta)h(\Theta')\big]. \tag{4.3.10}$$

To simplify notation, let $mg(X, C), rmg(X, C, \Theta)$ and $rmg(X, C, \Theta')$ simply be written as $mg, rmg(\Theta)$ and $rmg(\Theta')$ respectively. Using (4.3.9), it is now

possible to denote the variance of the margin by

$$\begin{aligned} Var(mg) &= E_{X,C}(mg^2) - \big[E_{X,C}(mg)\big]^2 \\ &= E_{X,C}\Big\{E_{\Theta,\Theta'}\big[rmg(\Theta)rmg(\Theta')\big]\Big\} - E_{X,C}\Big\{E_{\Theta}\big[rmg(\Theta)\big]\Big\}^2. \end{aligned}$$

Applying the identity in (4.3.10) to the second term in the last line, and swapping expectations yield

$$\begin{aligned} Var(mg) &= E_{\Theta,\Theta'}\Big\{E_{X,C}\big[rmg(\Theta)rmg(\Theta')\big]\Big\} - E_{\Theta,\Theta'}\Big\{E_{X,C}\big[rmg(\Theta)\big]E_{X,C}\big[rmg(\Theta')\big]\Big\} \\ &= E_{\Theta,\Theta'}\Big\{E_{X,C}\big[rmg(\Theta)rmg(\Theta')\big] - E_{X,C}\big[rmg(\Theta)\big]E_{X,C}\big[rmg(\Theta')\big]\Big\} \\ &= E_{\Theta,\Theta'}\Big\{cov_{X,C}\big[rmg(\Theta), rmg(\Theta')\big]\Big\}. \end{aligned}$$

Therefore,

$$Var(mg) = E_{\Theta,\Theta'}\Big\{\rho_{X,C}\big[rmg(\Theta), rmg(\Theta')\big]\sigma\big[rmg(\Theta))\sigma(rmg(\Theta')\big]\Big\},$$

where $\rho(\cdot,\cdot)$ is the correlation function and $\sigma(\cdot)$ is the standard deviation. Holding Θ, Θ' fixed, the mean value of the correlation between $rmg(\Theta)$ and $rmg(\Theta')$ is given by

$$\begin{aligned} \bar{\rho} &= \frac{E_{\Theta,\Theta'}\Big\{\rho_{X,C}\big[rmg(\Theta), rmg(\Theta')\big]\sigma\big[rmg(\Theta))\sigma(rmg(\Theta')\big]\Big\}}{E_{\Theta,\Theta'}\Big\{\sigma\big[rmg(\Theta)\big]\sigma\big[rmg(\Theta')\big]\Big\}} \\ &= E_{\Theta,\Theta'}\Big\{\rho_{X,C}\big[rmg(\Theta), rmg(\Theta')\big]\Big\}. \end{aligned}$$

Therefore,

$$\begin{aligned} Var(mg) &= \bar{\rho}E_{\Theta,\Theta'}\Big\{\sigma\big[rmg(\Theta)\big]\sigma\big[rmg(\Theta')\big]\Big\} \\ &= \bar{\rho}E_{\Theta}\Big\{\sigma\big[rmg(\Theta)\big]\Big\}^2 \\ &\leq \bar{\rho}E_{\Theta}\Big\{Var\big[rmg(\Theta)\big]\Big\}, \end{aligned} \tag{4.3.11}$$

since for any random variable $X, Var(X) \geq 0$ implies that $E(X)^2 \leq E(X^2)$. Furthermore,

$$\begin{aligned} E_\Theta\Big\{Var\big[rmg(\Theta)\big]\Big\} &= E_\Theta\Big\{E_{X,C}\big[rmg(\Theta)^2\big] - E_{X,C}\big[rmg(\Theta)\big]^2\Big\} \\ &= E_\Theta\Big\{E_{X,C}\big[rmg(\Theta)^2\big]\Big\} - E_\Theta(mg^2) \\ &\leq E_\Theta\Big\{E_{X,C}\big[rmg(\Theta)^2\big]\Big\} - E_\Theta(mg)^2. \end{aligned}$$

With the maximum value of the raw margin equal to 1, it follows that

$$E_\Theta\Big\{Var\big[rmg(\Theta)\big]\Big\} \leq 1 - S^2. \tag{4.3.12}$$

Finally, combining (4.3.6), (4.3.11) and (4.3.12) leads to the following upper bound for the generalisation error for a random forest:

$$Err^* \leq \frac{\bar{\rho}(1 - S^2)}{S^2}. \tag{4.3.13}$$

This shows that the two components that bound the generalisation error are the correlation between the classification trees and the strength of each tree in the random forest with respect to its raw margin function. For an ensemble, the decrease in correlation results in a reduction in variance. Although Breiman (2001*a*) notes that the bound is likely to be quite loose, it helps to somewhat more rigorously illuminate the inner workings of random forests.

4.4 Random Forests and Overfitting

Breiman (2001*a*) claims that random forests are impervious to overfitting and provides the following proof to back his assertion. Consider a fixed training set and a fixed vector Θ characterising the splitting structure of a tree classifier. The set $\{\boldsymbol{x}|t(\boldsymbol{x}, \Theta) = k\}$ for some class $k \in \{1, ..., K\}$ represents a union of hyper-rectangles, the neighbourhoods created in p-dimensional space as a result of binary partitioning. Since trees cannot be grown to infinite depth, for any tree classifier $t(\boldsymbol{x}, \Theta)$, there exists only a finite number L of such unions of neighbourhood regions, denoted here by $S_1, ..., S_L$. Define the function

$$\xi(\Theta) = l \cdot I\big(\{\boldsymbol{x}|t(\boldsymbol{x}, \Theta) = k\} = S_l\big), \tag{4.4.1}$$

where $l \in \{1, ..., L\}$. The function in (4.4.1) maps the particular structure of a tree (given a fixed training set) via $l \in \{1, ..., L\}$ to $S_1, ..., S_L$. Suppose B trees are grown using bootstrap resampling and let b_l be the average number of times that $\xi(\Theta_b) = l$, where $l = 1, ..., L$ and $b = 1, ..., B$. Then

$$\frac{1}{B}\sum_{b=1}^{B} I\big(t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b) = k\big) = \sum_{l=1}^{L} b_l \cdot I(\boldsymbol{x} \in S_l).$$

Furthermore by the Law of Large Numbers,

$$b_l = \frac{1}{B}\sum_{b=1}^{B} I(\xi(\Theta_b) = l) \underset{a.s.}{\to} P_\Theta(\xi(\Theta) = k). \tag{4.4.2}$$

Note that for any given ensemble based on $\Theta_1, \Theta_2, ...\Theta_B$ there exists a set E of zero probability containing all the unions of sets for which (4.4.2) did not converge for some value l. Therefore Breiman (2001*a*) concludes that outside of E,

$$\frac{1}{B}\sum_{b=1}^{B} I\big(t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b) = k\big) \underset{a.s.}{\to} \sum_{l} P_\Theta(\xi(\Theta) = k) \cdot I(\boldsymbol{x} \in S_l) = P_\Theta(t(\boldsymbol{x}, \Theta) = k).$$

The above provides a guarantee that the ensemble approaches a limiting value of the generalisation error as more trees are added. However, Hastie *et al.* (2009) suggest that there is some confusion between this limiting value, conditional on the training data, and the complexity of the resulting model. They state that adding more trees does not cause the random forest to overfit in the sense that it is estimating $P_\Theta(t(\boldsymbol{x}, \Theta) = k)$ for each class, but rather that the limit itself specific to $t(\boldsymbol{x}, \Theta)$ can overfit the data. In fact, Segal (2004) noted that most of the benchmark datasets from the UCI repository used for performance comparisons of random forests were resistant to overfitting, even to single maximum-sized grown trees. By expanding the range of datasets to include ones on which fully grown trees *do* overfit, Segal was able to show (at least in a regression setting) that random forests can indeed overfit. A suggested remedy is to use pruned trees instead of the more commonly used fully grown trees. However, overfitting still remains a rare occurrence. This is especially the case in classification problems. Hence it is suggested that having to deal with the extra tuning parameter associated with tree pruning (the α in cost-complexity pruning), is not worth the small potential gains in performance (Hastie *et al.*, 2009).

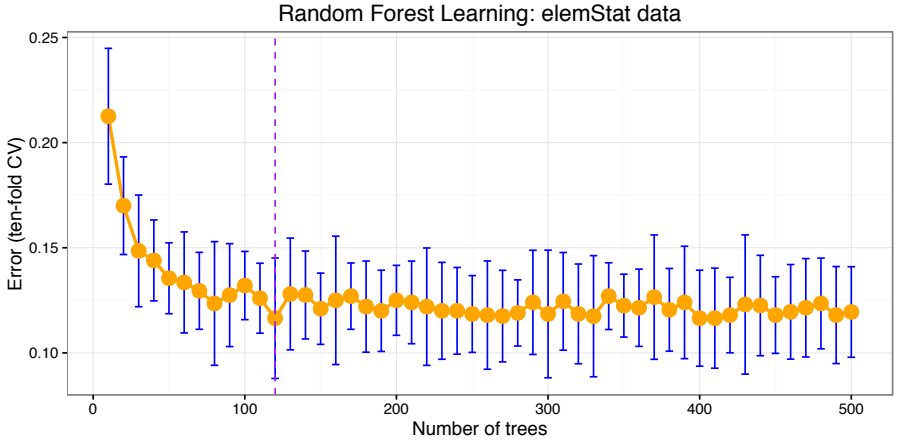

Figure 4.1: Ten-fold cross-validation errors per additional 10 trees for a random forest fit to the mixture data.

In Figure 4.1, ten-fold cross-validation is used to select the optimal ensemble size for a random forest (specifically Forest-RI discussed in the following section) fit to the elemStat data. The purple vertical line indicates the appropriate number of trees to be 120 as opposed to the maximum of 500. However, the prevailing trend seems to suggest that the performance of the random forest does not suffer much as more trees are added.

4.5 Breiman's Forest-RI

In addition to providing a general definition, Breiman (2001*a*) also proposed his own version of a random forest, *viz.* Forest-RI.[1] The Forest-RI algorithm for classification is a combination of bagging classification trees while using the random subspace method of Ho (1995), but following the node specific implementation of Amit and Geman (1997) when constructing each tree. When the tree is grown on a bootstrapped training set, only $\zeta < p$ of the input variables are selected at random as candidates for splitting at each node. The idea of this modification to bagging (essentially inducing more randomness into the procedure) is to reduce the correlation between each tree in the ensemble while having only a negligible increase in the variance of each individual tree. Therefore, Forest-RI improves on the performance of bagging by reducing $\bar{\rho}$ in (4.3.13). The Forest-RI algorithm is given in Algorithm 2 (Hastie *et al.*, 2009).

[1]RI is an abbreviation for "Random Input".

Algorithm 2 Forest-RI

1. For $b = 1$ to B:
   a) Draw a bootstrap sample Ω_b^* of size N from Ω_{tr}.
   b) Grow a *randomised tree* $t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b)$ to the boostrapped data, by recursively repeating the following steps for each terminal node of the tree, until each node contains at most N_{min} observations.
      i. Select ζ variables at random from the p input variables.
      ii. Pick the best variable on which to split, and the best corresponding split-point among the ζ chosen variables.
      iii. Split the node into two child nodes.
2. Output the ensemble of trees $\{t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b), b = 1, ..., B\}$. The random forest classification for an input $\boldsymbol{x}$ is given by the majority vote

$$\bar{t}_{FRI}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} I(t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b) = k).$$

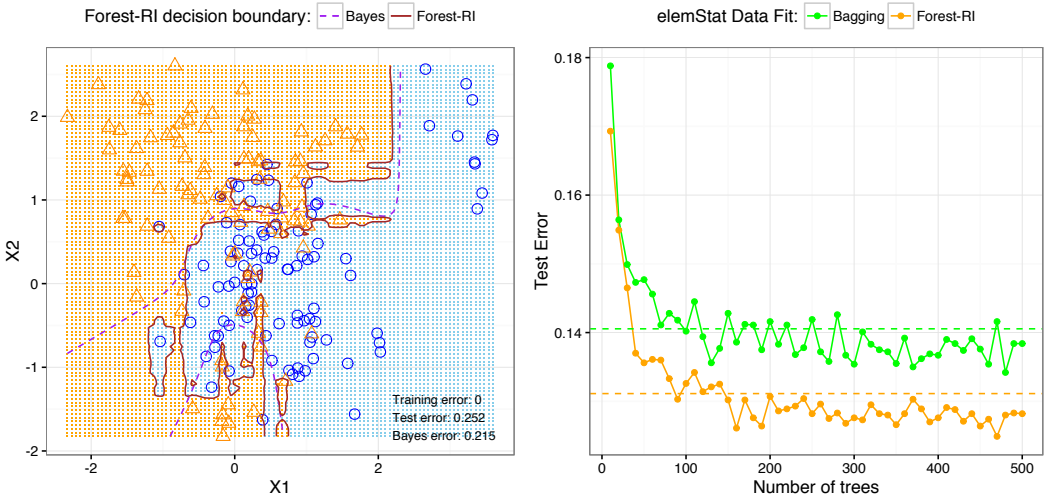

Figure 4.2: A Forest-RI fit to the mixture data: *The decision boundary is represented by the solid brown line.*

The decision boundary of a Forest-RI using 100 trees fit to the mixture data is shown in the left panel of Figure 4.2. Since the data consist of only two input variables, the subset size of randomly selected variables as candidates

for splitting at each node (the tuning parameter ζ) was set equal to one.[2] The boundary is highly non-linear due to the randomised trees used in the algorithm with the training error reduced all the way to zero. When it comes to the test error, Forest-RI has the same performance as bagging on the mixture data. However, the right panel of Figure 4.2 gives the test error curves for bagging and Forest-RI fit to the elemStat data. Presented with this richer data set, Forest-RI indeed outperforms bagging for any ensemble size selected. Also, on average over ensemble size (average errors are represented by dashed lines), Forest-RI performs the best.

4.6 More Detail Regarding Random Forests

An interesting aspect of a random forest using bootstrap sampling is that when each tree is fit, there exists a sample of observations that were not used in the construction of the tree. Specifically, if each observation has an equal probability $1/N$ of being sampled, the probability of an observation not being in the bootstrap sample is $(1-1/N)^N$. For large N, this probability is approximately equal to 36.8%. In other words, each time a tree is fit in the sequence to create the ensemble, roughly a third of the data will not be used. This collection of unused observations is referred as the *out-of-bag* (OOB) sample.

4.6.1 Out-of-Bag (OOB) Error Estimates

The proposal of using out-of-bag observations to obtain estimates of the generalisation error of a bagged classifier stemmed from work done by Tibshirani (1996) and Wolpert and Macready (1999). Given a training set Ω_{tr}, out-of-bag error estimates can be obtained by first constructing the classifiers $\{g_{\Omega_b^*}(\boldsymbol{x}), b = 1, ..., B\}$ using bootstrap resampling from the training set. Now, considering the i^{th} observation and by only using the classifiers in which this observation was in the out-of-bag set, approximately $B/3$ votes for the class to which it belongs can be collected (James *et al.*, 2013). Using the usual majority vote method gives a single out-of-bag prediction. In a similar fashion, an out-of-bag prediction can be obtained for all the observations in the training set and from these, an out-of-bag error estimate is given by the misclassification rates associated with these predictions. Concretely, the estimate is defined as

$$\bar{Err}_{OOB} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|E^{-i}|}\sum_{b\in E^{-i}} L(c_i, g_{\Omega_b^*}(\boldsymbol{x}_i)),$$

[2] If $\zeta = p$, Forest-RI is equivalent to bagging.

where E^{-i} is the set of indices corresponding to the models that did not use observation i to train and $|E^{-i}|$ is the number of indices inside this set. However, sampling with replacement also means that

$$P(observation\ i \in \Omega_b^*) = 1 - (1 - 1/N)^N \approx 0.632 \tag{4.6.1}$$

for large N. Now suppose only $B = 3$ bootstrap datasets are constructed, then (4.6.1) implies that roughly $(0.632)^3 \approx 1/4$ of the observations from the original training set will be used by all of the models for training. Therefore, either B should be chosen to be sufficiently large, meaning $|C^{-i}| > 0, \forall i = 1, ..., N$, or observations for which $|E^{-i}| = 0$ should be omitted in the computation.

A useful consequence of this estimate is that unlike the usual setting (which requires that cross-validation be performed explicitly), implicit to a random forest is a validation step in parallel with the model fit. In fact, the training phase can be conducted in such a way that the number of trees used in the random forest can be increased until some point is reached where the out-of-bag error has stabilised (Hastie *et al.*, 2009). Furthermore, Breiman (1996*b*) argued that the need for an independent test set becomes unnecessary when the out-of-bag error estimate is available. To illustrate the similarity between the two estimates, Figure 4.3 shows the out-of-bag error compared with the test error of a Forest-RI fit to the elemStat data.

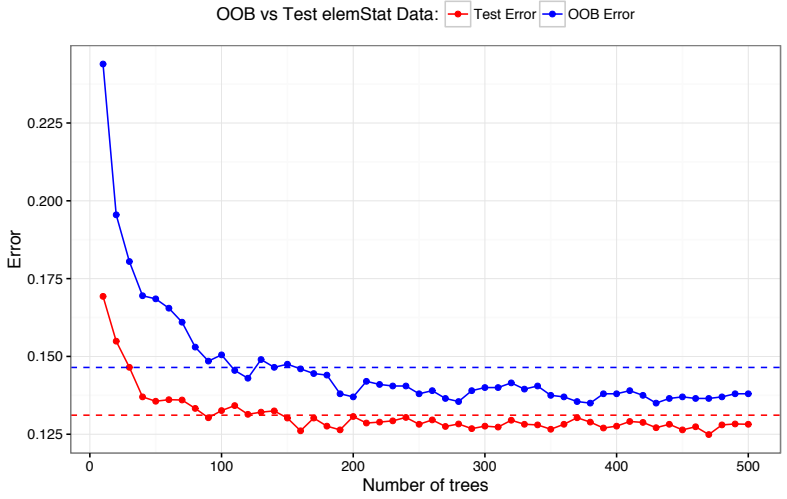

Figure 4.3: OOB error computed on the Spam training data, compared to the test error.

The two error curves in Figure 4.3 are fairly similar. Hence it seems reasonable to use the OOB error to select the optimal ensemble size. Both curves show that far fewer than 500 trees are required to obtain a stable estimate, from which point any additional trees result in a negligible reduction in misclassification error. This is an illustration of how the parameters for a random forest can be tuned using only the internal OOB estimates of the model during the fitting process. Compared to the test error, the OOB estimate is biased upwards, but in practice may still be a useful upper bound. In the next section another useful aspect of out-of-bag samples is discussed. In particular, Section 4.6.2 describes the use of out-of-bag samples in determining the relative importance of input variables in terms of their association with the response.

4.6.2 Interpretability of Random Forests

Random forests can use out-of-bag samples to measure the importance of each input variable. This is a feature of random forests that was already included in Breiman's original paper (Breiman, 2001*a*), but which has more recently also been investigated by Strobl *et al.* (2008) and Genuer *et al.* (2010). Algorithm 3 provides the necessary steps to compute variable importance (Breiman, 2001*a*).

Algorithm 3 A variable importance algorithm for Random Forests

1. For the b^{th} tree, where $b = 1, ..., B$:
   a) Drop the out-of-bag sample down the tree and store the prediction accuracy.
   b) For each input variable $X_j, j = 1, ..., p$, randomly permute the values for X_j in the out-of-bag sample and recompute and store the prediction accuracy.
2. Over all trees $\{t_{\Omega_b^*}(\boldsymbol{x}, \Theta_b), b = 1, ..., B\}$, compute the average decrease in accuracy that resulted from randomly permuting the values of each of $X_1, X_2, ..., X_p$. These quantities are then considered to measure the importance of each input variable.

The importance of a specific variable is computed as the average decrease in prediction accuracy across all trees when the values of that variable is permuted. The input variable which causes the largest decrease in accuracy after being permuted is considered to be the most important.

As a running example, in order to facilitate explanation of the interpretability of random forests, the well known *spam* data set (donated to the UCI machine learning repository by George Forman from Hewlett-Packard labs) pertaining to text classification is used. The spam data set consists of 4601 e-mails characterised by 57 predictors which are the observed percentages of the occurrence of certain words, characters and punctuation as well as specific characteristics, such as the number of capital letters used within an e-mail. The task is to use these data as input to a classifier in order to predict whether an e-mail is spam (junk) or nonspam (relevant and important to the recipient). The variable importance measures for the spam data are displayed in Figure 4.4.

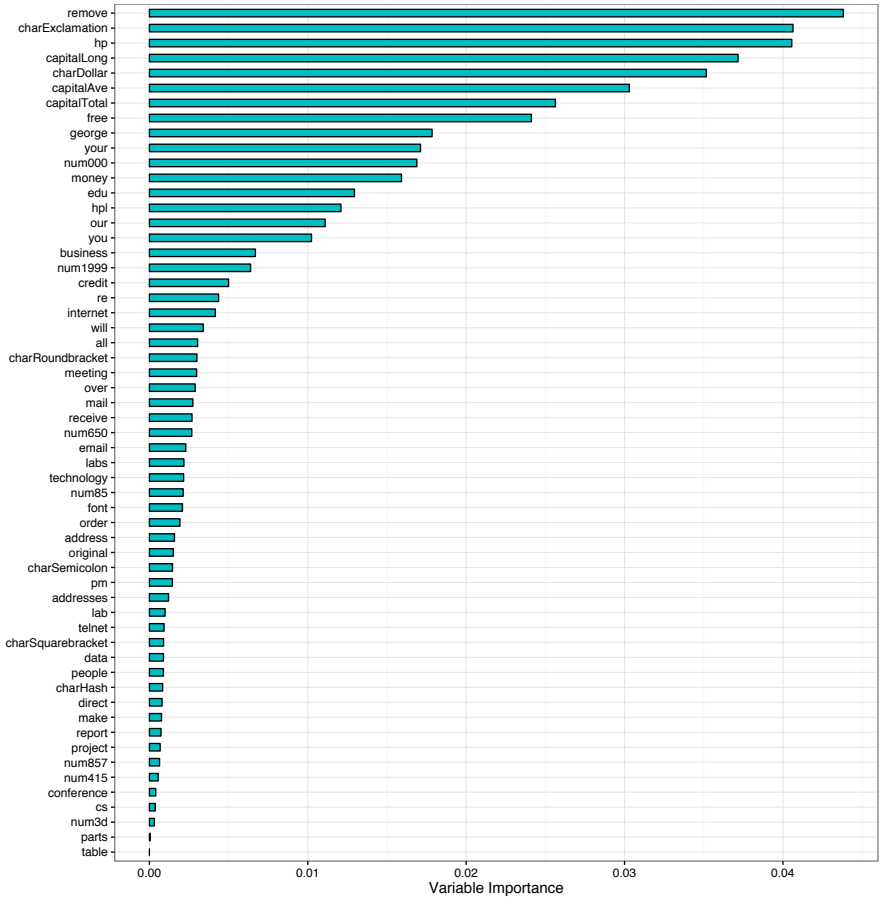

Figure 4.4: Variable importance for the spam data.

From Figure 4.4, the word *remove* is deemed most important with the percentage of exclamation marks and the occurrence of the word *hp* in close second and third place overall. Given the source of the data, it is interesting to note the importance of *hp* (short for Hewlett-Packard) and of the word *george* (the

name of the donor). For each, it is presumably the presence of the word in the e-mail that is indicative of the e-mail not being spam.

In order to further verify the above speculations, consider the matrix scatterplot in Figure 4.5, showing the top 3 (top row) and bottom 3 (bottom row) predictors with respect to their overall importance ranks as shown in Figure 4.4. The x-axis in each panel corresponds to the percentage occurrence of the particular word in an e-mail. The y-axis provides the respective posterior probabilities for each observation given $\boldsymbol{x}$.

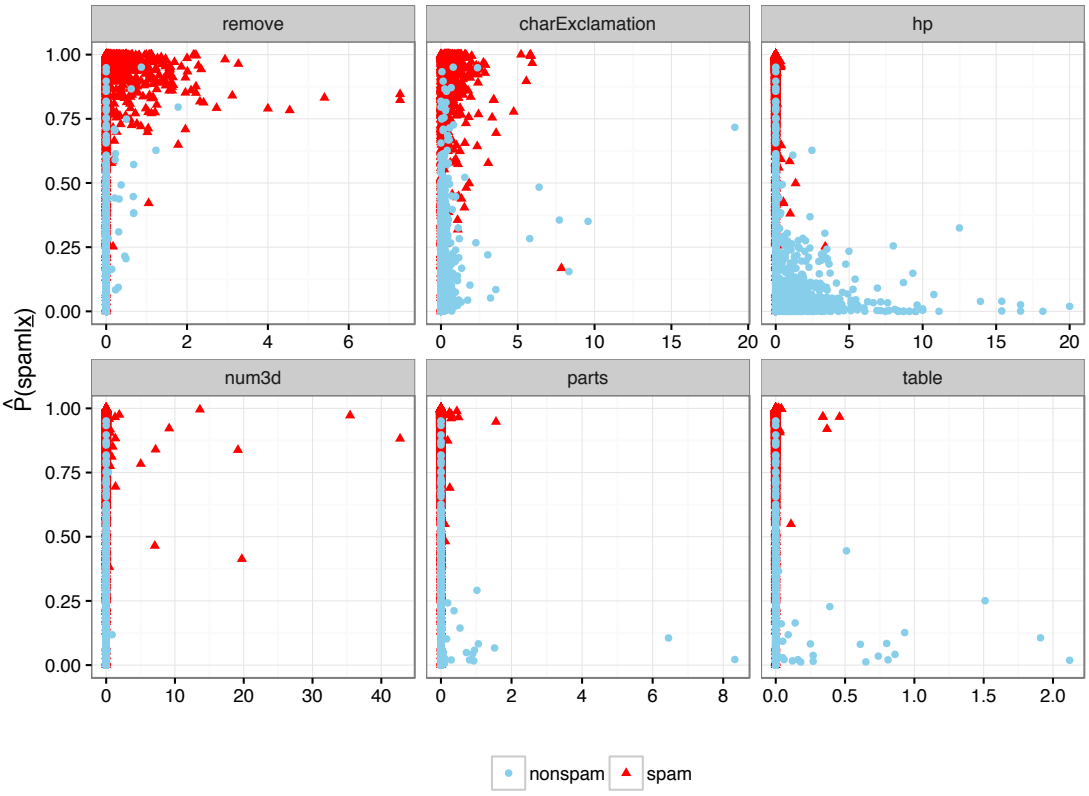

Figure 4.5: Spam data variable exploration plot: *The top row corresponds to the three most important variables and the bottom row the three least important variables.*

Considering the centre panel in the top row of Figure 4.5, it is indeed clear that the variable *hp* is useful in classifying spam since none of the spam e-mails contain this abbreviation. On the other hand, in the case of *remove* and *charExclamation*, the presence of the word in an e-mail is a very good indicator that the particular e-mail is spam. In contrast, the words *num3d*, *parts* and

table shown in the bottom row of Figure 4.5, provide very little distinction between the two classes.

In a paper addressed to the statistical community, Breiman (2001*b*) advocated what he referred to as the *algorithmic modeling culture*. Here the statistician should refrain from assuming that the data were generated from some stochastic model and rather estimate a functional relationship between the predictors and the response using an algorithm (such as a random forest). In this camp, the emphasis is placed on prediction accuracy, with the argument being that a highly accurate model should be more similar to the true unknown function that is of interest, and therefore that conclusions drawn from it should be more reliable. The main drawback is that the unknown function is replaced with a complex (but often very accurate) "black box", which is difficult to interpret. In fact, commenting on the paper, Cox and Efron expressed serious reservations regarding the abandonment of the data modeling approach - largely due to its simplicity and relative success in providing useful information. Breiman, well aware of this, recalls being told by biostatistician friends that, "Doctors can interpret logistic regression" and that faced with a choice between high accuracy and interpretability, they will opt for the latter. This is where Breiman advocates variable importance derived from random forests as having the best of both worlds, *viz.* a high level of interpretability and high prediction accuracy. But can variable importance really replace interpretation from say, a logistic regression model and when might it be more appropriate to abandon the one approach for the other?

To illuminate aspects of the two cultures, a logistic regression model was fit to the spam data. Table 4.1 shows only the significant predictors where the significance level was chosen as $\alpha = 0.05$. The list of variables in Table 4.1 seems to largely overlap with the more important variables identified in Figure 4.4. Moreover, it is interesting to note that the correlation between the rankings of the variables derived from using variable importance and from using p-values is 0.625. This suggests that performing variable selection based on cut-offs of either value could provide similar variable subsets, however this might change if variables are sequentially deleted. For more details on variable selection using random forests see Genuer *et al.* (2010).

Interpreting the logistic model, a percentage increase of 1% in the occurrence of the word *free* in an e-mail accounts for an increase in the odds of spam of $e^{1.091} = 2.977$, or roughly an increase of $\frac{e^{1.091}}{1+e^{1.091}} = 74.9\%$ in probability. This is the type of information that cannot be ascertained through variable importance measures. One could imagine many other scenarios where it may be important to have an estimate of the effect on a given outcome if a certain variable is changed.

Table 4.1: Significant predictors from the logistic regression fit to the spam data.

	Estimate	**Std. Error**	**z value**	**Pr(>|z|)**
(Intercept)	-2.079	0.227	-9.177	0
charDollar	7.658	1.167	6.563	0
remove	3.454	0.609	5.675	0
free	1.091	0.208	5.255	0
our	0.801	0.162	4.957	0
hp	-2.172	0.453	-4.794	0
edu	-1.598	0.374	-4.270	0
re	-0.883	0.216	-4.078	0
num000	2.661	0.666	3.994	0.0001
capitalTotal	0.001	0.0004	3.726	0.0002
technology	1.419	0.428	3.313	0.001
george	-5.463	1.886	-2.896	0.004
hpl	-2.351	0.843	-2.789	0.005
your	0.195	0.070	2.781	0.005
telnet	-7.851	2.854	-2.751	0.006
charExclamation	0.318	0.118	2.697	0.007
capitalLong	0.011	0.004	2.560	0.010
business	0.832	0.328	2.538	0.011
meeting	-2.970	1.292	-2.300	0.022
num85	-4.110	1.831	-2.245	0.025
internet	0.465	0.208	2.242	0.025
conference	-3.838	1.714	-2.240	0.025
over	0.582	0.261	2.226	0.026
charSemicolon	-1.063	0.482	-2.208	0.027
cs	-63.139	28.695	-2.200	0.028
capitalAve	0.159	0.072	2.196	0.028
pm	-1.189	0.551	-2.157	0.031
receive	-0.981	0.458	-2.145	0.032
you	0.101	0.047	2.143	0.032

A "black box" alternative is the *partial dependence plot* (Hastie *et al.*, 2009). Consider $\boldsymbol{X}_S$ a subvector consisting of $r < p$ input variables from $\boldsymbol{X}^T = (X_1, ..., X_p)$ where $S \subset \{1, 2, ..., p\}$. Let $C = S^c$, which implies that $S \cup C = \{1, ..., p\}$, and $P(k|\boldsymbol{x}) = P(k|\boldsymbol{x}_S, \boldsymbol{x}_C)$. Then the partial dependence of $P(k|\boldsymbol{x})$ on $\boldsymbol{X}_S$ is

$$P_S(k|\boldsymbol{x}_S) = E_{\boldsymbol{x}_C}\big[P(k|\boldsymbol{x}_S, \boldsymbol{x}_C)\big]. \tag{4.6.2}$$

The expectation in (4.6.2) can be estimated by

$$\hat{P}_S(k|\boldsymbol{x}_S) = \frac{1}{N}\sum_{i=1}^{N} \hat{P}(k|\boldsymbol{x}_S, x_{Ci}), \tag{4.6.3}$$

where $x_{Ci}, i = 1, ..., N$, are the values of $\boldsymbol{x}_C$ in the training data. In practical

terms, the partial dependence is computed by estimating the posterior probability for each fixed value of $\boldsymbol{X}_S$, using the training set. This quantifies the change in probability as a function of change in $\boldsymbol{X}_S$.

The partial dependence of the posterior probability estimate for a spam e-mail on the variables *free* and *george* is given in Figure 4.6. In terms of the direction of the effect of each variable, these plots are in agreement with the logistic regression output in Table 4.1. An increase in the occurrence of the word *free* increases the probability of spam, and vice versa for the word *george*. In addition, both approaches indicate the change in class probability to taper off. This can be seen in Figure 4.6 and in the case of logistic regression stems from the logistic sigmoid function. However, two different pictures are being sketched.

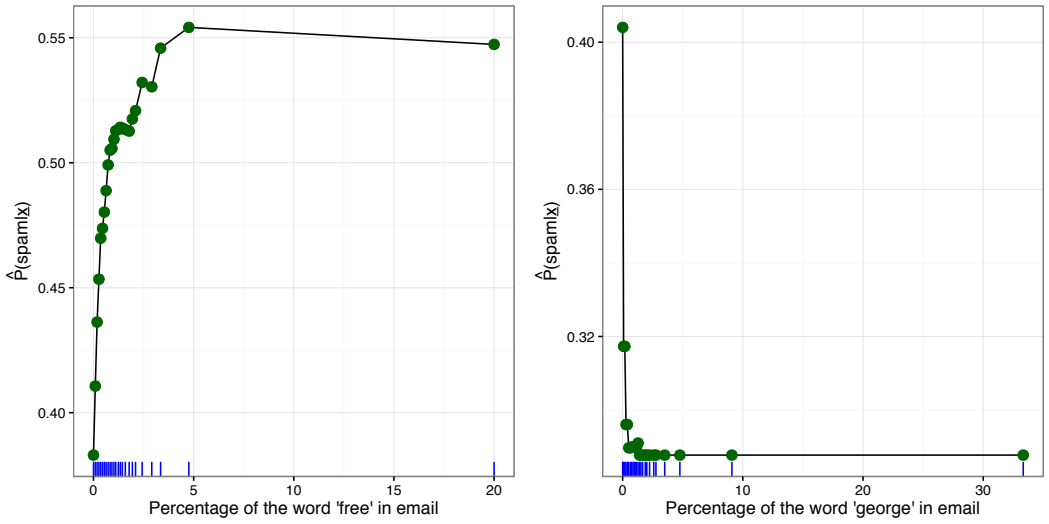

Figure 4.6: Random forest partial dependence plot: *Left: Partial dependence for the word "free". Right: Partial dependence for the word "george".*

For example, on the one hand, logistic regression is saying that if enough (roughly an increase of 5% in this case) occurrences of the word *free* is observed in an e-mail, the probability of spam will be very close to 1. On the other hand, the partial dependence hints at a limiting effect situated around only 55% for spam. So at this point, which approach better reflects the truth is left to the realm of the subjective. Even so, the algorithmic argument still begs the question of whether there is a price to pay for interpreting coefficients obtained from a less accurate model. Table 4.2 provides the confusion matrices for both the Forest-RI and logistic regression fit to the spam data.

Table 4.2: Model confusion matrices (logistic regression abbreviated as *LR*).

Forest-RI: True Class

		Nonspam	Spam
Predicted Class	Nonspam	58.4%	3.6%
	Spam	2.2%	35.8%

LR: True Class

		Nonspam	Spam
Predicted Class	Nonspam	57.3%	4.7%
	Spam	3.3%	34.7%

Forest-RI outperforms the logistic classifier in terms of overall prediction accuracy. However, in the case of spam detection there exists asymmetric costs in misclassification. In particular, a higher price is paid for *false positives*. Classifying an e-mail as spam when in fact it is not, is far more damaging than letting a spam e-mail slip through detection. In Table 4.2 it is seen that the logistic regression classifier has a false positive rate which is 1.1% worse than that of Forest-RI. This does not sound like much, but consider the following: according to technology market research, in 2015 approximately 205 billion e-mails were sent every day (The Radicati Group, 2015). This means that if conclusions are drawn from the logistic model, they are based on a classifier that would potentially mislabel legitimate e-mail as spam roughly 2.3 billion times more than the Forest-RI model, *every day*. An objection might be that the classification threshold can simply be adjusted to account for a poor false positive rate, therefore the ROC curves for both models are depicted in Figure 4.7. The ROC curves show that for low values of the false positive rate, there is no threshold for which the logistic regression model will outperform the random forest. Although this is only one example, the accuracy-interpretability trade-off tends to hold in general (James *et al.*, 2013).

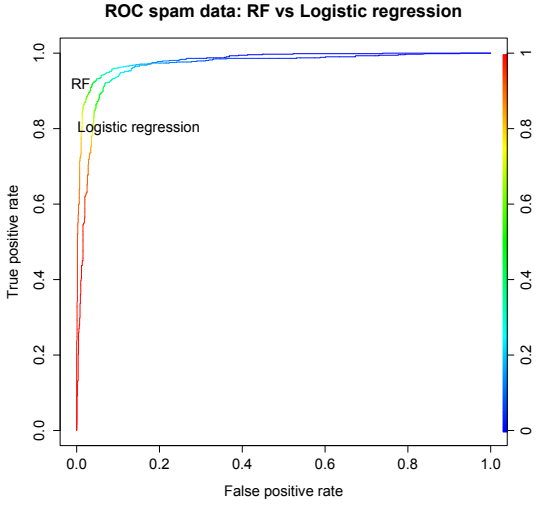

Figure 4.7: ROC curves for a random forest and logistic regression fit to the spam data.

In the end, Breiman (2001*b*) seems to be saying that if less accuracy translates to unreliable conclusions, shouldn't accuracy (which algorithmic models excel at) always be the top priority, even when interpretation is the main goal?

Another by-product of a random forest classifier is a *proximity plot* (Breiman, 2001*a*). These plots are constructed by first forming an $N \times N$ symmetric proximity matrix. This is obtained by recording, for each tree in the sequence, the pairwise number of out-of-bag observations falling into the same terminal node for each tree in the sequence. The next step is to use multidimensional scaling (MDS) to obtain the best two-dimensional approximation of the full N-dimensional proximity space.[3] The main idea of the proximity plot is to give a visual perspective on "distances" between observations based on the number of times that observations share the same terminal node. In order to gain a clearer understanding of the inner workings of these plots, the left panel of Figure 4.8 compares the positions of points on a proximity plot to the corresponding positions in the input space in the right panel for a random forest fit on the mixture data.

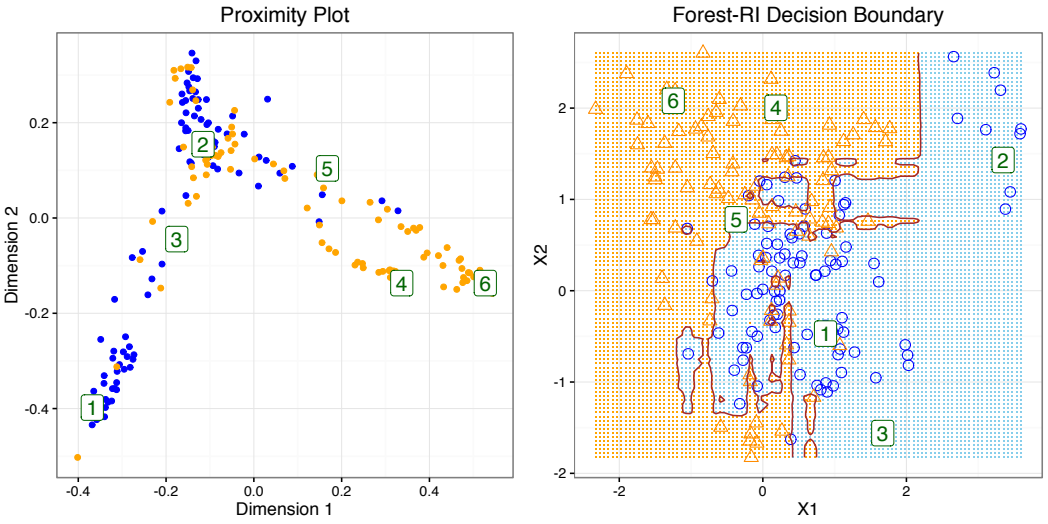

Figure 4.8: Random forest proximity plots: a comparison of a proximity plot with RF decision boundary.

In Figure 4.8, the numbered labels indicate the locations of specific points. The proximity plot has an upside down "V" shape, with each side roughly corresponding to one of the classes. According to Hastie *et al.* (2009), the main takeaway message from these plots are that points that lie at the tips of the

[3]MDS is described in more detail in Section 6.5.

"V" are often found to lie safely inside their respective class neighbourhoods, whereas points closer to the intersection tend to lie near the decision boundary. However, this is clearly not always the case, as can be observed with the point labelled 2. Hastie *et al.* (2009) also comment on the usefulness of proximity plots, stating that their shapes are highly invariable across different data sets, which reduces their explanatory power. Furthermore, proximity plots seem to be regularly omitted from discussions on random forests and aspects pertaining to the algorithm class (Siroky *et al.*, 2009; Boulesteix *et al.*, 2012; Ziegler and König, 2014).

In contrast, Xu *et al.* (2012) describe a method for performing missing value imputation using random forest proximity weighted averages or using a weighted majority vote if the missing value is categorical. In addition, they compare traditional Euclidean MDS to proximity plots on prostate cancer micro-array data and conclude that proximity plots provide greater visual structure. No formal analysis is given, but the authors seem convinced that proximity plots are of real use and form an integral part of the random forest toolkit.

4.7 Concluding Remarks

Random forests are memoryless non-deterministic ensemble classifiers that exclusively use trees as base learners. Their generalisation error is bounded by the strength of each tree in the ensemble, as well as by the correlation between trees, where by increasing the former, and reducing the latter, an improved classifier can be obtained. Furthermore, it has been argued that random forests are resistant to overfitting. A popular random forest is Breiman's Forest-RI, which modifies bagging by using trees as base learners where at each node split, only a subset of the input variables are selected as candidates for splitting.

By sampling with replacement, random forests can have at each step of the ensemble creation an out-of-bag sample that is not used to construct the current tree. These samples can be used to obtain estimates of the test error, as well as for selecting the optimal ensemble size, thereby obviating the need for cross-validation. Furthermore, implicit to random forests is a toolkit which facilitates model interpretation, including variable importance, partial dependence and proximity plots.

Towards a deeper understanding of random forests in terms of bias and variance, the next chapter investigates the bias-variance trade-off in the context of random forests using a particular bias-variance decomposition for classification. The road map forward is presented in Figure 4.9 below.

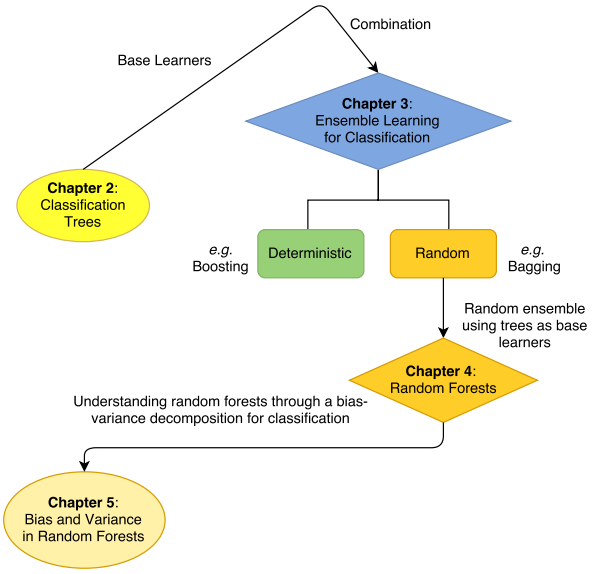

Figure 4.9: Road map to Chapter 5: An investigation of bias and variance in random forests.

Chapter 5

Bias and Variance in Random Forests

It has already been argued that in random forests, classification performance is improved by a further reduction in variance compared to that achieved by aggregation. The extent to which random forests affect bias is less clear, therefore a bias-variance analysis of random forests is of interest. In Section 5.1 a brief overview is provided of key bias and variance concepts, as well as of the bias-variance decomposition in regression. A related discussion on probability estimation (essentially a regression task) is given in Section 5.2. In Section 5.3 the difficulty of defining bias and variance for classification is discussed. This is followed in Section 5.4 by a review of several proposed bias-variance definitions in the case of 0-1 *loss, and a generalisation of bias and variance in the case of symmetric loss. Section 5.5 is concerned with the effect of randomisation and aggregation on the bias and variance of an ensemble. Finally in this chapter, Section 5.6 presents an empirical investigation of bias and variance in random forests, with concluding remarks in Section 5.7.*

5.1 Introduction

To refresh and expand on some of the key concepts of bias and variance, a temporary switch is made from classification to the regression setting. Suppose the true distribution of a quantitative response Y given a point $\boldsymbol{X} = \boldsymbol{x}$ is given by $P(Y|\boldsymbol{x})$. Using the training data Ω_{tr}, this distribution can be estimated. Let $P_{\Omega_{tr}}(Y|\boldsymbol{x})$ denote this estimate. Figure 5.1 provides an illustration of a possible relationship between $P(Y|\boldsymbol{x})$ and $P_{\Omega_{tr}}(Y|\boldsymbol{x})$ (Geurts, 2002).

In more detail, using the additive error model $Y = f_B(\boldsymbol{x}) + \epsilon$, where $E(\epsilon) = 0$,

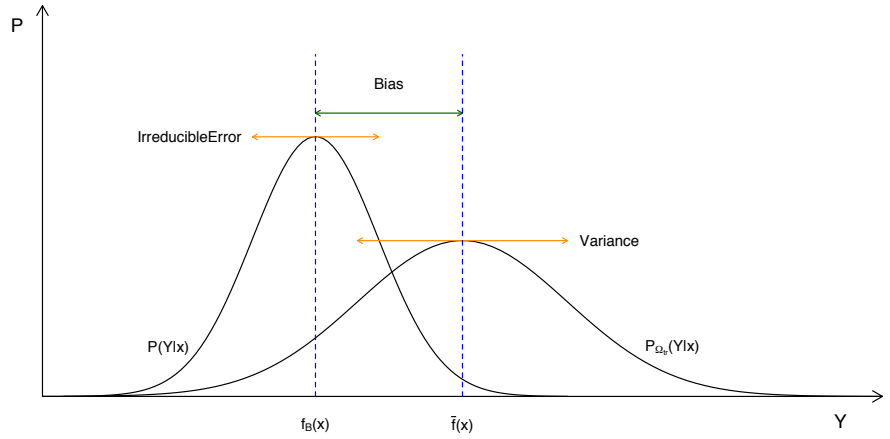

Figure 5.1: Bias and variance in regression.

$Var(\epsilon) = \sigma_\epsilon^2$, and squared-error loss $L_{SE}(Y, f(\boldsymbol{x})) = (Y - f(\boldsymbol{x}))^2$, the following decomposition of the expected prediction error of an estimated function f at a point $\boldsymbol{X} = \boldsymbol{x}$ can be derived:[1]

$$\begin{aligned} Err_{SE}(\boldsymbol{x}) &= E[(Y - f(\boldsymbol{x}))^2] \\ &= E[(f_B(\boldsymbol{x}) + \epsilon - f(\boldsymbol{x}))^2] \\ &= E[(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))^2 + 2\epsilon(f_B(\boldsymbol{x}) - f(\boldsymbol{x})) + \epsilon^2] \\ &= E[(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))^2] + 2E[\epsilon(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))] + E(\epsilon^2). \end{aligned} \tag{5.1.1}$$

The function $f_B(\boldsymbol{x})$ represents the model that obtains the Bayes error rate, *i.e.* $f_B(\boldsymbol{x}) = \text{argmin}_a E[(Y - a)^2|\boldsymbol{x}] = E(Y|\boldsymbol{x})$. Since the model error ϵ is assumed to be independent from the data generating process and $E(\epsilon) = 0$, the second term in equation (5.1.1) becomes

$$2E[\epsilon(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))] = 2E(\epsilon)E[(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))] = 0.$$

Furthermore,

$$\sigma_\epsilon^2 = E(\epsilon^2) - E(\epsilon)^2 = E(\epsilon^2)$$

and therefore (5.1.1) simplifies to

$$Err_{SE}(\boldsymbol{x}) = E[(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))^2] + \sigma_\epsilon^2. \tag{5.1.2}$$

[1]The original decomposition is attributed to Geman *et al.* (1992), but can be found in various texts such as Hastie *et al.* (2009) and James *et al.* (2013).

The first term in (5.1.2) represents the *reducible* part of the generalisation error, while σ_ϵ^2 is the so-called *irreducible error*. Adding and subtracting the expectation of the estimated regression function at $\boldsymbol{X} = \boldsymbol{x}$, the reducible error may be further decomposed:

$$\begin{aligned} E[(f_B(\boldsymbol{x}) - f(\boldsymbol{x}))^2] &= E[(f_B(\boldsymbol{x}) - E(f(\boldsymbol{x})) + E(f(\boldsymbol{x})) - f(\boldsymbol{x}))^2] \\ &= E[(f_B(\boldsymbol{x}) - E(f(\boldsymbol{x})))^2 \\ &\quad + 2(f_B(\boldsymbol{x}) - E(f(\boldsymbol{x})))(E(f(\boldsymbol{x})) - f(\boldsymbol{x})) \\ &\quad + (E(f(\boldsymbol{x})) - f(\boldsymbol{x}))^2], \end{aligned}$$

where $E[(E(f(\boldsymbol{x})) - f(\boldsymbol{x})] = E(f(\boldsymbol{x})) - E(f(\boldsymbol{x})) = 0$. Finally, letting $\bar{f}(\boldsymbol{x}) = E(f(\boldsymbol{x}))$, the expected prediction error of f can be decomposed into three parts, *viz.*

$$\begin{aligned} Err_{SE}(\boldsymbol{x}) &= \sigma_\epsilon^2 + (f_B(\boldsymbol{x}) - \bar{f}(\boldsymbol{x}))^2 + E[(\bar{f}(\boldsymbol{x}) - f(\boldsymbol{x}))^2] \\ &= Irreducible\ Error + Bias^2 + Variance. \end{aligned} \tag{5.1.3}$$

In general, as the complexity of f increases, the squared bias decreases, and vice versa for the variance. As soon as the increase in variance from a more complex model starts to dominate the decrease in bias, the expected prediction error of the model will increase. On the other hand, if the bias is large and an increase in variance is associated with a larger decrease in bias, the use of a more complicated model is justified. Figure 5.2 shows the bias-variance trade-off for two opposite scenarios, where either bias or variance is the dominating factor affecting prediction performance (Geurts, 2002).

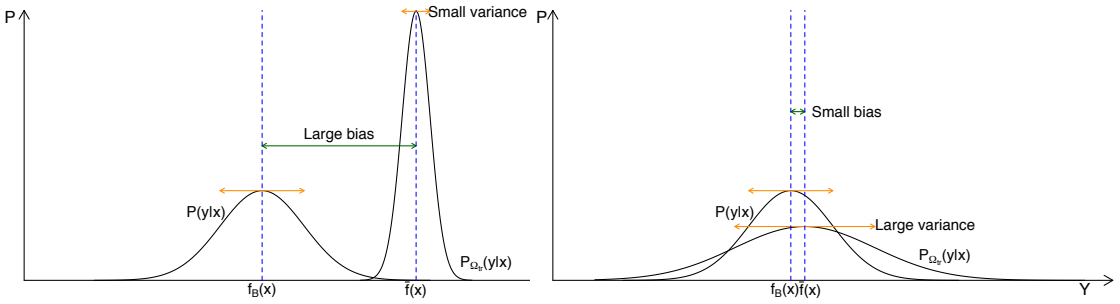

Figure 5.2: Bias and variance of an estimated distribution: Left: *Large bias and small variance.* Right: *Small bias and large variance.*

In the left panel of Figure 5.2 the estimates from the fitted model do not vary

much around the average $\bar{f}$, which means that the variance in (5.1.3) is small. However, $\bar{f}$ is far from the expected value of the true distribution of f_B, which results in the (squared) bias in (5.1.3) to be large and to increase the expected prediction error. The opposite scenario is shown in the right panel of Figure 5.2, where the average of the estimated model is very close to the expected value of the true distribution. But unfortunately, the model estimates vary a lot around their mean, causing the expected prediction error to increase. It is clear that both low bias and low variance are required for optimal prediction performance. However note that even in the case of $\bar{f} = f_B$ and a zero variance for f, the expected prediction error would still not be equal to zero. This is due to the irreducible variance of the true distribution.

5.2 A Probability Estimate Perspective

Since estimating probabilities can be seen as a regression task, the remarks made in the previous section regarding bias and variance also hold in this context. However, when the probability estimates are ultimately used to perform classification (for example by using a decision threshold), the situation becomes more complicated. Refer in this regard to Friedman (1997) and the fact that the mean and the variance of an estimated model affects classification error differently than in the case of squared prediction error in regression.

In more detail, let the loss associated with a binary classification problem be symmetric. For example, if $\ell_0 = \ell_1 (= 1 \text{ say})$, then the appropriate decision threshold is equal to $1/2$ (as was shown in Section 1.3). Therefore, as is commonly done, given a function $\hat{p}(\boldsymbol{x}) = \hat{P}(C = 1|\boldsymbol{x})$ estimated from the training data Ω_{tr}, the associated classifier can be constructed as

$$g_{\Omega_{tr}}(\boldsymbol{x}) = I\left(\hat{p}(\boldsymbol{x}) \geq \frac{1}{2}\right). \tag{5.2.1}$$

The expected loss is $E[L(C, g_{\Omega_{tr}}(\boldsymbol{x}))] = P(g_{\Omega_{tr}}(\boldsymbol{x}) \neq C)$, where $C \in \{0, 1\}$. Since the optimal classifier is the Bayes classifier $g_B(\boldsymbol{x})$, the performance of $g(\boldsymbol{x})$ depends on how often it agrees with $g_B(\boldsymbol{x})$, or equivalently $P(\bar{g}_{\Omega_{TR}}(\boldsymbol{x}) = g_B(\boldsymbol{x})|\boldsymbol{x})$. Here Ω_{TR} does not symbolise a specific training set, but denotes the estimated distribution over repeated sampling from the data generating process. Therefore, $\bar{g}_{\Omega_{TR}}(\boldsymbol{x}) = \arg\max_k E_{\Omega_{TR}}\{I(g(\boldsymbol{x}) = k)\}$ is the majority vote classifier taken over multiple training data sets. However, for the time being when referring to $P(\bar{g}_{\Omega_{TR}}(\boldsymbol{x})|\boldsymbol{x})$ and similar expressions, the subscript Ω_{TR} and the dependence on $\boldsymbol{x}$ will be dropped for convenience. Let $P_{\Omega_{TR}}$ denote the distribution of probabilities produced by $\hat{p}(\boldsymbol{x})$ over repeated sampling, then

$$P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x})) = I\left(p < \frac{1}{2}\right) \int_{-\infty}^{1/2} P_{\Omega_{TR}} d\hat{p} + I\left(p \geq \frac{1}{2}\right) \int_{1/2}^{\infty} P_{\Omega_{TR}} d\hat{p}, \tag{5.2.2}$$

where $p = P(C = 1|\boldsymbol{x})$ represents the true probability. The expression given in (5.2.2) is the proportion of the distribution $P_{\Omega_{TR}}$ that lies on the correct side of the decision threshold. What is of interest is how the mean and variance of $P_{\Omega_{TR}}$ affect the prediction performance through (5.2.2). The exact form of $P_{\Omega_{TR}}$ is unknown. Friedman (1997) proceeds by approximating $P_{\Omega_{TR}}$ by a normal distribution.[2] That is:

$$P_{\Omega_{TR}} \approx \frac{1}{\sqrt{2\pi Var(\hat{p})}} e^{-\frac{(\hat{p}-E(\hat{p}))^2}{2Var(\hat{p})}}. \tag{5.2.3}$$

Plugging (5.2.3) into (5.2.2), $P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x}))$ becomes

$$I\left(p < \frac{1}{2}\right) \int_{-\infty}^{1/2} \frac{1}{\sqrt{2\pi Var(\hat{p})}} e^{-\frac{(\hat{p}-E(\hat{p}))^2}{2Var(\hat{p})}} d\hat{p} + I\left(p \geq \frac{1}{2}\right) \int_{1/2}^{\infty} \frac{1}{\sqrt{2\pi Var(\hat{p})}} e^{-\frac{(\hat{p}-E(\hat{p}))^2}{2Var(\hat{p})}} d\hat{p}. \tag{5.2.4}$$

Using the substitution rule with $u = \frac{\hat{p}-E(\hat{p})}{\sqrt{Var(\hat{p})}}$ gives $du = 1/\sqrt{Var(\hat{p})} d\hat{p}$. The integral boundaries change to $u = \frac{1/2-E(\hat{p})}{\sqrt{Var(\hat{p})}}$ when $\hat{p} = 1/2$, to $u = \infty$ when $\hat{p} = \infty$, and to $u = -\infty$ when $\hat{p} = -\infty$. Resubstituting the above into (5.2.4), (5.2.2) changes to

$$I\left(p < \frac{1}{2}\right) \int_{-\infty}^{\frac{1/2-E(\hat{p})}{\sqrt{Var(\hat{p})}}} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du + I\left(p \geq \frac{1}{2}\right) \int_{\frac{1/2-E(\hat{p})}{\sqrt{Var(\hat{p})}}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du. \tag{5.2.5}$$

Furthermore, due to the symmetry of the normal distribution centred at zero, an integral over the range $(-\infty, -a]$ is equal to the integral over $[a, \infty)$ for any

[2] Friedman (1997) justifies the approximation by pointing out that many classification algorithms tend to compute an average over outcomes which should in theory be more normally distributed. However, Geurts (2002) notes that the normality assumption will not always be satisfied. For example, a classification tree where all terminal nodes are pure nodes will have a probability distribution that is condensed at either zero or one. Even so, both authors are still of the opinion that the qualitative insights gained from the derivation hold in general.

$a \in \mathbb{R}$. More specifically, with $a = \frac{E(\hat{p})-1/2}{\sqrt{Var(\hat{p})}}$,

$$\int_{-\infty}^{\frac{1/2-E(\hat{p})}{\sqrt{Var(\hat{p})}}} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du = \int_{\frac{E(\hat{p})-1/2}{\sqrt{Var(\hat{p})}}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du. \tag{5.2.6}$$

Finally, combining (5.2.5) and (5.2.6), (5.2.2) can be written as

$$P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x})) = \bar{\Phi}\left[sign(1/2-p) \cdot \frac{E(\hat{p}) - 1/2}{\sqrt{Var(\hat{p})}}\right], \tag{5.2.7}$$

where $\bar{\Phi}(z) = \int_z^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz$ and where if $z \geq 0$, $sign(z) = 1$, otherwise $sign(z) = -1$.

The conclusions drawn from (5.2.7) are as follows:

- if $sign(1/2-p) = -1$ and $E(\hat{p}) > 1/2$, which implies f to on average lie on the correct side of the decision threshold, $P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x}))$ increases as $Var(\hat{p})$ decreases;
- if $sign(1/2-p) = -1$ and $E(\hat{p}) < 1/2$, which implies f to on average lie on the wrong side of the decision threshold, interestingly by *increasing* $Var(\hat{p})$, $P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x}))$ can also be increased.

To aid in better understanding the implications of (5.2.7), the left panel of Figure 5.3 illustrates a scenario where the expectation of $P_{\Omega_{TR}}$ lies on the correct side of the decision threshold (Geurts, 2002). The probability of $\bar{g}(\boldsymbol{x})$ agreeing with the Bayes classifier at a point $\boldsymbol{x}$ is shown as the area under the distribution coloured in red. By decreasing the variance of $P_{\Omega_{TR}}$ as shown in the right panel of Figure 5.3, the area associated with $P(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x}))$ increases in size. Note that this means that as long as the expectation is on the correct side, perfect classification can be achieved simply by sufficiently reducing the variance of $\hat{p}$. Theoretically, this is possible irrespective of the bias of the probability estimates.

The more interesting scenario, where the expectation of $P_{\Omega_{TR}}$ lies on the wrong side of the decision threshold, is shown in Figure 5.4. In this case, by increasing the variance of probability estimates, a greater proportion of the distribution $P_{\Omega_{TR}}$ will fall on the correct side of the decision threshold. Therefore, in such a setting the focus must be on reducing the bias, whilst however also trying to increase the variance as much as possible.

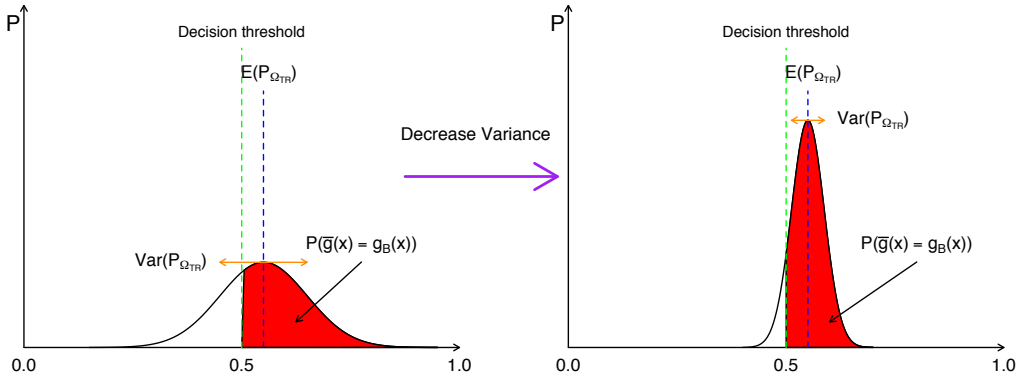

Figure 5.3: The effect of decreasing the variance of probability estimates on classification when $p > 0.5$ and $E(P_{\Omega_{TR}}) > 0.5$.

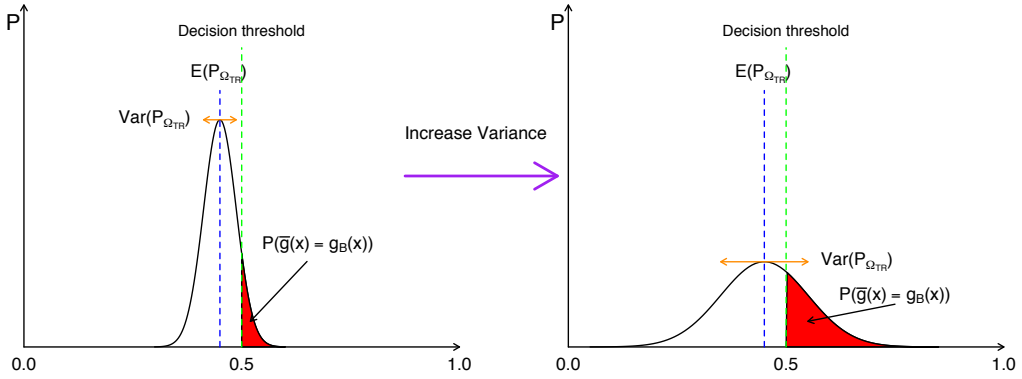

Figure 5.4: The effect of increasing the variance of probability estimates on classification when $p > 0.5$ and $E(P_{\Omega_{TR}}) < 0.5$.

The phenomenon outlined above has become widely discussed in the literature on bias and variance in classification and is now known as the *Friedman effect*. However not everyone accepts its validity, with probably the most substantial critique of the Friedman effect provided by Wolpert (1997). Nevertheless, many authors (including Wolpert) regard it as a significant contribution towards better understanding bias and variance in classification.

5.3 Bias and Variance of a Classifier

The focus in the previous section was on how bias and variance associated with probability estimates affect classification performance. But what about the notion of bias and variance directly applied to the task of assigning an observation to a specific class, or in other words applied to a classifier $g(\boldsymbol{x})$? In analogue with the regression case, the following could serve as possible def-

initions for the irreducible error, bias and variance of $g(\boldsymbol{x})$ at a point $\boldsymbol{x}$ for 0-1 loss (Geurts, 2002):

- The irreducible error is defined in terms of misclassification error rate of the Bayes classifier:

$$\sigma_{0-1}(\boldsymbol{x}) = 1 - P(g_B(\boldsymbol{x})). \tag{5.3.1}$$

- Bias is defined in terms of disagreement with the Bayes classifier:

$$Bias_{0-1}(\boldsymbol{x}) = I(\bar{g}(\boldsymbol{x}) \neq g_B(\boldsymbol{x})). \tag{5.3.2}$$

- Variance is defined in terms of the probability of $\boldsymbol{x}$ being assigned to the class predicted by $\bar{g}(\boldsymbol{x})$ taken over multiple training sets sampled from Ω_{TR}:

$$Var_{0-1}(\boldsymbol{x}) = 1 - P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x})). \tag{5.3.3}$$

In (5.3.2) the bias is zero if $\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x})$, and equal to one otherwise. Furthermore, in (5.3.3), $Var_{0-1}(\boldsymbol{x}) = 0$ if over all of the sampled training sets, $g(\boldsymbol{x})$ assigns $\boldsymbol{x}$ to the same class. At the other end, the variance reaches a maximum when $P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x})) = \frac{1}{K}$ ($= 0.5$ in the binary case). This corresponds to the highest degree of uncertainty among the classifiers over the sampled training sets. Therefore, the above definitions seem reasonable, but unfortunately they do not yield an additive decomposition as in the regression case. That is,

$$Err_{0-1}(\boldsymbol{x}) \neq \sigma_{0-1}(\boldsymbol{x}) + Bias_{0-1}(\boldsymbol{x}) + Var_{0-1}(\boldsymbol{x}). \tag{5.3.4}$$

As an example, Figure 5.5 shows the true distribution for an observation $\boldsymbol{x}$ over three possible classes, accompanied by estimated class distributions obtained from two different classifiers g_1 and g_2, fit to training sets repeatedly sampled from Ω_{TR} (James, 2003). For this scenario $\sigma_{0-1}(\boldsymbol{x}) = 0.4$.

Both $\bar{g}_1(\boldsymbol{x})$ and $\bar{g}_2(\boldsymbol{x})$ have a bias equal to one at $\boldsymbol{x}$, by classifying to the second class when the Bayes classifier in fact assigns $\boldsymbol{x}$ to the first. However, the variance of $\bar{g}_1(\boldsymbol{x})$, *viz.* $Var^1_{0-1}(\boldsymbol{x}) = 1 - 0.7 = 0.3$, is less than the variance of $\bar{g}_2(\boldsymbol{x})$, *viz.* $Var^2_{0-1}(\boldsymbol{x}) = 1 - 0.5 = 0.5$. To illustrate (5.3.4), note that the expected classification error for the two approaches is $Err^l_{0-1}(\boldsymbol{x}) =$

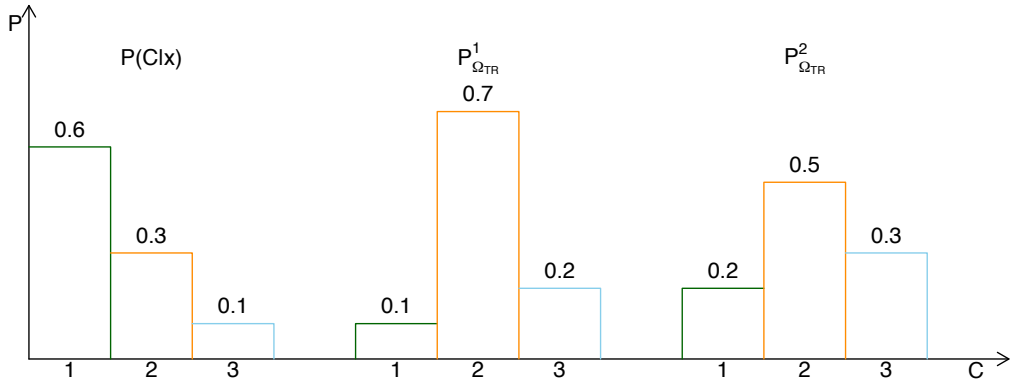

Figure 5.5: Class distributions for a three class classification task: Left: *The true distribution.* Middle: *Class distribution over training set samples for the first classifier.* Right: *Class distribution over training set samples for the second classifier.*

$1 - \sum_{k=1}^{3} P(k|\boldsymbol{x}) \cdot P_{\Omega_{TR}}(\bar{g}_l(\boldsymbol{x}) = k|\boldsymbol{x})$, where $l = 1, 2$. Hence,

$$Err^1_{0-1}(\boldsymbol{x}) = 1 - 0.6(0.1) - 0.3(0.7) - 0.1(0.2) = 0.71 \text{ and}$$
$$Err^2_{0-1}(\boldsymbol{x}) = 1 - 0.6(0.2) - 0.3(0.5) - 0.1(0.3) = 0.7.$$

However, for classifier g_1, $\sigma^1_{0-1}(\boldsymbol{x}) + Bias^1_{0-1}(\boldsymbol{x}) + Var^1_{0-1}(\boldsymbol{x}) = 0.4 + 1 + 0.3 = 1.7 \neq 0.71 = Err^1_{0-1}(\boldsymbol{x})$. Similarly, $\sigma^2_{0-1}(\boldsymbol{x}) + Bias^2_{0-1}(\boldsymbol{x}) + Var^2_{0-1}(\boldsymbol{x}) = 0.4 + 1 + 0.5 = 1.9 \neq 0.7 = Err^2_{0-1}(\boldsymbol{x})$. Furthermore, using the above definitions for bias and variance, the Friedman effect is again observed. Given that both algorithms disagree with the Bayes classifier, $\bar{g}_2$ achieves a lower expected error by having a variance *greater* than that of $\bar{g}_1$. However, some authors believe that the counter-intuitive nature of the effect is due to an inappropriate set of definitions for bias and variance. In fact, a general disagreement among authors regarding bias and variance in classification has led to many different proposed definitions. These include Dietterich and Kong (1995), Breiman (1996*a*), Kohavi and Wolpert (1996), Tibshirani (1996), James and Hastie (1997), Heskes (1998), Breiman (2000) and Domingos (2000). Each new pair of definitions is based on certain requirements seen as fitting to each concept (such as that the variance should always be positive), with each author favouring a different set of desired properties for bias and variance. Furthermore, among most of the proposals is an interest in finding an additive decomposition specifically for the expected 0-1 loss analogous to that found for squared error loss in regression. A brief exposition of the proposed definitions for bias and variance that may be found in the literature, is given below.[3]

[3]For clarity regarding the summary, suppose $C \in \{1, 2, ..., K\}$ then $P(2)$ is the true probability of the second class. The quantity $P_{\Omega_{TR}}(2)$ is the probability of the second class

- Kong and Dietterich (1995):

$$Bias_{KD}(\boldsymbol{x}) = I(\bar{g}(\boldsymbol{x}) \neq g_B(\boldsymbol{x}))$$
$$Var_{KD}(\boldsymbol{x}) = 1 - P_{\Omega_{TR}}(g_B(\boldsymbol{x})) - I(\bar{g}(\boldsymbol{x}) \neq g_B(\boldsymbol{x})).$$

  The bias is identical to (5.3.2), which is equal to one when g disagrees with the Bayes classifier, and zero otherwise. The variance can be seen as the expected error minus the bias. Note that $1 - P_{\Omega_{TR}}(g_B(\boldsymbol{x})) \leq 1$ which implies that the variance can be negative for biased observations.

- Breiman (1996*a*):

$$Bias_{B96}(\boldsymbol{x}) = I(\bar{g}(\boldsymbol{x}) \neq g_B(\boldsymbol{x})) \cdot [P(g_B(\boldsymbol{x})) - \sum_k P(k) \cdot P_{\Omega_{TR}}(k)]$$
$$Var_{B96}(\boldsymbol{x}) = I(\bar{g}(\boldsymbol{x}) = g_B(\boldsymbol{x})) \cdot [P(g_B(\boldsymbol{x})) - \sum_k P(k) \cdot P_{\Omega_{TR}}(k)].$$

  Breiman's definitions ensure that both bias and variance are always non-negative, since $max_{\Omega_{TR}}(\sum_k P(k) \cdot P_{\Omega_{TR}}(k)) = max_k(P(k)) = P(g_B(\boldsymbol{x}))$. Furthermore, the variance of a constant model is zero. To see this, consider a classifier that provides identical predictions over all the training sets at a point $\boldsymbol{x}$. Then either the point is biased, which means that the variance is zero, or the point is unbiased and $P_{\Omega_{TR}}(k) = 1$ (where $k = g_B(\boldsymbol{x})$), leading to $Var_{B96}(\boldsymbol{x}) = 0$. The latter scenario also guarantees that the Bayes classifier will always have a bias and variance equal to zero. Unfortunately the definitions attribute all the reducible error either entirely to bias or entirely to variance. Recalling insights from the Friedman effect, this might make sense for unbiased points. However for biased points it makes more sense to attribute the reducible error to a mixture of both bias and variance.

- Kohavi and Wolpert (1996):

$$Bias_{KW}(\boldsymbol{x}) = \frac{1}{2}\sum_k [P(k) - P_{\Omega_{TR}}(k)]^2$$
$$Var_{KW}(\boldsymbol{x}) = \frac{1}{2}[1 - \sum_k P_{\Omega_{TR}}(k)^2].$$

as estimated by an average over multiple training sets. Furthermore, $P_{\Omega_{TR}}(g_B(\boldsymbol{x}))$ represents the probability of the class predicted by the Bayes classifier given the distribution obtained by averaging over multiple training sets. Finally, $P(\bar{g}(\boldsymbol{x}))$ is the true probability of the class predicted by $\bar{g}$ at a point $\boldsymbol{x}$. A similar summary can be found in Appendix A of Geurts (2002).

The bias is defined in terms of a difference in probability distributions. However, a decrease in this bias does not necessarily translate into a reduction of the expected error. Also, consider a classifier for which $P_{\Omega_{TR}}(k) = 1$, where $k = g_B(\boldsymbol{x})$, then the expected error would be equal to the Bayes error rate. But although the variance is zero for such a classifier, the above definitions would assign a positive bias given that the Bayes error rate is not equal to zero.

- Tibshirani (1996):

$$Bias_T(\boldsymbol{x}) = P(g_B(\boldsymbol{x})) - P(\bar{g}(\boldsymbol{x}))$$
$$Var_T(\boldsymbol{x}) = 1 - P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x})).$$

Since $max\{P(g(\boldsymbol{x}))\} = P(g_B(\boldsymbol{x}))$, the bias is always non-negative and equal to zero in the case of the Bayes classifier. However, even though the variance is also non-negative, it is not always equal to zero in the case of the Bayes classifier. This is because the irreducible error is included in the definition of the variance.

- Heskes (1998):

$$Bias_H(\boldsymbol{x}) = \sum_c P(c)(P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x})) - P_{\Omega_{TR}}(k))$$
$$Var_H(\boldsymbol{x}) = 1 - P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x})).$$

Heskes' definitions for bias and variance for classification stem from a more general argument based on the Kullback-Leibler divergence (Kullback and Leibler, 1951). The divergence measures the difference between two densities, however strictly it is not a distance function since it is not symmetric. Using the limit of a log-likelihood error decomposition the above definitions are obtained. Unfortunately, Heskes (1998) notes that by taking the limit, natural interpretations of the associated quantities are lost.

- Breiman (2000):

$$Bias_{B00}(\boldsymbol{x}) = [P(g_B(\boldsymbol{x})) - P(\bar{g}(\boldsymbol{x}))] \cdot P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x}))$$
$$Var_{B00}(\boldsymbol{x}) = \sum_{k \neq \bar{g}(\boldsymbol{x})} [P(g_B(\boldsymbol{x})) - P(k))] \cdot P_{\Omega_{TR}}(k).$$

In a paper discussing the idea of increasing accuracy by randomising the response variable when performing prediction, Breiman (2000) derives a second set of definitions for bias and variance. Again the bias is zero when $g(\boldsymbol{x})$ agrees with the Bayes classifier, and positive otherwise. In addition, the variance is always non-negative and zero for the constant model given that the prediction is unbiased. That is, $P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x}) = k) = 1$, where $k = g_B(\boldsymbol{x})$. However, if the constant model is biased, $P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x}) = k) = 1$, where $k \neq g_B(\boldsymbol{x})$ and the variance will be positive.

- Domingos (2000):

$$Bias_D(\boldsymbol{x}) = I(\bar{g}(\boldsymbol{x}) \neq g_B(\boldsymbol{x}))$$
$$Var_D(\boldsymbol{x}) = c_2 \cdot [1 - P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x}))],$$

  where $c_2 = 1$ if $\bar{g}$ is unbiased at $\boldsymbol{x}$, otherwise $c_2 = -\frac{P_{\Omega_{TR}}(g_B(\boldsymbol{x}))}{1-P_{\Omega_{TR}}(\bar{g}(\boldsymbol{x}))}$. Note that the definitions are identical to those given in (5.3.2) and (5.3.3), except for the added weighting. Domingos (2000) attempts to obtain a *unified decomposition* such that both the errors associated with regression and classification are additively decomposable (for example, in regression $c_2 = 1$, with bias and variance defined as usual). Unfortunately in the classification setting, the weights themselves are functions of bias and variance, which results in a multiplicative relationship. In fact, Friedman (1997) suggested that the effect he observed when analysing bias and variance was due to the relationship between these quantities and the generalisation error of a model being multiplicative and not additive, as is the case in regression.

Arguably the most convincing explanation for the reason why so many different sets of definitions and attempts at finding an appropriate general decomposition exist, is given by James and Hastie (1997) and James (2003). The key observation is that the bias and variance of a model each play two different roles:

1. *Inherent measure*: The bias measures the disagreement between the average model and the Bayes model, and the variance measures the variation of the estimate around its mean.

2. *Effect measure*: The bias measures the proportion of the generalisation error attributed to the disagreement between the average model and the Bayes model (the effect of bias on error), and the variance measures the proportion of the generalisation error attributed to the variability of the estimated model (the effect of variance on error).

James (2003) notes that in regression these two roles are indistinguishable. In other words, the inherent measures of bias and of variance are equal to their respective effects on the generalisation error. However, this is *not the case in general*, and more specifically, it is not the case for the expected 0-1 loss.

5.4 A Generalisation of Bias and Variance for Symmetric Loss

Reconsider the squared error decomposition given in (5.1.3), which can be rewritten as[4]

$$\begin{aligned} Err_{SE}(\boldsymbol{x}) &= Irreducible\ Error + Bias^2 + Variance \\ &= E[L_{SE}(Y, f_B)] + L_{SE}(\bar{f}, f_B) + E[L_{SE}(f, \bar{f})] \\ &= E[(Y - f_B)^2] + (\bar{f} - f_B)^2 + E[(f - \bar{f})^2] \\ &= E[(Y - f_B)^2] + E[(Y - \bar{f})^2 - (Y - f_B)^2] \\ &\quad + E[(Y - f)^2 - (Y - \bar{f})^2] \end{aligned} \tag{5.4.1}$$

since

$$\begin{aligned} E[(Y - \bar{f})^2 - (Y - f_B)^2] &= E[Y^2 - 2Y\bar{f} + \bar{f}^2 - Y^2 + 2Yf_B - f_B^2] \\ &= -2E(Y)\bar{f} + \bar{f}^2 + 2E(Y)f_B - f_B^2 \\ &= \bar{f}^2 - 2f_B\bar{f} + f_B^2 \\ &= (\bar{f} - f_B)^2 \end{aligned}$$

and

$$\begin{aligned} E[(Y - f)^2 - (Y - \bar{f})^2] &= E[Y^2 - 2Yf + f^2 - Y^2 + 2Y\bar{f} - \bar{f}^2] \\ &= -2E(Y)E(f) + E(f^2) + 2E(Y)\bar{f} - \bar{f}^2 \\ &= -2f_B\bar{f} + E(f^2) + 2f_B\bar{f} - \bar{f}^2 \\ &= Var(f) + E(f)^2 - \bar{f}^2 \\ &= E[(f - \bar{f})^2]. \end{aligned}$$

Finally, (5.1.3) becomes

$$\begin{aligned} Err_{SE}(\boldsymbol{x}) = \sigma_\epsilon^2 &+ E[L_{SE}(Y, \bar{f}) - L_{SE}(Y, f_B)] \\ &+ E[L_{SE}(Y, f) - L_{SE}(Y, \bar{f})]. \end{aligned} \tag{5.4.2}$$

The second term in (5.4.2) measures the effect on generalisation error from the

[4]Here the argument $\boldsymbol{x}$ is omitted for convenience, in other words $f(\boldsymbol{x})$ is simply written as f.

expected difference in loss between the average model and the Bayes classifier. The third term measures the effect on generalisation error from the expected difference in loss between the specific estimate f and the average model. However, the decomposition given in (5.4.2) is not restricted to squared error loss and is valid for any symmetric loss function. Therefore, for an estimate h of a response S (numeric or categorical) at $\boldsymbol{x}$, with

$$\sigma(\boldsymbol{x}) = E[L(S, h_B)] \tag{5.4.3}$$
$$SE(\boldsymbol{x}) = E[L(S, \bar{h}) - L(S, h_B)] \tag{5.4.4}$$
$$VE(\boldsymbol{x}) = E[L(S, \hat{h}) - L(S, \bar{h})], \tag{5.4.5}$$

where h_B is the Bayes model, $\bar{h}$ is the average model and $L(\cdot, \cdot)$ is any symmetric loss, a general decomposition is given as

$$Err(\boldsymbol{x}) = \sigma(\boldsymbol{x}) + SE(\boldsymbol{x}) + VE(\boldsymbol{x}). \tag{5.4.6}$$

James (2003) refer to $SE(\boldsymbol{x})$ and $VE(\boldsymbol{x})$ as the *systematic effect* and the *variance effect* respectively. In regression with squared error loss, the systematic and variance effects are indistinguishable from bias and variance. However, in classification the situation is not the same.

Consider the expected 0-1 loss, $E[L_{0-1}(C, g)] = P(g \neq C)$, then the analogous decomposition of (5.4.1) is

$$\begin{aligned}
&\textit{Irreducible Error} + \textit{Bias} + \textit{Variance} \\
&= E[L_{0-1}(C, g_B)] + L_{0-1}(g_B, \bar{g}) + E[L_{0-1}(g, \bar{g})] \\
&= P(g_B \neq C) + I(\bar{g} \neq g_B) + P_{\Omega_{TR}}(\bar{g} \neq g) \\
&\neq P(g_B \neq C) + [P(\bar{g} \neq C) - P(g_B \neq C)] + [P(g \neq C) - P(\bar{g} \neq C)] \\
&= \sigma_{0-1}(\boldsymbol{x}) + E[L_{0-1}(C, g) - L_{0-1}(C, g_B)] + E[L_{0-1}(C, g) - L_{0-1}(C, \bar{g})] \\
&= \sigma_{0-1}(\boldsymbol{x}) + SE_{0-1}(\boldsymbol{x}) + VE_{0-1}(\boldsymbol{x}) \\
&= \textit{Irreducible Error} + \textit{Systematic effect} + \textit{Variance effect}
\end{aligned}$$

Therefore, in classification the effect of bias and variance on generalisation error is not equal to the inherent measures of bias and variance. To see this more clearly, consider the distributions given in Figure 5.6 (James, 2003). The associated quantities for the first classifier are as follows:

$$
\begin{aligned}
Err^1_{0-1}(\boldsymbol{x}) &= 1 - 0.6(0.3) + 0.3(0.5) + 0.1(0.2) = 0.65 \\
\sigma^1_{0-1}(\boldsymbol{x}) &= P(g_B \neq C) = 0.4 \\
Bias^1_{0-1}(\boldsymbol{x}) &= I(\bar{g} \neq g_B) = 1 \\
Var^1_{0-1}(\boldsymbol{x}) &= P_{\Omega_{TR}}(g \neq \bar{g}) = 0.5 \\
SE^1_{0-1}(\boldsymbol{x}) &= P(\bar{g} \neq C) - P(g_B \neq C) = 0.7 - 0.4 = 0.3 \\
VE^1_{0-1}(\boldsymbol{x}) &= P(g \neq C) - P(\bar{g} \neq C) = 0.65 - 0.7 = -0.05.
\end{aligned}
$$

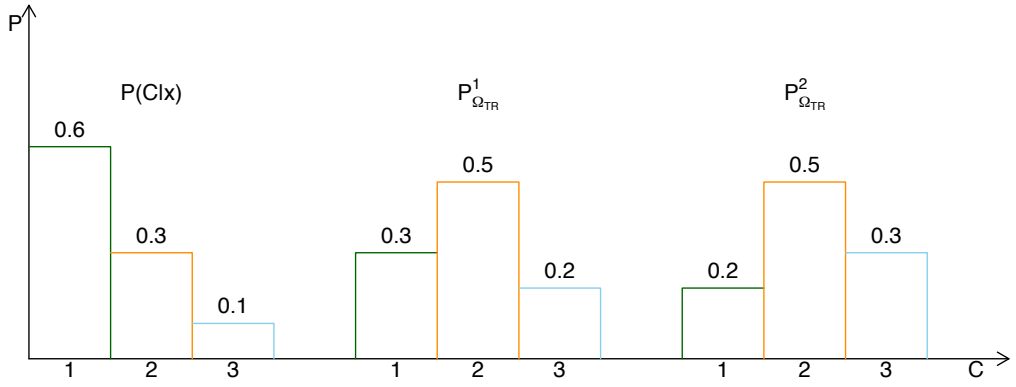

Figure 5.6: Class distributions for a three class classification task with both estimated distributions having equal variance. The true distribution is given on the left.

It follows that

$$Err^1_{0-1}(\boldsymbol{x}) = 0.65 = 0.4 + 0.3 - 0.05 = \sigma^1_{0-1}(\boldsymbol{x}) + SE^1_{0-1}(\boldsymbol{x}) + VE^1_{0-1}(\boldsymbol{x})$$

while

$$\sigma^1_{0-1}(\boldsymbol{x}) + Bias^1_{0-1}(\boldsymbol{x}) + Var^1_{0-1}(\boldsymbol{x}) = 0.4 + 1 + 0.5 = 1.9 \neq Err^1_{0-1}(\boldsymbol{x}).$$

Note the minus sign of the variance effect, meaning that the effect on generalisation error of the variance is actually improving classification performance (another example of the Friedman effect). Furthermore, it is interesting given $Var^1_{0-1}(\boldsymbol{x}) = Var^2_{0-1}(\boldsymbol{x}) = 0.5$, to find

$$VE^1_{0-1}(\boldsymbol{x}) = -0.05 < VE^2_{0-1}(\boldsymbol{x}) = 0.7 - 0.7 = 0.$$

Therefore, even with equal variance, the first classifier has a greater *effect* on generalisation error as a result of variation. This makes sense because the variation around the average model for the first classifier is more concentrated at

the first class (the class chosen by the Bayes model) than the second classifier and is more likely to agree with the Bayes classifier by chance.

The following relationships between bias and the systematic effect, and between variance and the variance effect are given in James (2003) and proven here. This is done for 0-1 loss, but hold for any convex loss function.

- *The bias is identical to the systematic effect if the Bayes error rate is zero.*

  PROOF: If $\boldsymbol{x}$ is a biased point and the Bayes error rate is zero, then

  $$\begin{aligned} Bias^1_{0-1}(\boldsymbol{x}) = I(\bar{g} \neq g_B) = 1 &= P(\bar{g} \neq C) - P(g_B \neq C) \\ &= 1 - 0 = SE^1_{0-1}(\boldsymbol{x}), \end{aligned}$$

  and similarly, the two quantities are equal for unbiased points.

- *The systematic effect will be zero if the bias is equal to zero.*

  PROOF: If the bias is zero, it implies that $P(\bar{g} \neq C) = P(g_B \neq C)$ such that

  $$SE^1_{0-1}(\boldsymbol{x}) = P(\bar{g} \neq C) - P(g_B \neq C) = 0.$$

- *The variance effect will be zero if the variance is equal to zero.*

  PROOF: If the variance is zero, the model is the constant model so that $P(g = k) = P(\bar{g} = k) = 1$ for some class $k \in C$, independent of the training data. Therefore, $P(g \neq C) = P(\bar{g} \neq C)$ and

  $$VE^1_{0-1}(\boldsymbol{x}) = P(g \neq C) - P(\bar{g} \neq C) = 0.$$

Although the above relationships exist, there is no guarantee that they will hold when the Bayes error rate, bias or variance are in fact non-zero. It is interesting however that James (2003) finds the median correlation between the bias and the systematic effect to be as high as 96.6% in his simulation study conducted on six different data sets. In addition, the median correlation between variance and the variance effect is found to be 81.1%. Therefore, James (2003) remarks that although these quantities are not always strictly related, bias and variance seem to be good predictors of the their respective effects on generalisation error.

5.5 The Effects of Randomisation and Aggregation

Thus far in this chapter, the conversation has been very general, but in this section the focus is returned to random forests. Consider a single randomised tree classifier for which the expected 0-1 loss at a point $\boldsymbol{x} = \boldsymbol{x}$ is given by

$$Err^{T}_{0-1}(\boldsymbol{x}) = 1 - \sum_{k} P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x}, \Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x}), \tag{5.5.1}$$

where $\bar{t}(\boldsymbol{x}) = \arg\max_k E_{\Omega_{TR}}[I(t(\boldsymbol{x}, \Theta) = k)]$ is the majority vote over multiple training sets. Randomisation of a single tree classifier is likely to increase both the bias and variance, but with the former less severely affected (Geurts *et al.*, 2006). It can be argued that the systematic effect will also increase, however the variance effect might change in either direction depending on the similarity between the estimated distribution and the truth. Now consider a random forest majority vote classifier defined as

$$\bar{t}_{RF}(\boldsymbol{x}, \Theta) = \arg\max_k E_{\Omega_{TR}}\big\{I(\arg\max_l E_{\Theta}[I(t(\boldsymbol{x}, \Theta) = l)]) = k)\big\}, \tag{5.5.2}$$

where an additional expectation is taken over Θ, representing the independent and identically distributed random vectors characterising each randomised tree. The expected 0-1 loss for a random forest is

$$Err^{RF}_{0-1}(\boldsymbol{x}) = 1 - \sum_{k} P_{\Omega_{TR}}(\bar{t}_{RF}(\boldsymbol{x}, \Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x}), \tag{5.5.3}$$

which is similar to expression (5.5.1). However, for (5.5.3) it is possible to partition the error between biased and unbiased points over the random vector Θ. In more detail, let the set $B_{\Theta} = \{\boldsymbol{x}|\bar{t}(\boldsymbol{x}, \Theta) \neq g_B(\boldsymbol{x})\}$ be the set of biased points and B^c_{Θ} represent the set of unbiased points. Then the expected loss for a random forest can be written as[5]

$$\begin{aligned} Err^{RF}_{0-1}(\boldsymbol{x}) = 1 - \Bigg[&\int_{\boldsymbol{x}\in B_{\Theta}} \sum_{k} P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x}, \Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x})d\Theta \\ &+ \int_{\boldsymbol{x}\in B^c_{\Theta}} P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x}, \Theta) = g_B(\boldsymbol{x}))d\Theta \Bigg]. \end{aligned} \tag{5.5.4}$$

[5]The presented argument is similar to one found in Breiman (1996*a*).

Therefore, if most points are unbiased, or in other words if B^c_Θ is large,

$$\begin{aligned} Err^{RF}_{0-1}(\boldsymbol{x}) &\approx 1 - P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x},\Theta) = g_B(\boldsymbol{x})) \\ &= 1 - max\Big\{\sum_k P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x},\Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x})\Big\} \\ &= min(Err^T_{0-1}(\boldsymbol{x})). \end{aligned} \tag{5.5.5}$$

From the above, if the majority of points are unbiased, a random forest will reach a nearly optimal error rate. Conversely, if B^c_Θ is small, it could be the case that

$$\sum_k P_{\Omega_{TR}}(\bar{t}_{RF}(\boldsymbol{x},\Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x}) < \sum_k P_{\Omega_{TR}}(\bar{t}(\boldsymbol{x},\Theta) = k|\boldsymbol{x})P(k|\boldsymbol{x}), \tag{5.5.6}$$

causing a random forest model to perform worse than a single tree. Unfortunately, the inner workings of randomisation and aggregation relating bias and variance and their respective effects to generalisation error remain difficult to discern theoretically.[6] Based on insights from regression arguments Geurts (2002) provides an illustration of the likely effects of randomisation and aggregation on bias and variance similar to Figure 5.7 below.

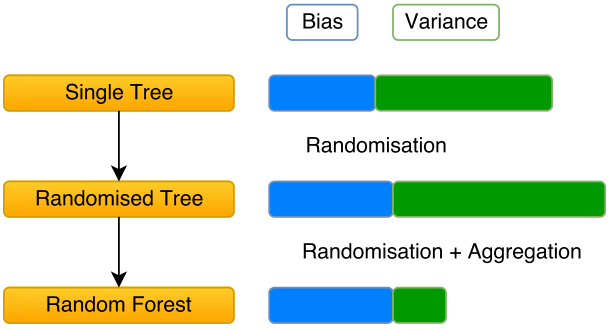

Figure 5.7: The likely effects on bias and variance from randomisation and aggregation.

[6] An avenue of interest in this regard is the following. **Theorem 6** found in Domingos (2000) (and proved) states that: *The margin of a learner on an example* $\boldsymbol{x}$ *can be expressed as* $mg(X,C) = \pm[2Bias_{0-1}(\boldsymbol{x}) - 1][2Var_{0-1}(\boldsymbol{x}) - 1]$, *with positive sign if* $g_B = C$ *and negative otherwise.* The starting point for Breiman's proof for the generalisation performance of a random forest is in fact the margin, $Err^* \leq Var(mg)/E(mg)^2$. Hence an attempt was made at finding a satisfactory expression connecting Err^* to bias and variance, unfortunately to no avail.

Combining Figure 5.7 with remarks from James (2003), the following is speculated:

- The classification performance of a single tree depends on the nature of its systematic and variance effects. In turn these quantities have empirically been found to be highly correlated with bias and variance.
- Adding a source of randomisation to a single tree will likely cause the bias and the variance to increase. The systematic effect will likely also increase, but the direction of the variance effect is less clear. The additional randomness could prove beneficial for biased points, but detrimental for unbiased points.
- Adding an aggregation step where an ensemble of randomised trees are used to construct a random forest will decrease the variance, whereas the bias will be less affected. The systematic effect is also likely to remain similar to that of a single randomised tree, however the variance effect might decrease or increase, depending on the nature of the bias. If most points are unbiased, the variance effect will reduce the generalisation error. The opposite is true in the case of most points being biased.

5.6 An Empirical Investigation

The difficulties of a theoretical analysis of bias and variance for random forests naturally leave as an alternative an empirical investigation. In this section, bias and variance and their respective effects are estimated on simulated data sets. A comparison is conducted between a single classification tree, bagging, Forest-RI and boosting.

5.6.1 Data sets

In total 16 different simulated data sets were used in the empirical investigation. The first set of four simulated data sets consists of observations drawn from a multivariate normal distribution. Each data set has $p = 15$ input variables with the pairwise correlation between all variables configured as follows: $\rho = 0.9$ (highly correlated), $\rho = 0.5$ (fairly correlated), $\rho = 0.1$ (weakly correlated) and $\rho = 0$ (uncorrelated). All the input variables are associated with the response, which is coded as $C = 1$, if $1/(1 + e^{-\sum_{j=1}^{15} X_j}) > 0.5$, or as $C = 0$ otherwise.[7]

The second set of simulated data sets are generated using the following simulation setup: let $X_1, ..., X_p \sim U[0, 1]$ and $C = 1$ if $q + (1 - 2q) \cdot I(\sum_{l=1}^{J} X_l >$

[7]The data were generated using the *pensim* R package.

$J/2) > 0.5$, otherwise let $C = 0$. Here $J < p$ and $0 < q < 1$ (Mease and Wyner, 2008). This implies that the response C only depends on $X_1, X_2, ...X_J$. The remaining $p - J$ variables are noise. With $p = 30$ and $q = 0.15$[8], four configurations were chosen: $J = 2$ (mostly noise), $J = 5$ (fairly noisy), $J = 15$ (half signal/half noise) and $J = 20$ (mostly signal).

In addition to the above eight simulated data sets, eight popular data sets where selected from among the *machine learning benchmark problems* provided in the *mlbench* R package (Leisch and Dimitriadou, 2010). Some of these problems were also used in Breiman (1996*a*) and Breiman (2000). Figure 5.8 provides an illustration of the simulated data in two dimensions, with a description of each simulation configuration given as follows:

- *2dnormals*: A classification task with six classes, each generated from a two-dimensional normal distribution with unit standard deviation around a circle with radius $\sqrt{6}$.
- *Twonorm*: A binary classification task with 20 inputs where the data simulated for the first class are drawn from a multivariate normal, $N\big(\boldsymbol{\mu}_1 = (\mu_1, ..., \mu_{20}), I\big)$, with mean components $\mu_1 = ... = \mu_{20} = 2/(20)^{\frac{1}{2}}$ and covariance matrix I, the identity matrix. The data for the second class are drawn from $N\big(\boldsymbol{\mu}_2 = (-\mu_1, ..., -\mu_{20}), I\big)$.
- *Threenorm*: A binary classification task with 20 inputs where the data simulated for the first class are drawn either from $N\big(\boldsymbol{\mu}_1, I\big)$ or from $N\big(\boldsymbol{\mu}_2, I\big)$ with equal probability. The data for the second class are drawn from $N\big((\mu_1, -\mu_2, \mu_3..., \mu_{19}, -\mu_{20}), I\big)$, *i.e.* a normal distribution with means that alternate in sign.
- *Ringnorm*: A binary classification task with 20 input variables where the data simulated for the first class are drawn from $N\big(\mathbf{0}, 4I\big)$. The data for the second class are drawn from $N\big(\boldsymbol{\mu}_1/2, I\big)$.
- *Circle*: A binary classification task with 20 inputs where $X_1, ..., X_{20} \sim U(0, 1)$. The data simulated for class one form a p-dimensional ball inside the hypercube, with the data for the second class filling the remainder of the cube. The size of the centre ball is such that both classes have a prior probability equal to 1/2.
- *Cassini*: A multi-class classification task where there are three classes and where each input is uniformly distributed in two-dimensional space. One class forms a circle in the middle, with the other two wrapping around the circle from above and below.

[8]The parameter q controls the Bayes error rate and the choice is rather arbitrary. For example, Mease and Wyner (2008) decided on $q = 0.1$ for some of their experiments and on $q = 0.2$ for others.

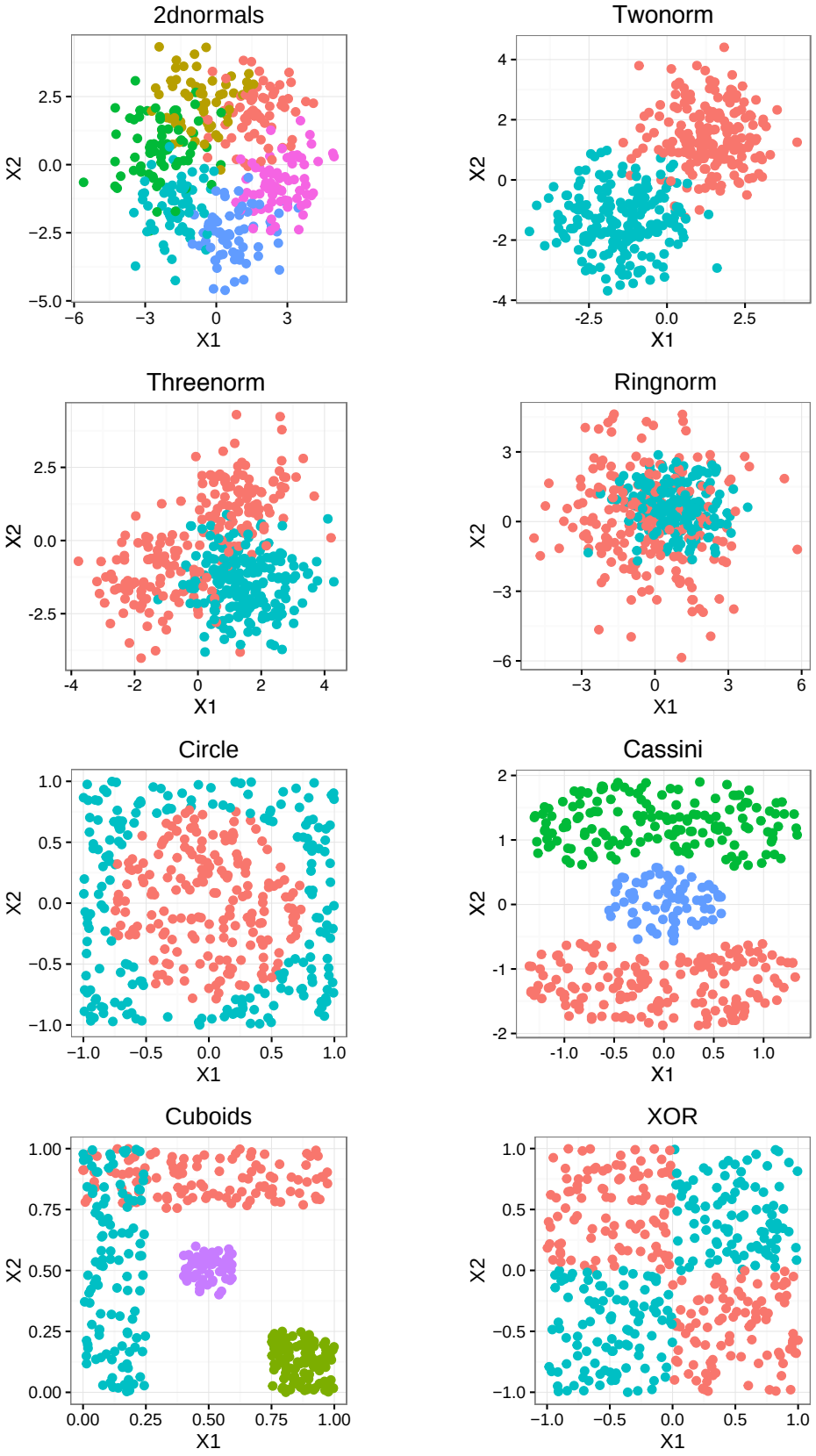

Figure 5.8: A two-dimensional representation of the simulated data from the *machine learning benchmark problems* found in the *mlbench* R package.

- *Cuboids*: A multi-class classification task where there are four classes and where each input is uniformly distributed in three-dimensional space. Each class occupies a cube-like shape.
- *XOR*: A binary classification task in two-dimensional space, representing the *exclusive OR* problem (true if one of two arguments is true, and false otherwise).

5.6.2 Experimental design

To approximate the necessary probabilities (expectations over indicator functions), the first step was to simulate 100 different training sets of size 400 and to fit a model to each training set. For each model fitted, predictions were made on a test set of size 1000. Using the known data generating mechanism to obtain the Bayes classifier, together with the predictions from each fit, the bias, variance, systematic and variance effects could be computed. More specifically, let $\Omega_1, ..., \Omega_D$ denote the $D = 100$ training sets and let $\Omega_{te} = \{(\boldsymbol{x}_{0i}, c_{0i}), i = 1, ..., N_0\}$ be the test set. Further, let $\bar{g}(\boldsymbol{x}) = \arg\max_k \frac{1}{D}\sum_{d=1}^{D} I(\hat{g}_d(\boldsymbol{x}) = k)$. Then,

$$\begin{aligned}
\widehat{Bias} &= \frac{1}{N_0}\sum_{i=1}^{N_0} I(\bar{g}(\boldsymbol{x}_{0i}) \neq g_B(\boldsymbol{x}_{0i})),\\
\widehat{Variance} &= \frac{1}{D}\sum_{d=1}^{D}\frac{1}{N_0}\sum_{i=1}^{N_0} I(g_{\Omega_d}(\boldsymbol{x}_{0i}) \neq \bar{g}(\boldsymbol{x}_{0i})),\\
\widehat{SE} &= \frac{1}{N_0}\sum_{i=1}^{N_0} I(\bar{g}(\boldsymbol{x}_{0i}) \neq C_{0i}) - \frac{1}{N_0}\sum_{i=1}^{N_0} I(g_B(\boldsymbol{x}_{0i}) \neq C_{0i}),\\
\widehat{VE} &= \frac{1}{D}\sum_{d=1}^{D}\frac{1}{N_0}\sum_{i=1}^{N_0} I(g_{\Omega_d}(\boldsymbol{x}_{0i}) \neq C_{0i}) - \frac{1}{N_0}\sum_{i=1}^{N_0} I(\bar{g}(\boldsymbol{x}_{0i}) \neq C_{0i}).
\end{aligned}$$

It is typically the case that the tuning parameters of an algorithm heavily affects its bias and variance (James *et al.*, 2013). Therefore, before each fit ten-fold cross-validation was performed to find the optimal tuning parameters for each algorithm among a pre-specified grid of available parameters. The pre-specified grids were chosen as follows:

- **Trees**: The *cost-complexity* parameter was chosen from $cp = \{0.1, 0.2, ..., 0.9, 1.0\}$.

- **Bagging**: The number of unpruned trees were specified as $B = 200$.[9] Note that this means that bagging had no tuning parameters.
- **Forest-RI**: The number of unpruned trees were taken as $B = 200$, with the subset size of randomly selected variables selected from $\xi = \{1, 3, 5, ..., p-1\}$.
- **Boosting**: The number of trees were $B = 200$, tree interaction depth was either one or six, and the step-length factor $\nu = \{0.01, 0.05, 0.1\}$.[10] For binary classification, the exponential loss was used, and in the case of multi-class problems, the multinomial loss.

The ensemble size was selected after observing that substantial gains in performance are rarely made beyond $B = 200$ for moderately sized data sets (refer to Figures 3.2, 4.1, 4.2 and 4.3).

5.6.3 Results

The results pertaining to the first eight simulated data sets are given in Table 5.1, with the results for the *mlbench* problems provided in Table 5.2. In each table, the values in bold represent the minimum achieved for a particular quantity among the algorithms. The following conclusions are drawn:

- *Single tree vs. ensemble*: For each data set randomisation and aggregation succeeded in drastically reducing the variance as well as the variance effect. Interestingly, the bias and systematic effect was either also reduced or remained equal to that of a single tree. Therefore, the empirical findings are fairly in line with the speculations made in Section 5.5.
- *Bagging vs. Forest-RI*: For the majority of data sets, the additional randomisation at each node of a tree in Forest-RI resulted in a further reduction of the variance and of the variance effect when compared to bagging. Interestingly, bagging on the other hand managed to reduce the bias and systematic effect substantially in many of the data sets. This is the most evident in the case of data simulated from multivariate normal distributions (Sim 1 to Sim 4 and Sim 10). A possible explanation is that the distribution of the mean of several random variables approximates a normal distribution, and that this is exactly what bagging is modelling using trees.

[9] According to Hastie *et al.* (2009), tree depth in random forests has a small effect on prediction performance. The trees where grown until each node had a maximum size of five.

[10] Hastie *et al.* (2009) note that trees with a depth of between four and eight perform well in boosting. They recommend setting tree depth equal to six. Since stumps are also a popular choice in boosting, an interaction depth of one was included in the grid.

Table 5.1: Estimated bias, variance, systematic effect and variance effect on simulated data. Values in bold indicate row-wise minima.

Name	Data	Quantity	Tree	Bagging	Forest-RI	Boosting
Sim 1	**mvnorm** $p = 15$, $\rho = 0.9$	Error	0.105	0.044	**0.038**	0.043
		Bayes Error	0.028	0.028	0.028	0.028
		Systematic Effect	0.015	**0.003**	0.004	0.005
		Variance Effect	0.062	0.013	**0.006**	0.010
		Bias	0.025	**0.005**	0.006	0.007
		Variance	0.097	0.028	**0.018**	0.026
Sim 2	**mvnorm** $p = 15$, $\rho = 0.5$	Error	0.233	0.075	**0.061**	0.068
		Bayes Error	0.040	0.040	0.040	0.040
		Systematic Effect	0.034	**0.009**	0.010	0.013
		Variance Effect	0.159	0.026	**0.011**	0.015
		Bias	0.060	**0.013**	0.022	0.033
		Variance	0.224	0.056	**0.034**	0.043
Sim 3	**mvnorm** $p = 15$, $\rho = 0.1$	Error	0.368	0.152	**0.129**	0.130
		Bayes Error	0.078	0.078	0.078	0.078
		Systematic Effect	0.064	**0.011**	0.014	0.026
		Variance Effect	0.226	0.063	0.037	**0.026**
		Bias	0.098	**0.027**	0.028	0.040
		Variance	0.355	0.120	0.085	**0.084**
Sim 4	**mvnorm** $p = 15$, $\rho = 0$	Error	0.427	0.239	0.216	**0.200**
		Bayes Error	0.141	0.141	0.141	0.141
		Systematic Effect	0.074	**0**	**0**	0.004
		Variance Effect	0.212	0.101	0.077	**0.055**
		Bias	0.168	**0.031**	0.044	0.060
		Variance	0.405	0.197	0.163	**0.136**
Sim 5	**Mease-Wyner (2008)** $p = 30$, $J = 2$	Error	0.295	**0.212**	0.214	**0.212**
		Bayes Error	0.147	0.147	0.147	0.147
		Systematic Effect	0.050	**0.002**	0.008	0.017
		Variance Effect	0.098	0.063	0.059	**0.048**
		Bias	0.072	**0.002**	0.008	0.021
		Variance	0.202	0.094	0.096	**0.092**
Sim 6	**Mease-Wyner (2008)** $p = 30$, $J = 5$	Error	0.390	0.275	0.272	**0.259**
		Bayes Error	0.143	0.143	0.143	0.143
		Systematic Effect	0.075	0.021	**0.017**	0.021
		Variance Effect	0.172	0.111	0.112	**0.095**
		Bias	0.095	**0.029**	**0.029**	0.037
		Variance	0.338	0.184	0.179	**0.159**
Sim 7	**Mease-Wyner (2008)** $p = 30$, $J = 15$	Error	0.452	0.308	0.304	**0.284**
		Bayes Error	0.136	0.136	0.136	0.136
		Systematic Effect	0.117	0.034	**0.015**	0.020
		Variance Effect	0.199	0.138	0.153	**0.128**
		Bias	0.155	0.046	0.029	**0.028**
		Variance	0.424	0.233	0.228	**0.201**
Sim 8	**Mease-Wyner (2008)** $p = 30$, $J = 20$	Error	0.455	0.318	0.308	**0.290**
		Bayes Error	0.134	0.134	0.134	0.134
		Systematic Effect	0.171	0.043	0.034	**0.023**
		Variance Effect	0.150	0.141	0.140	**0.133**
		Bias	0.237	0.059	0.046	**0.031**
		Variance	0.421	0.248	0.236	**0.212**

Table 5.2: Estimated bias, variance, systematic effect and variance effect for *mlbench* problems. Values in bold indicate row-wise minima.

Name	Data	Quantity	Tree	Bagging	Forest-RI	Boosting
Sim 9	**2d Norms** $p = 2$, $K = 6$	Error	0.420	0.301	0.292	**0.273**
		Bayes Error	0.243	0.243	0.243	0.243
		Systematic Effect	0.076	0.004	0.005	**0.002**
		Variance Effect	0.101	0.054	0.044	**0.028**
		Bias	0.157	**0.019**	**0.019**	**0.019**
		Variance	0.290	0.158	0.138	**0.099**
Sim 10	**Two-norm** $p = 20$, $K = 2$	Error	0.319	0.057	**0.033**	0.040
		Bayes Error	0.024	0.024	0.024	0.024
		Systematic Effect	0.025	**0**	0.003	0.004
		Variance Effect	0.270	0.033	**0.006**	0.012
		Bias	0.037	**0.008**	0.011	0.014
		Variance	0.314	0.047	**0.018**	0.025
Sim 11	**Three-norm** $p = 20$, $K = 2$	Error	0.392	0.177	**0.157**	0.167
		Bayes Error	0.085	0.085	0.085	0.085
		Systematic Effect	0.092	0.039	**0.037**	0.048
		Variance Effect	0.215	0.053	0.035	**0.034**
		Bias	0.132	0.077	**0.075**	0.088
		Variance	0.368	0.127	**0.094**	0.102
Sim 12	**Ring-norm** $p = 20$, $K = 2$	Error	0.300	0.087	**0.042**	0.051
		Bayes Error	0.018	0.018	0.018	0.018
		Systematic Effect	0.161	0.013	**0.007**	0.019
		Variance Effect	0.121	0.056	0.017	**0.014**
		Bias	0.177	0.023	**0.021**	0.029
		Variance	0.256	0.077	**0.030**	0.034
Sim 13	**Circle** $p = 20$, $K = 2$	Error	0.171	0.168	0.168	**0.140**
		Bayes Error	0	0	0	0
		Systematic Effect	0.171	0.171	0.171	**0.152**
		Variance Effect	0	−0.003	−0.003	**-0.012**
		Bias	0.171	0.171	0.171	**0.152**
		Variance	**0**	0.007	0.004	0.046
Sim 14	**Cassini** $p = 2$, $K = 3$	Error	0.003	0.003	**0.002**	0.004
		Bayes Error	0	0	0	0
		Systematic Effect	0.001	0.001	**0**	0.001
		Variance Effect	**0.002**	**0.002**	**0.002**	0.003
		Bias	0.001	0.001	**0**	0.001
		Variance	0.003	0.003	**0.002**	0.003
Sim 15	**Cuboids** $p = 3$, $K = 4$	Error	0.074	0.0001	**0**	0.0002
		Bayes Error	0	0	0	0
		Systematic Effect	**0**	**0**	**0**	**0**
		Variance Effect	0.074	0.0001	**0**	0.0002
		Bias	**0**	**0**	**0**	**0**
		Variance	0.074	0.0001	**0**	0.0002
Sim 16	**XOR** $p = 2$, $K = 2$	Error	0.059	**0.007**	0.009	0.008
		Bayes Error	0	0	0	0
		Systematic Effect	0.002	**0**	**0**	**0**
		Variance Effect	0.057	**0.007**	0.009	0.008
		Bias	0.002	**0**	**0**	**0**
		Variance	0.059	**0.007**	0.009	0.008

- *Forest-RI vs. boosting*: The adaptation strategy employed by the boosting algorithm had a significant effect on variance. This is especially evident in the presence of noise (Sim 5 to Sim 8). In turn, the reduction in variance lead to a substantial reduction in the variance effect over and above that achieved by Forest-RI on several of the data sets.

- *Negative variance effects*: For all of the algorithms, the *circle* data set resulted in most, if not all of the reducible error being attributed to the systematic effect, with a bias identical in magnitude. However, bagging and Forest-RI managed to obtain a variance very close to zero as well as a small negative variance effect. Boosting had the largest variance among the algorithms, but also the largest negative variance effect (equal to -0.012) — a possible demonstration of the Friedman effect.

- *Inherent and effect measure correlation*: As was observed by James (2003), bias and variance seem to be highly correlated with their respective effects on generalisation error. The median correlation between bias and the systematic effect in the empirical study was 93.93%, while the median correlation between the variance and variance effect was 95.61%.

In summary, Table 5.3 provides a win/tie analysis of the results. For each measure represented in the rows of Table 5.3, the number of wins/ties achieved by each algorithm are tallied. For example, the 8/0 entry for Forest-RI indicates that it managed to outperform the other approaches eight times (out of 16) in terms of misclassification error and never had an error rate that was equal to another algorithm.

Table 5.3: Win/Tie analysis of bias, variance, systematic effect and variance effect. An asterisk indicates a significant p-value with $\alpha = 0.05$.

Quantity	(**Tree**)	**Bagging**	**Forest-RI**	**Boosting**	**p-val**
Error	0/0	1/1	**8/0**	6/1	**0.001***
Systematic Effect	0/1	**5/3**	**4/4**	3/2	0.311
Variance Effect	0/1	1/1	3/2	**10/0**	**0.0004***
Bias	0/1	**6/4**	3/4	3/3	0.154
Variance	1/0	1/0	**7/0**	**7/0**	**0.004***
Total	1/3	14/9	**25/10**	29/4	

The final column in Table 5.3 display p-values obtained through a statistical comparison test. The null hypothesis is that there is no difference between bagging, Forest-RI and boosting in terms of their performances, as measured by the corresponding quantity.[11] The test used in this scenario was the *Fried-*

[11]The ensemble methods were clearly superior to classification trees. Therefore, trees were omitted when these tests were conducted.

man aligned ranks test (the reader is referred to Section 7.2, where the topic of statistical comparisons of algorithms is discussed in much greater detail).

Given that the null hypothesis was rejected, Table 5.4 provides three additional p-values per quantity from pairwise comparisons, obtained using the *Shaffer static test*. The null hypothesis is that there is no difference between a particular pair of algorithms. The pairs are displayed in the following order: (**1**) bagging vs. Forest-RI; (**2**) bagging vs. boosting; and (**3**) Forest-RI vs. boosting.

Table 5.4: Adjusted p-values from the Shaffer static post-hoc test used for pairwise comparisons. An asterisk indicates a significant p-value with $\alpha = 0.05$.

Quantity	**1**	**2**	**3**
Error	**0.003***	**0***	0.181
Systematic Effect		N/A	
Variance Effect	**0.009***	**0***	**0.033***
Bias		N/A	
Variance	**0.004***	**0***	0.344

From Tables 5.3 and 5.4 it is seen that bagging was the best performer in terms of bias as measured by wins/ties. This also translated into the best performance with respect to the systematic effect. However, in the case of both quantities, the difference was statistically not significant. Forest-RI and boosting were tied for the top position in terms of variance, with no significant difference detected between the two algorithms. With respect to the variance effect however, boosting significantly outperformed both bagging and Forest-RI. Despite this, Forest-RI managed to achieve the lowest error rate on the highest number of simulation configurations. Although not significantly different from boosting, this might suggest that Forest-RI performs well not because it exclusively reduces either the bias/systematic effect or the variance/variance effect. Rather, Forest-RI seems to be successful in reducing both these quantities simultaneously.

5.6.4 Tuning Parameter Variability

To illustrate the necessity of parameter tuning, Figure 5.9 shows the variation in the selection of the optimal subset size of randomly selected inputs at each node for Forest-RI over 100 training sets. The left side of each panel in Figure 5.9 shows the selected subset sizes, while the right sides are bar plots displaying the frequency of the variable subset sizes selected among the pre-specified parameter grid.

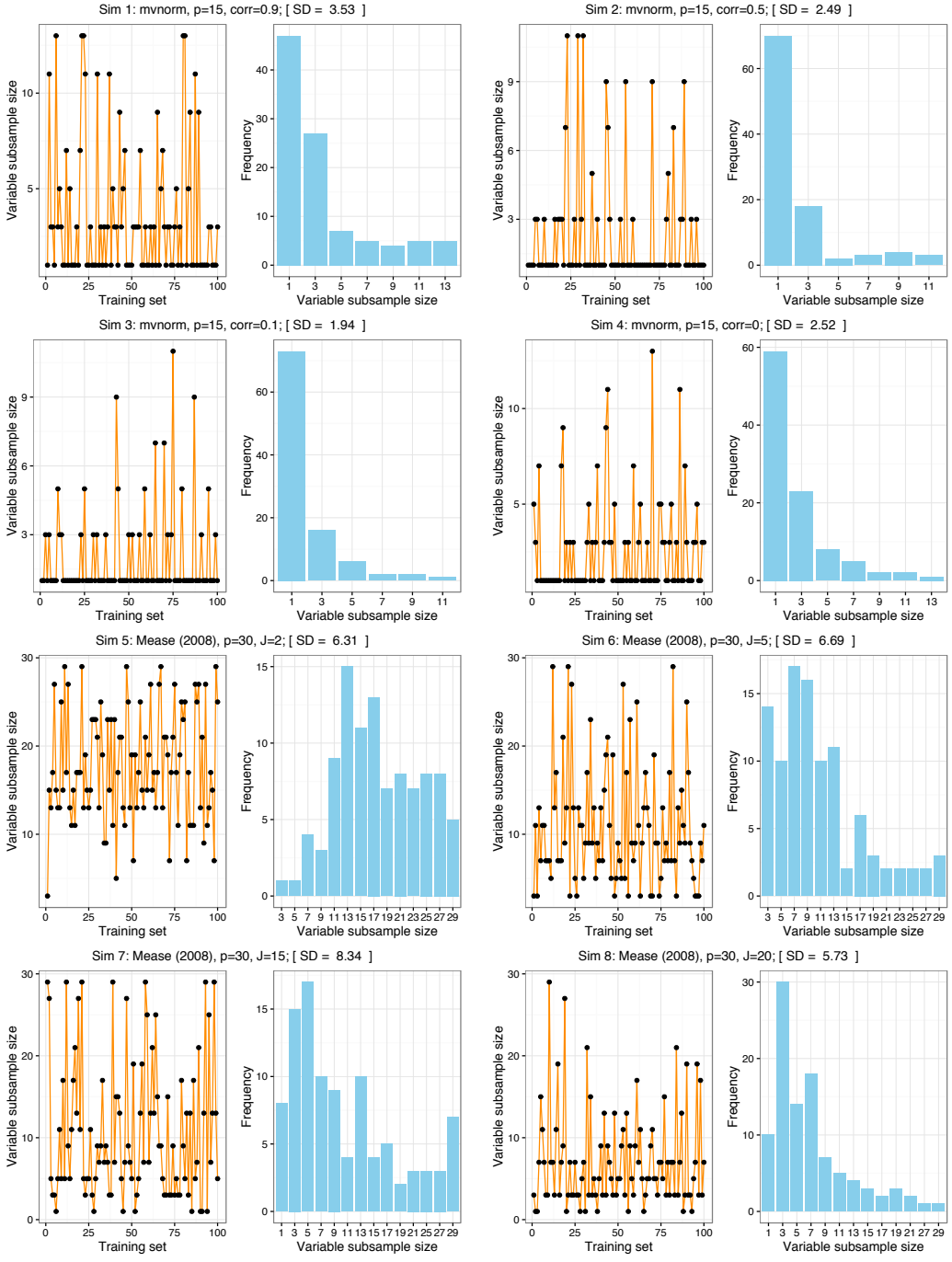

Figure 5.9: Variation in the selection of the optimal subset size of randomly selected input variables at each node for Forest-RI over 100 training sets displayed for the first eight simulation configurations.

For each of the first eight simulation configurations, the overall variability is quite high. However, even as the correlation between inputs decrease (in Sim 1 to Sim 4), the most commonly selected subset size remains equal to one. This might be expected when all the variables are highly correlated, since selecting any one of the variables should provide a similar amount of information at each data split. In contrast, when all of the variables are uncorrelated, it might intuitively be expected that larger subsets would perform better than smaller ones. This is however not the case here.

In Sim 5 to Sim 8, it is shown how the optimal parameters are affected by the amount of noise. As noise decreases, the distribution of subset sizes changes from being skewed to the left (larger subsets preferred) to being skewed to the right (smaller subsets preferred). This makes sense: if only a small number of inputs are associated with the response, then larger subsets are required to ensure these relevant inputs to be included at node splits. On the other hand, if there is little noise and most input variables are useful for splitting, large subsets become unnecessary and might reduce the diversity of the ensemble.

5.7 Concluding Remarks

In regression using squared-error loss, the generalisation error can be decomposed into three parts: the irreducible error, (squared) bias and variance. The latter two quantities are dependent on the estimated model, which if reduced, improves the generalisation error. However, in classification the situation becomes more complicated and finding an analogous decomposition for 0-1 loss is more difficult. The reason is that in regression the quantities measured as bias and variance are indistinguishable from their effects on generalisation error. But with estimated probabilities that are ultimately used for classification, the bias and variance can differ from their respective effects based on the similarity between the estimated distribution and the truth. Therefore, an attempt at generalising bias and variance for any symmetric loss function results in two sets of definitions: one for the inherent measure of bias and variance, and the other for their effects. In regression the two sets are equal, however in classification they are not.

Towards a deeper understanding of random forests, the interest is in how randomisation and aggregation affect these quantities in a classification setting. The already complicated nature of bias and variance, coupled with the complexity of random forests makes it difficult to provide a theoretical analysis, but at least empirically these effects can be studied. Using 16 different simulated data sets it was found that randomisation and aggregation tend to decrease both the variance and the variance effect. Furthermore, the bias and

systematic effect either remain unchanged, or are also reduced.

In the next chapter, the discussion becomes less general as an overview is given of specific random forest algorithms that have been proposed in the literature. The updated road map is presented in Figure 5.10.

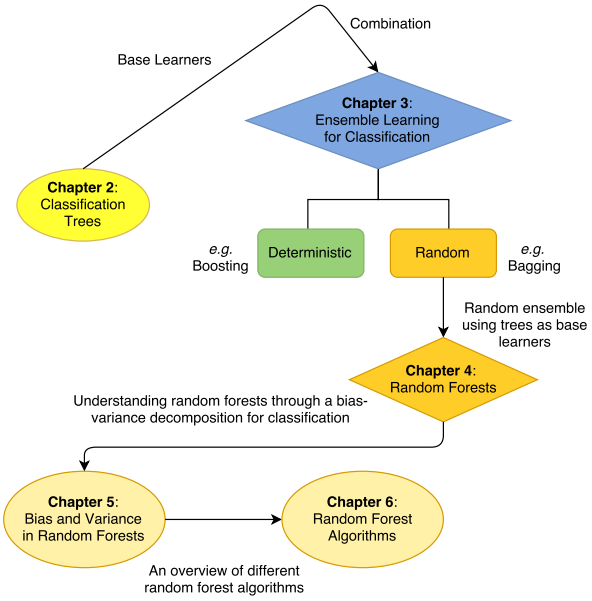

Figure 5.10: Road map to Chapter 6: An overview of different random forest algorithms.

Chapter 6

Random Forest Algorithms

Several different random forest algorithms have been proposed in the literature. Briefly these can be categorised in terms of the sources of randomisation and deterministic modifications used, in this regard a taxonomy of random forests is presented in Section 6.1. Random forests that introduce different sources of randomisation is the topic of Section 6.2, with deterministic modifications found in the literature discussed in Section 6.3. Furthermore, some interesting and unique algorithms related to random forests are described in Section 6.4, with a visual perspective on the various approaches presented in Section 6.5. Towards an evaluation of the random forest algorithms found in the literature, a bias-variance analysis on a selection of algorithms is discussed in Section 6.6. The analysis leads to a novel random forest framework, which is proposed in Section 6.7. This is followed by concluding remarks in Section 6.8.

6.1 Introduction

Within Breiman's definition of a random forest, several different approaches exist (Breiman, 2001*a*). The most apparent distinction between the different approaches is the way in which the independent identically distributed random vectors $\{\Theta_b\}$ are obtained. Implicitly, this represents the construction of each tree or its source of randomisation. Furthermore, in addition to the nature of $\{\Theta_b\}$, different random forest algorithms may be obtained by either preprocessing sampled training data, optimising the ensemble size of the forest and/or changing the ensemble voting scheme. In a similar, but extended fashion to Tripoliti *et al.* (2013), random forests can be categorised by their possession of a unique combination of traits from among those presented in Figure 6.1. In other words, a random forest can be specified by choosing its sources of randomness (Category R) as well as by deciding on some deterministic modifications pertaining to preprocessing, tree construction, ensemble combination

and/or "smoothing"[1] (Categories A to D).

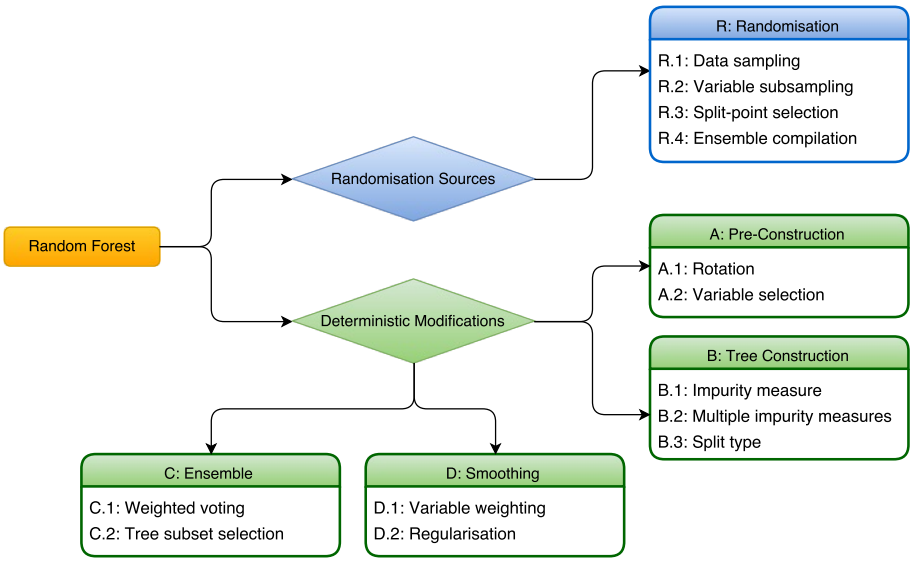

Figure 6.1: Properties of a random forest.

For example, to arrive at Breiman's Forest-RI, the sources of randomisation are bootstrap sampling and variable subsampling, whereas deterministic modifications (usually) entail using the Gini index as node impurity measure and the use of orthogonal splits. The above categorisation does not discriminate between all random forests: two approaches may share the same sources of randomness and deterministic modifications (for example using weighted voting), but differ one level deeper (such as having *different* weighting strategies in the voting step). An attempt was made to distinguish between algorithms at an additional level. Such a taxonomy however no longer succeeded in clarifying similarities and differences amongst algorithms. Hence the categorisation as presented in Figure 6.1 is kept to provide at least some idea of where a specific random forest lies within the cosmos that is Breiman's definition.

The following sections contain a brief overview of the literature concerning proposals of different random forests. Proposals will be presented based on the categories discussed in Section 6.1. Since typically the design of a random forest stretches over different categories, initial introductions will be confined to the category having the most discriminatory power between the algorithm and other random forests.

[1]Smoothing is discussed in more detail in Section 6.3.4.

6.2 Randomisation Sources

As discussed in Section 4.2, among the first approaches to inject randomness into the construction of an ensemble of trees was proposed by Kwok and Carter (1990) and Dietterich (1998). The idea was to randomly select splits from a ranked list (R.3). In Kwok and Carter (1990), the list includes only the top three splits, whereas in Dietterich (1998) the top twenty splits are used. Ho (1995) produced randomised trees by selecting only a subset of the available input variables before constructing each tree (R.2). Shortly after, Breiman (1996*a*) proposed bagging, which uses any base learner and which randomises via bootstrap sampling (R.1). Note that bagging defines a larger class of algorithms, where in a sense many random forests using bootstrap sampling can simply be redefined as bagging using randomised trees as base learners. Conversely, according to Breiman's definition, bagging any type of tree learner is a random forest.

Probably the most widely used strategy to inject randomness into a tree ensemble is to select a subset of inputs at *each* node when computing an optimal split. This was first explored by Amit and Geman (1997) on a character recognition problem, and subsequently by Ho (1998). Breiman (2001*a*) was the first to essentially "bag" this type of randomised tree — which became the popular Forest-RI algorithm (R.1 and R.2). Since then, many proposals have settled on this configuration of randomisation (bootstrap sampling combined with variable subsampling) and have subsequently focused on improving the algorithm by way of deterministic modifications.

In Cutler and Zhao (2001), the authors introduced the idea of creating a so-called *perfect random tree ensemble* (PERT). At each step, using a bootstrap sample and starting at the root node, the procedure selects two observations at random. If these are from different classes, the node is split on a randomly chosen input variable using a linear combination of the two points, where the respective weights are $\alpha \sim U[0, 1]$, and $1 - \alpha$. The procedure is continued in a recursive manner until within a node, two observations belonging to different classes cannot be found within ten tries. The node is then declared terminal. The above procedure is unique in the sense that there are three sources of randomisation, *viz.* data sampling, variable subsampling and random split-points (R.1, R.2 and R.3). In a similar fashion, but without the use of data sampling, Geurts *et al.* (2006) proposed *extremely randomised trees*, where both a subset of input variables and a split-point are selected randomly at each node (R.2 and R.3). The difference is that extremely randomised trees allow more than a single variable to be randomly selected. Among these, the best variable and split-point pair is obtained using a node impurity measure (even if split-points are chosen at random). Restricting the variable subset size to one in extremely randomised trees is almost identical to PERT.

Using Minimum Message Length (MML) to perform oblique splits, Tan and Dowe (2006) grow a large tree and from it create subtrees using randomly chosen branches (R.4). The subtrees constitute the ensemble, where the size of the ensemble is specified a priori. Bader-El-Den and Gaber (2012) build a Forest-RI, treating the ensemble as the population from which trees are randomly sampled. Using a genetic algorithm with accuracy as fitness function, a new ensemble "evolves" from the original sample of randomised trees. The above approach is called Genetic Algorithm based Random Forests (GARF).

Introducing a form of meta-randomisation, Bernard *et al.* (2009) proposed the *Forest-RK*, which at each node not only selects a random subset of inputs (R.2), but also randomly chooses the size of this subset (one level deeper).

In summary, randomisation in a random forest can stem from either data sampling (R.1), variable subsampling (R.2), split-point selection (R.3) and/or ensemble compilation (R.4). However, as shown with Forest-RK, additional randomisation strategies may be added to the aforementioned list. In a rather ad hoc way, it is a simple task to conceive of many other sources of randomisation in a random forest. Among many, these could include:

- Randomising node impurity measures at each node (or for each tree);
- Randomising between orthogonal and oblique splits at each node (or for each tree);
- Randomising between different types of multivariate models for oblique splitting at each node (or for each tree);
- Randomising between methods used for rotating the input matrix for each tree;
- Or any combination of the above mentioned approaches with those already mentioned.

The question is whether any of the proposed additions are sensible. From a variance reduction stand-point, it can be argued that the increase in randomisation will further reduce the correlation between the trees, in turn further reducing the variance of the ensemble. However, it remains unclear what the effect will be on the bias of the ensemble. Intuitively, it might be reasonable to suggest that by randomly combining approaches one can average out particular weaknesses related to certain randomisation strategies, but the same would probably hold true for their strengths.

6.3 Deterministic Modifications

As previously mentioned, many proposals in the literature are unique — not in terms of their design for injecting randomisation into the ensemble creation procedure, but instead by the way they deterministically modify some aspect of the algorithm. These modifications can affect different stages of the ensemble, *viz.* (A) pre-construction (before trees are fit to the data), (B) tree construction (modifying the tree fit to the data), (C) ensemble creation (affecting the compilation of the ensemble) and (D) smoothing. In this context, smoothing refers to the act of influencing the randomisation such that the sampling probabilities are "smoothed" into a shape different from uniformity. For example, instead of variable subsampling with equal probability, sampling is done using a pre-specified probability distribution.

6.3.1 Category A: Pre-construction

Rodriguez *et al.* (2006) build a *rotation forest* where each tree is presented with a rotated version of the training data (A.1). The first step in the procedure is to create S disjoint subsets of the inputs. Next, for each subset $s = 1, ..., S$, only the observations belonging to a random subset of classes are selected. Therefore, the original training data matrix is subsampled both in terms of the input variables (p) and in terms of the observations (N). Subsequently, a bootstrap sample of 75% of the observations is drawn and principal component analysis (PCA) is applied to this submatrix. The PCA coefficients for each input variable subset is then placed inside a "rotation" matrix,

$$R = \begin{pmatrix} a_{11}, a_{12}, \cdots, a_{1M_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & a_{21}, a_{22}, \cdots, a_{2M_2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & a_{N1}, a_{N2}, \cdots, a_{NM_S} \end{pmatrix}, \tag{6.3.1}$$

where $M_1, ..., M_S$ denote the respective sizes of the subsets. The matrix R^a is formed by rearranging the columns of R such that they correspond to the order of the original training set, which is then rotated using R^a. Finally, to build a rotation forest of size B, each tree t_b is trained using a rotated version of the training set obtained using the above steps. Rodriguez *et al.* (2006) use ordinary classification trees as base learners, although the algorithm does not specifically require classification trees. Therefore technically, it can be extended to form a rotation ensemble framework able to be implemented using many different base learners. Related to rotation forests, Zhang and Suganthan (2014) combine several rotated versions of the inputs into an augmented

set. Each rotation is presented to the root node, and the rotation corresponding to the best split is selected. This rotation is then used at all other node splits in the tree. Blaser and Fryzlewicz (2015) generalise the concept of ensembles constructed from base learners fit on rotated versions of the training data using QR decomposition.

Another preprocessed random forest is the one proposed by Genuer *et al.* (2010) which involves two stages. In the first stage a Forest-RI is fit and is used to rank variables based on the Forest-RI variable importance measures. In the second stage, a sequence of Forest-RI models are constructed by starting with the most important variable and only adding an additional variable to the final ensemble if the gain in accuracy exceeds a specified threshold. The purpose is to select the optimal set of variables from the sequence corresponding to the most accurate random forest (A.2).

6.3.2 Category B: Tree Construction

In an attempt to improve the Forest-RI algorithm, Robnik-Šikonja (2004) investigated the use of multiple impurity measures for splitting. By using a different impurity for every fifth of a constructed tree, the aim was to increase the diversity between trees in the ensemble in order to improve the final classifier (B.2).

Das *et al.* (2009) trained a conditional inference forest to identify the factors related to the severity of automobile accidents. Their method manages to supplant an impurity measure approach for determining the best split at each node of the tree (B.1), by using appropriate test statistics instead. For more detail, the reader is referred to Hothorn *et al.* (2006) in which a conditional inference framework for unbiased binary partitioning was developed.

Both Ho (1995, 1998) and Breiman (2001*a*) included versions of their random forests using oblique (non-orthogonal) splitting rules (B.3). Lemmond *et al.* (2008) proposed a *discriminant random forest* which uses linear discriminant analysis (LDA) to perform splits. In line with the above, Menze *et al.* (2011) compare the use of LDA with other models such as ridge regression and random linear combinations of the inputs for node splitting.

6.3.3 Category C: Ensemble Creation

Latinne *et al.* (2001) investigate selecting the optimal number of trees in a random forest model during the construction of the ensemble (C.2). This is done by training (say) 50 trees and adding 10 more, then using the *McNemar test* of significance to test if the accuracy is improved by using 60 trees instead of the initial 50 trees. If the accuracy is not significantly improved, the optimal

ensemble size is 50. Otherwise the procedure is repeated, adding another 10 trees to the ensemble. In a similar way, Bernard *et al.* (2009) explore using sequential forward or backward selection to find an optimal size for a random forest ensemble. Fawagreh *et al.* (2015) use clustering to identify groups of trees that are similar to each other and "prune" a random forest by removing redundant trees. This approach differs from the previous two procedures in the sense that the entire forest has to be constructed before the procedure can be implemented.

In addition to using multiple node impurity measures, Robnik-Šikonja (2004) also explored the use of weighted voting (C.1). At test time, the idea is to compute similarities between training observations and each test observation by means of the random forest proximity measure (as discussed in Section 4.8). Using only the training observations similar to the current test point, the margin as given in (4.3.1) is computed for trees where the training points are out-of-bag. Trees with negative margins are discarded and the weighted vote is taken as the votes of the remaining trees for the current test point, weighted by their average margin on the out-of-bag training points. Related to this approach is *dynamic integration* introduced by Tsymbal *et al.* (2006). Dynamic integration starts by computing similarities between training observations using a technique such as k-nearest neighbours or random forest proximities. Subsequently, "local" error rates for each neighbourhood consisting of similar training observations are obtained for each tree. At test time, dynamic integration allows for three different strategies: *dynamic selection* (DS), *dynamic voting* (DV) and *dynamic selection and voting* (DSV). For a test point DS only selects the tree with the smallest error on similar training observations; DV uses weighted voting where the weights are proportional to the error on each tree for similar training observations; and DSV selects trees with error rates below a specified threshold and then performs weighted voting. Bostrom (2007) compares different approaches to estimating class probabilities and their associated accuracies when used for prediction. These strategies include *average vote*, *relative class frequency*, *Laplace estimate* and an approach called the *m-estimate*. Each estimate is essentially a weighted voting strategy using probabilities instead of class labels.

6.3.4 Category D: Smoothing

Since the advent of Breiman's Forest-RI its use has gained popularity in many domains. This is especially the case in high-dimensional settings such as in the genomic sciences (Lee *et al.*, 2005; Díaz-Uriarte and De Andres, 2006; Caruana *et al.*, 2008; Boulesteix *et al.*, 2012; Klassen *et al.*, 2008). The performance of Forest-RI is however far from optimal in high-dimensional data scenarios. The detrimental effect of a large number of noise variables is illustrated in the following simulation setup. Let $X_1, ..., X_p \sim U[0, 1]$ and $C = 1$, if $q + (1 -$

$2q) \cdot I(X_1 + X_2 > 1) > 0.5$, where $q = 0.15$ (the Bayes error rate), otherwise $C = 0$ (Mease and Wyner, 2008). Therefore, the response C only depends on X_1 and X_2 and all the remaining $p - 2$ variables are noise. Figure 6.2 shows the performance of Forest-RI as a function of the number of noise variables in the simulated data set.

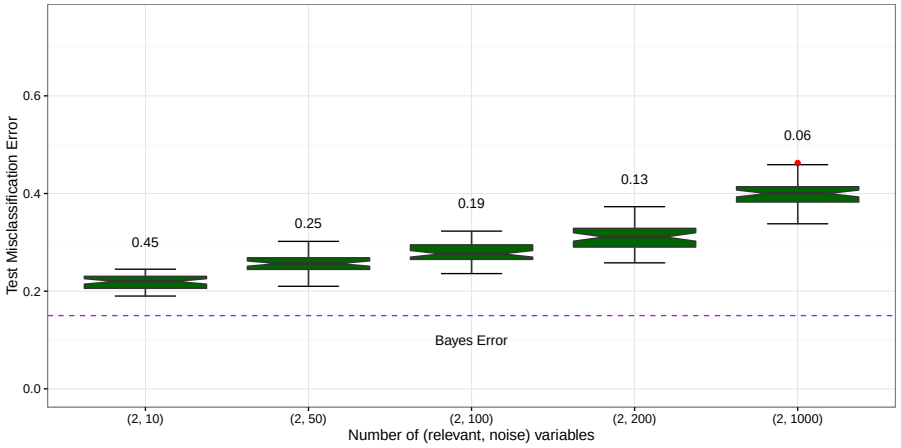

Figure 6.2: Performance of Forest-RI as a function of noise.

As the number of noise variables increases, the performance of Forest-RI deteriorates. The reason for this phenomenon is that computing splits at each node using only a subset of the inputs can lead to splits based solely on noise. In fact, the probability that at least one of the two relevant input variables are selected at a node is depicted at the top of each boxplot in Figure 6.2.[2] When $1,000$ noise variables are present, the probability of selecting at least one of the two relevant inputs is as low as 6%. The idea of addressing the problem illustrated in the aforementioned scenario by deterministically modifying the Forest-RI algorithm is shared by many random forest proposals for high-dimensional data.

Amaratunga *et al.* (2008) propose the *enriched random forest* which uses weighted sampling of the input variables at each node (D.1). Using a two-sample t-test to test for group mean effects between each variable and the response, a p-value can be obtained for each input. However, performing multiple tests means that the p-value vector is suspect if directly used as a weight vector. In addition, in genomic science the number of observations are often very few, putting in question the power associated with each hypothesis test.

[2]This is calculated as $\frac{\binom{2}{1}\binom{p-2}{\zeta-1}+\binom{2}{2}\binom{p-2}{\zeta-2}}{\binom{p}{\zeta}}$, where ζ is the size of the subset of randomly selected variables, chosen to be equal to $\lfloor\sqrt{p}\rfloor$. The function $\lfloor a \rfloor$, takes the floor of the argument a, *i.e.* it removes the decimals from a given number.

Therefore, instead of weighting the inputs using p-values, Amaratunga *et al.* (2008) use q-values, which preserve the relative ordering of the p-values, but adjust for false positives (Storey and Tibshirani, 2003). The q-value associated with the i^{th} smallest p-value (denoted $pv_{(i)}$) is computed as

$$q_{(i)} = min_{k \geq 1}\big[min(pv_{(i)} \cdot (p/k), 1)\big]. \tag{6.3.2}$$

Once the weights are obtained, the Forest-RI algorithm is used with the only modification being that at each node a subset of variables is selected by means of weighted sampling. Similarly, Xu *et al.* (2012) adjust the Forest-RI algorithm for high-dimensional data using a weighted sampling strategy called *weighted subspace random forests*. Here the inputs are discretised (keeping categorical variables as is, and "chopping up" real valued inputs), then for each variable a two-way contingency table is formed with the response. Using these tables, it is possible to compute weights for each variable based on their association with the response using either a χ^2-statistic measure or *information gain ratio*. Ye *et al.* (2013) proposed *stratified random forests* which work by dividing the inputs into two groups, a group consisting of inputs strongly associated with the response and a group of unrelated inputs. In broad terms, any method can be used to compute weights for each variable. For example, Ye *et al.* (2013) use the weights from an LDA projection along the input space direction with the most variation. Once these weights are computed, the set of inputs is split into the two groups using a pre-specified threshold. Thereafter the algorithm is identical to Forest-RI, except that at each node split, variables are sampled in equal proportion from each group using weighted sampling. Nguyen *et al.* (2015) build on this approach with an addition to the algorithm that attempts to make node splits less biased.

Influenced by Friedman and Popescu (2008), Deng and Runger (2012) proposed a *regularised random forest* which attempts to address the issue facing Forest-RI in high-dimensions via a tree regularisation framework (D.2). Suppose $Q(X)$ represents the node impurity measure computed when splitting a node using X and initialise $Q(X) = 0$. Furthermore, consider the empty set of inputs J, from which candidates can be selected for splitting at each node. The basic idea behind the regularisation framework is to populate J until $|J| = \zeta$ by only allowing new input variables into the set if

$$\lambda \cdot Q(X_l) > max_j(Q(X_j)), \tag{6.3.3}$$

where $\lambda \in (0, 1]$, $X_j \in J$ and $X_l \notin J$. Placing a penalty on new inputs entering the set J, the parameter λ acts as a regularisation parameter. Note

that during ensemble construction, regularised random forests keep track of the set of variables J across different trees. Hence the procedure has memory. Strictly speaking, a regularised random forest is therefore not a random forest according to Breiman's definition. Furthermore, in the algorithm if $\lambda \cdot Q(X_w) = \lambda \cdot Q(X_l) > max_j(Q(X_j))$ for $w \neq l$ and $X_w, X_l \notin J$, then one of the two variables is selected at random. This aspect of the algorithm can cause an issue for the framework in the setting for which it was intended. During partitioning of the input space, as nodes contain fewer and fewer observations the possible number of unique impurity measure values computable at each node decreases (Deng and Runger, 2013). This means that in a high-dimensional setting, many inputs will share identical impurities and the selection procedure to populate the set J will end up mimicking random sampling. Deng and Runger (2013) refer to this issue as the *node sparsity issue*. To amend this, Deng and Runger (2013) came up with a strategy using random forest variable importance measures to "guide" the selection procedure. The algorithm is called *guided regularised random forests*. In this approach, for the j^{th} input the regularisation parameter λ in (6.3.3) is replaced with a weighted average

$$\lambda_j = \alpha Vimp_j + (1 - \alpha)\lambda, \tag{6.3.4}$$

where $\alpha \in [0, 1]$ and $Vimp_j$ is the variable importance of the j^{th} variable computed from a Forest-RI. Therefore, X_l will enter the set J if $\lambda_l \cdot Q(X_l) > max_j(Q(X_j))$, $Vimp_l > max_w(Vimp_w)$[3] and $|J| < \zeta$. This means that ties among impurity measures are broken based on pre-calculated importance measures. However, Deng (2013) notes that this may cause the trees in the forest to become highly correlated, thereby potentially reducing the performance of the ensemble due to an increase in variance. For this reason Deng (2013) proposed the *guided random forest* which replaces regularisation with variable importance weighted sampling. Concretely, (6.3.4) is replaced with

$$\lambda_j = 1 - \alpha + \alpha \frac{Vimp_j}{Vimp^*}, \tag{6.3.5}$$

where $Vimp^* = max(Vimp_1, ..., Vimp_p)$ computed from a Forest-RI.

In summary, many of the presented proposals create novel random forest algorithms by deterministically modifying Breiman's Forest-RI. These modifications target different aspects of a random forest, such as pre-construction (A.1, A.2), tree-construction (B.1, B.2, B.3), the ensemble (C.1, C.2) and/or smoothing (D.1, D.2). Similar to sources of randomisation it is easily possible

[3]Here w indexes the set of inputs that is not in J and does not contain X_l.

to conceive of novel random forest algorithms that combine the approaches from the different categories. For example, by:

- Combining rotations with oblique trees;
- Combining trees built with multiple impurity measures with weighted voting;
- Combining dynamic integration with regularisation;
- Or by using a combination of rotation, oblique trees, weighted voting and regularisation.

The above list can be expanded[4], but again as with ad hoc randomisation proposals, the theoretical justifications for these proposed algorithms remain unclear.

6.4 Other Related Approaches

In this section, some of the more exotic proposals related to random forests are briefly discussed. Some of these fit within Breiman's definition of a random forest, but the majority are only related in a superficial sense.

Boinee *et al.* (2008) proposed *meta random forests* which is a strategy based on bagging and boosting using random forests as base learners. In the case of bagging random forests, the only difference between this approach and a very large random forest is that there are two layers of bootstrap sampling, one externally for the bagging procedure and the next within each random forest base learner. The boosting approach follows the AdaBoost algorithm using exponential loss and random forests as base learners. Each approach is clearly computationally much more intensive than the original proposals which use single trees as base learners.

A more sensible approach attempting to combine boosting with random forests is *RotBoost* proposed by Zhang and Zhang (2008). The idea is to combine rotation forests with the AdaBoost algorithm. The modification is fairly simple: before fitting a tree as base learner in AdaBoost, one rotates the data using a rotation matrix which is obtained as proposed by Rodriguez *et al.* (2006). The motivation for the approach is to attempt to simultaneously reduce both the bias (through boosting) and the variance (through random rotations) of

[4]For an empirical investigation of some of these random forest combinations, see Tripoliti *et al.* (2013).

the classifier.

An interesting approach developed by Welbl (2014) recasts random forests as an ensemble of artificial neural networks.[5] The basic idea is that the nodes and branches of a randomised tree can be represented as a neural network with two hidden layers, and that an ensemble can be obtained using many of these tree network representations. With the recent successes and excitement surrounding neural networks and *deep learning* (a form of *representation learning* which uses neural networks with many hidden layers), it might be interesting to further investigate this connection (Goodfellow *et al.*, 2016).

A *disjunction normal random forest* as proposed by Seyedhosseini and Tasdizen (2015) is a particularly unique strategy. As a starting point, consider the tree representation for binary classification given in Figure 6.3. The function $\phi_m(\boldsymbol{x}, \Theta) \in \{0, 1\}$ acts as the derived split rule at the m^{th} node. For an observation $\boldsymbol{x}$, if $\phi_m(\boldsymbol{x}, \Theta) = 0$, the observation moves down the left branch at Node m, otherwise it moves down the right branch. To clarify, suppose the rule $X_2 > a, a \in \mathbb{R}$, applies at Node 3 (say), then $\phi_3(\boldsymbol{x}, \Theta) = I(x_2 > a)$. Seyedhosseini and Tasdizen (2015) show that any such binary tree can be reformulated as a *disjunction* of *conjunctions*.[6]

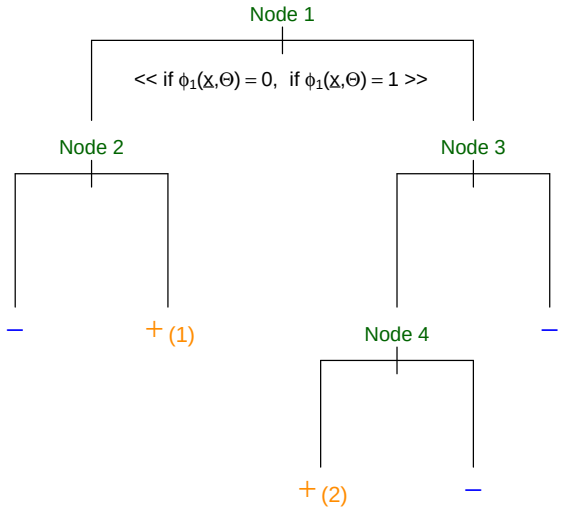

Figure 6.3: Binary tree representation.

[5] For more details regarding neural networks see for example Gurney (1997).

[6] A disjunction is a function consisting of OR operators and is true if either argument is true. A conjunction consists of AND operators and is true only if both arguments are true.

Let $\vee$ denote a disjunction (the OR operator) and $\wedge$ denote a conjunction (the AND operator). Any binary classification tree can then be reformulated as

$$t(\boldsymbol{x}, \Theta) = \bigvee_{i=1}^{M_+} \left[\bigwedge_{m \in RP_i} \phi_m(\boldsymbol{x}, \Theta) \cdot \bigwedge_{m \in LP_i} \neg\phi_m(\boldsymbol{x}, \Theta) \right], \tag{6.4.1}$$

where $\neg\phi_m(\boldsymbol{x}, \Theta) = 1$ if $\phi_m(\boldsymbol{x}, \Theta) = 0$, and vice versa. Here M_+ is the number of terminal nodes in the tree associated with the "+" class and RP_m indexes the internal nodes at which an observation is required to follow the right branch during the journey from the root if it is to reach the i^{th} positive terminal node. Similarly, LP_m indexes the internal nodes at which an observation is required to follow the left branch on its way down to a positive terminal node. For example, in Figure 6.3, $M_+ = 2$, $RP_1 = \{2\}, LP_1 = \{1\}, RP_2 = \{1\}$ and $LP_2 = \{3, 4\}$. In other words, to reach the first positive terminal node $+_{(1)}$, an observation would have to follow the left branch at Node 1 and then follow the right branch at Node 2. To reach $+_{(2)}$, the right branch should be followed at Node 1, and the left branch followed at both Node 3 and Node 4. The tree in Figure 6.3 can now be reformulated as

$$t(\boldsymbol{x}, \Theta) = \big[\neg\phi_1(\boldsymbol{x}, \Theta) \wedge \phi_2(\boldsymbol{x}, \Theta)\big] \vee \big[\phi_1(\boldsymbol{x}, \Theta) \wedge \neg\phi_3(\boldsymbol{x}, \Theta) \wedge \neg\phi_4(\boldsymbol{x}, \Theta)\big], \tag{6.4.2}$$

where $\boldsymbol{x}$ is classified as belonging to class "+" if $t(\boldsymbol{x}, \Theta) = 1$. Otherwise $\boldsymbol{x}$ is classified to the class "−". The form presented in (6.4.2) is also known as the *disjunctive normal form* of a tree, which is from where the algorithm derives its name. Note that since $\phi_m(\boldsymbol{x}, \Theta) \in \{0, 1\}$, the AND operator can be replaced with simple multiplication. In addition, $OR(w, z)$ can be written as $\neg AND(\neg w, \neg z)$ with $w, z \in \{0, 1\}$[7], which in turn is equivalent to $1 - (1 - w) \cdot (1 - z)$. Using these expressions, (6.4.2) becomes

$$t(\boldsymbol{x}, \Theta) = 1 - \prod_{i=1}^{M_+} \left\{ 1 - \prod_{m \in RP_i} \phi_m(\boldsymbol{x}, \Theta) \prod_{m \in LP_i} \big[1 - \phi_m(\boldsymbol{x}, \Theta)\big] \right\} \tag{6.4.3}$$

$$= 1 - \prod_{i=1}^{M_+} (1 - h_i(\boldsymbol{x}, \Theta)). \tag{6.4.4}$$

Furthermore, any rule $X > a$ can be written as $b + X > 0$, where in Seyedhosseini and Tasdizen (2015), $b = -a$ is called the *bias*. Fitting a classification

[7] If $w = 1, z = 0 \Rightarrow \neg[(\neg 1) \wedge (\neg 0)] = \neg(0 \wedge 1) = \neg 0 = 1$. Similarly, with $w = 0, z = 1, \neg[(\neg 0) \wedge (\neg 1)] = 1$. If $w = z = 1 \Rightarrow \neg[(\neg 1) \wedge (\neg 1)] = \neg(0 \wedge 0) = \neg 0 = 1$. Finally, $w = z = 0 \Rightarrow \neg[(\neg 0) \wedge (\neg 0)] = \neg(1 \wedge 1) = \neg 1 = 0$ which is equivalent to the OR operator.

tree is essentially equivalent to learning a set $\Theta = \{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_M\}$, where each $\boldsymbol{\theta}_m = [b_m, X]^T$, $X \in \{X_1, ..., X_p\}$ is a vector consisting of a split variable X and a bias term b_m associated with Node m. The key idea to the approach is to estimate these parameter vectors corresponding to discrete splitting rules derived from a fitted tree using the logistic sigmoid function given in Figure 6.4, and to recast the tree into a continuous function.

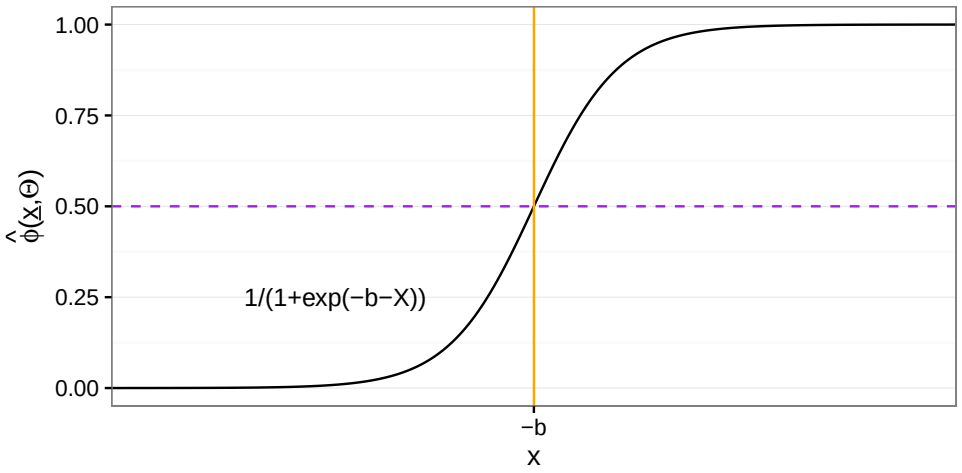

Figure 6.4: Logistic sigmoid function used to approximate a tree node splitting rule.

Concretely, a discrete splitting rule can be approximated using

$$\hat{\phi}(\boldsymbol{x}, \Theta) = \frac{1}{1 + e^{-b-X}}. \tag{6.4.5}$$

As shown in Figure 6.4, $X = -b$ implies that $b + X = 0$ (the split rule decision threshold) and $\hat{\phi}(\boldsymbol{x}, \Theta) = 0.5$, which corresponds to the highest degree of uncertainty regarding the branch an observation should take from the current node. In contrast, if $X \gg -b$ it means that $b + X \gg 0$, which corresponds to $\hat{\phi}(\boldsymbol{x}, \Theta) \approx 1$ and a high degree of certainty that an observation should move from the current node down the right branch.[8] Finally, if $X \ll -b \Rightarrow b + X \ll 0 \Rightarrow \hat{\phi}(\boldsymbol{x}, \Theta) \approx 0$, which corresponds to a high degree of certainty that an observation should follow the left branch down. Once an estimate has been obtained for each node, all of the bias terms can be updated using gradient descent (as discussed in Section 3.2.1).

[8]The notation $a \gg b$, means that a is much larger than b. Similarly, $a \ll b$ indicates that a is much smaller than b.

To obtain the gradient, suppose $\Omega_{tr} = \{(\boldsymbol{x}_i, c_i), i = 1, ..., N\}$, with $c_i \in \{0, 1\}$ and let $\hat{t}(\boldsymbol{x}, \Theta)$ denote (6.4.4) with $\phi(\boldsymbol{x}, \Theta)$ replaced by (6.4.5). Using squared error loss, the negative gradient component corresponding to a bias term at Node m for a training observation $(\boldsymbol{x}, c)$ is given by[9]

$$\begin{aligned} u_m &= -\frac{\partial(c - \hat{t}(\boldsymbol{x}, \Theta))^2}{\partial b_m} \\ &= 2x(c - \hat{t}(\boldsymbol{x}, \Theta)) \\ &\times \Bigg[\sum_{l|m \in RP_l} \Bigg(\prod_{r \neq l} (1 - h_r(\boldsymbol{x}, \Theta)) h_l(\boldsymbol{x}, \Theta)(1 - \hat{\phi}_m(\boldsymbol{x}, \Theta)) \Bigg) \\ &- \sum_{l|m \in LP_l} \Bigg(\prod_{r \neq l} (1 - h_r(\boldsymbol{x}, \Theta)) h_l(\boldsymbol{x}, \Theta) \hat{\phi}_m(\boldsymbol{x}, \Theta) \Bigg) \Bigg], \end{aligned} \tag{6.4.6}$$

where x represents the observed value corresponding to the selected variable on which the m^{th} node is split. The update for b_m with step length ν is then

$$b_m^{new} = b_m^{old} + \nu \cdot u_m. \tag{6.4.7}$$

Seyedhosseini and Tasdizen (2015) motivate their approach by stating that instead of learning each parameter vector separately as is the case with binary trees, the algorithm can learn all the parameter vectors simultaneously using gradient descent. This in turn translates into smoother decision boundaries and better generalisation performance. From this point, it is relatively straightforward to build a disjunctive normal random forest. To create a new ensemble, the above approach is followed for each tree obtained from a random forest such as a Forest-RI. The final classifier may then be obtained as

$$\bar{t}_{DNRF}(\boldsymbol{x}) = I\left[\sum_{b=1}^{B} I(\hat{t}(\boldsymbol{x}, \Theta) > 0.5) > \frac{B}{2} \right].$$

6.5 A Visual Perspective

The previous sections in this chapter briefly outlined some of the different types of random forests and related methods that have been proposed in the literature. However, it still remains difficult to develop a bird's eye view of the jungle that comprises all of these techniques. This section attempts a further step towards such an overall view. The following approach was taken: for each method, the various characteristics with respect to the categories presented

Table 6.1: The variables describing each random forest algorithm.

Category	Variable	Type	Range
	author	categorical	NA
	year	numeric	1988 - 2015
R.1	r_data	numeric	{0, 1}
R.2	r_subsample_var	numeric	{0, 1}
R.3	r_split_points	numeric	{0, 1}
R.4	r_ensemble	numeric	{0, 1}
A.1	a1_rotation	numeric	{0, 1}
A.2	a2_var_select	numeric	{0, 1}
B.1	b1_single_impurity	numeric	{0, 1}
B.2	b2_multiple_impurity	numeric	{0, 1}
B.3	b3a_oblique	numeric	{0, 1}
	b3b_axis	numeric	{0, 1}
C.1	c1_weighted_vote	numeric	{0, 1}
C.2	c2_tree_subset_selection	numeric	{0, 1}
D.1	d1_var_weights	numeric	{0, 1}
D.2	d2a_regularisation	numeric	{0, 1}
	d2b_memory	numeric	{0, 1}

in Section 6.1 were recorded into a data frame. The variables describing each method are presented in Table 6.1.

Using this data frame, a distance matrix was computed containing the distances between each random forest in terms of their characteristics. This matrix represents the full "*trait space*", which can be reduced to a best two-dimensional approximation by way of multidimensional scaling (MDS). Briefly, MDS approximates dissimilarities (in this case Euclidean distances) by preserving as much as possible the original dissimilarity measurements between points in a higher-dimensional space when projected down into a lower-dimensional space. This is done by projecting the points onto the space comprised of the dimensions corresponding to the two largest eigen values obtained from a spectral decomposition of the full doubly centred dissimilarity matrix. The reader may refer to Cox and Cox (2000) for more detail.

Figure 6.5 shows the resulting MDS plot. Each approach is represented by a point labelled according to the author(s) who proposed it. The further two points (random forests) are from each other, the more dissimilar they are in terms of the categories discussed earlier. Every algorithm is represented using a two-way colour code. The colour of the author and year corresponds to a certain combination of randomisation sources, while the colour of the diamond symbol next to each author resembles the algorithm's combination of deterministic modifications. The latter is given in the legend at the top right of Figure 6.5.

[9]The full derivation is given in Appendix A of Seyedhosseini and Tasdizen (2015).

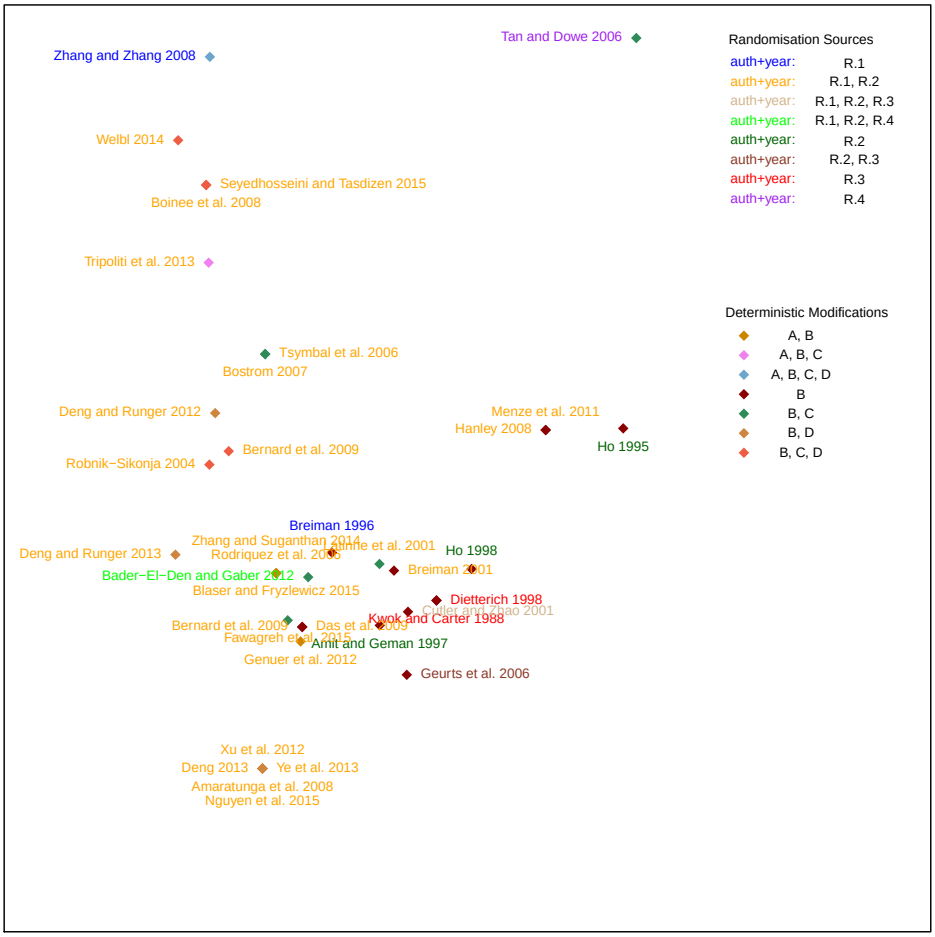

Figure 6.5: Trait based comparison of random forest proposals by way of a best two-dimensional MDS approximation of the full trait space.

Some of the more interesting aspects of the display are briefly discussed below:

- Most of the random forests share the same sources of randomisation, *viz.* data sampling and variable subsampling (R.1 and R.2). This was of course first proposed by Breiman (2001*a*);
- The display seems to present a reasonable picture regarding algorithm dissimilarity: many of the early developments (Kwok and Carter, 1990; Amit and Geman, 1997; Ho, 1998; Dietterich, 1998; Breiman, 2001*a*) are found "close" to each other, whereas more esoteric and scenario specific proposals are spread further out.
- Proposals which are only related to random forests (Boinee *et al.*, 2008; Zhang and Zhang, 2008; Welbl, 2014; Seyedhosseini and Tasdizen, 2015) are clearly distinguished occupying the top left corner of the display.

For example, the RotBoost algorithm that combines data rotation with boosting is situated in a fairly isolated position at the top left of the display. It is also interesting to note that disjunctive normal random forests (Seyedhosseini and Tasdizen, 2015) and meta random forests (Boinee *et al.*, 2008) share the same location.

- In the high-dimensional setting (Amaratunga *et al.*, 2008; Xu *et al.*, 2012; Ye *et al.*, 2013; Deng, 2013; Nguyen *et al.*, 2015), the granularity of the categorisation fails to discriminate between algorithms since all of the approaches share the same sources of randomisation (R.1 and R.2) and deterministic modifications (belonging to categories B and D) and differ only one level deeper (in terms of different strategies for performing weighted variable subsampling).

Zooming in, Figure 6.6 shows the decision boundaries for some of the proposals fit to the mixture data. Moving from left to right, top to bottom, the decision boundaries belong to the following random forests: extremely randomised trees (Geurts *et al.*, 2006), rotation forests (Rodriguez *et al.*, 2006), oblique random forests (Menze *et al.*, 2011), weighted subspace random forests (WSRF) (Xu *et al.*, 2012) and regularised random forests (RRF) (Deng and Runger, 2012) with $\lambda = 0.1$ and $\lambda = 0.6$ respectively.

Aspects of interest are the following:

- The additional randomisation (selecting the split-point at random) in extremely randomised trees produces a very "wiggly" decision boundary. This is presumably because the split direction changes more often than if the split-point was not selected at random;
- In the rotation forest, less randomisation combined with rotation produces a fairly smooth non-axis aligned decision boundary;
- Using logistic regression to perform node splitting, among all of the approaches the oblique random forest visually best approximates the Bayes decision boundary. Moreover, the fit achieves a test error that is on average (excluding regularised random forests with $\lambda = 0.1$) roughly 10% better than all the other approaches on the mixture data;
- Both WSRF and RRF are intended for high-dimensional settings which is clearly not the case for the mixture data only consisting of two input variables. However, at the core of both approaches is the Forest-RI algorithm. Hence it is not surprising that in this lower dimensional setting, the resulting decision boundaries (middle right and bottom right panel of Figure 6.6) closely resemble that of Forest-RI as seen in the left

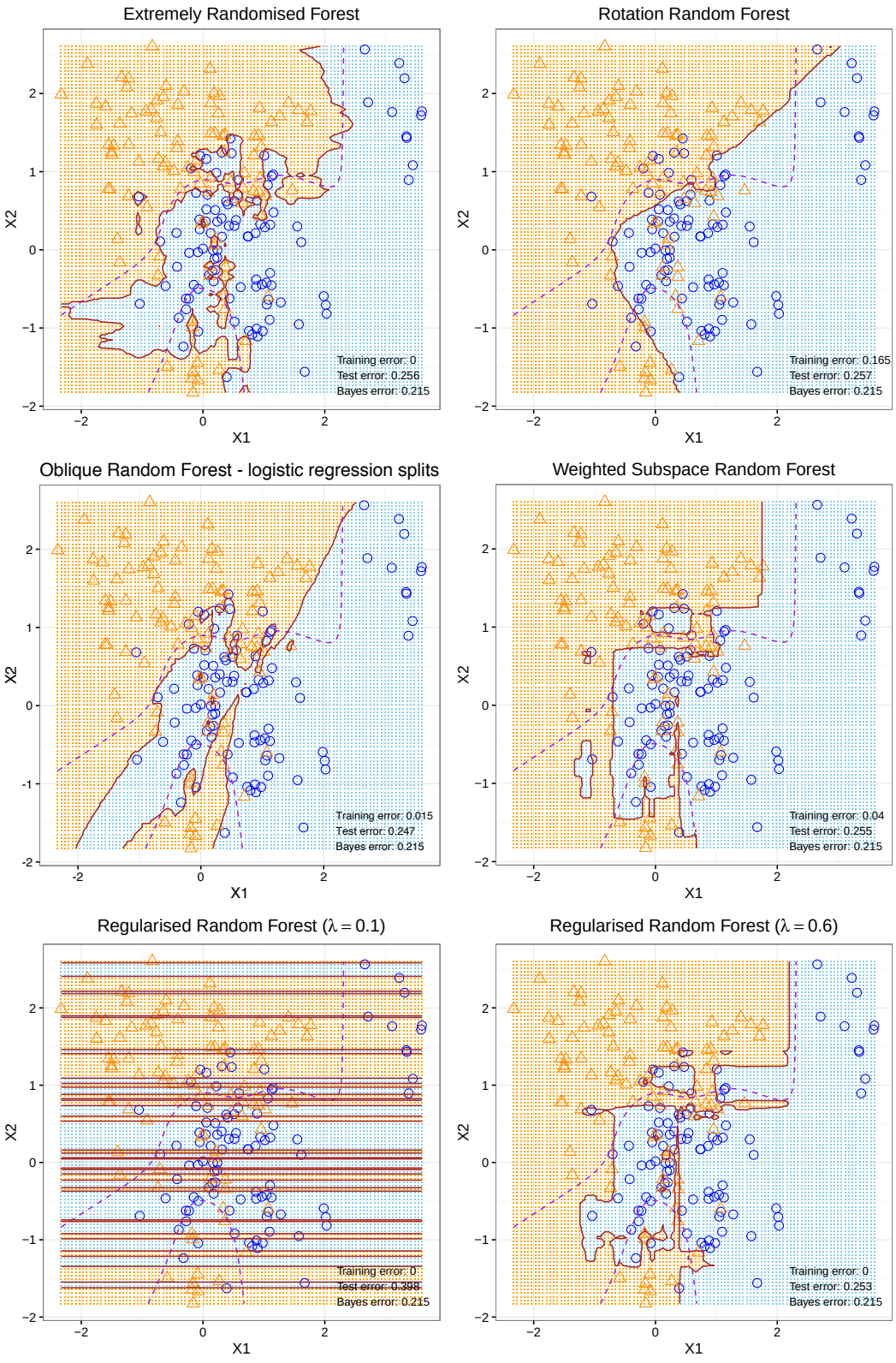

Figure 6.6: Random forest decision boundaries: top left: *extremely randomised forest*; top right: *rotation random forest*; middle left: *oblique random forest with logistic regression splits*; middle right: *weighted subspace random forest*; bottom left: *regularised random forest* ($\lambda = 0.1$); bottom right: *regularised random forest* ($\lambda = 0.6$)

panel of Figure 4.2. The RRF with $\lambda = 0.1$ in the bottom left panel of Figure 6.6 heavily restricts additional input variables to enter the set J. In this case, once X_2 entered the set, X_1 was unable to produce splits that were appreciably better than those of X_2 given the imposed penalty. Therefore, all the subsequent splits were also orthogonal to the X_2 axis. The ideal would be to visualise the exact decision boundary for these algorithms in higher-dimensional settings, but unfortunately this is not feasible.

To better illustrate the effect of a random forest for high-dimensions, Figure 6.7 compares the performance of WSRF with Forest-RI for the same simulated scenarios as were presented in Figure 6.2. Here the WSRF is seen to deal better with the increase in noise due to its weighted variable subsampling strategy which puts higher sampling probabilities on variables more associated with the response.

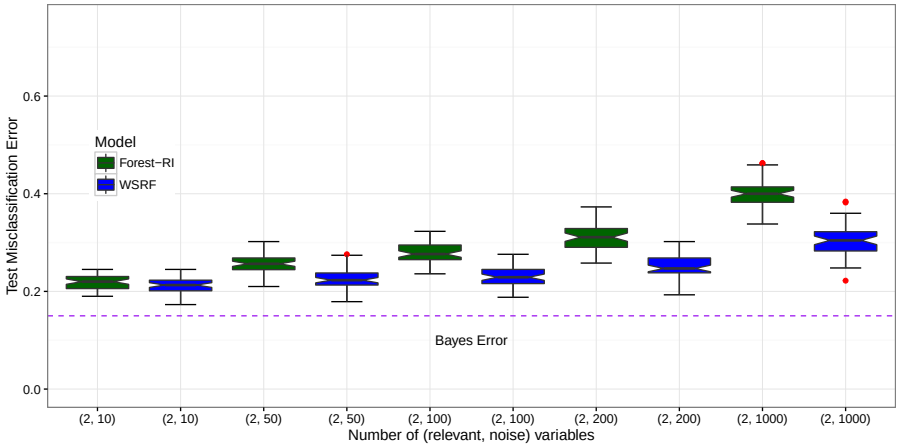

Figure 6.7: Comparing the performance of Forest-RI with WSRF as a function of noise.

6.6 Analysing Bias, Variance and their Effects

Similar to Section 5.6, the purpose of this section is to analyse the bias and the variance along with their respective effects, for a subset of the aforementioned random forest algorithms on simulated data sets. The subset consists of Forest-RI, extremely randomised forests (ERF), rotation forests (RotF) and oblique random forests using logistic splits (ORF-log). Boosting was also included in the analysis. The rationale of this particular selection of random forests is that it represents a broad mixture of categories, *viz.* $(R.1 + R.2)$, $(R.1 + R.2 + R.3)$, $(R.1 + A.1)$ and $(R.1 + R.2 + B.3)$, using only a small

selection of algorithms.

The simulated data in the analysis include: Sim 1 to Sim 4 (including different degrees of correlation between the input variables), Sim 5 to Sim 8 (including different signal to noise ratios) and Sim 9 to Sim 11 (including different data clusterings). The data sets Sim 12 to Sim 16 were omitted since the implementation of oblique random forests in R (found in the *obliqueRF* package), only supports binary classification tasks. Furthermore, the rather artificial scenarios, *viz Circle* and *XOR*, where also not included.

In order to find the optimal tuning parameters for each algorithm among a pre-specified grid of available parameters, ten-fold cross-validation was performed before each fit. The pre-specified grids were chosen as follows:

- **Forest-RI and ORF-log**: The number of trees were taken as $B = 200$, with the subset size of randomly selected variables selected from $\xi = \{1, \lfloor\sqrt{p}\rfloor, \lfloor p/2\rfloor\}$.
- **ERF**: The number of trees were taken as $B = 200$, with the subset size of randomly selected variables also selected from $\xi = \{1, \lfloor\sqrt{p}\rfloor, \lfloor p/2\rfloor\}$. Furthermore, the number of randomly selected split points were selected from $s = \{1, 5, 10, \lfloor N/2\rfloor\}$.
- **RotF**: The number of trees were taken as $B = 200$, with the number of variable subsets selected from $K = \{\lfloor p/2\rfloor, \lfloor p/3\rfloor, \lfloor p/4\rfloor\}$.
- **Boosting**: The number of trees were $B = 200$, tree interaction depth was either one or six, and the step-length factor $\nu = \{0.01, 0.05, 0.1\}$. The exponential loss was used.

The above specifications of possible parameter values differ in some respects to those in Section 5.6. The grid of subset sizes (ξ) was reduced due to the extensive training time required by oblique random forests. The new grid is motivated as follows: a subset size equal to one was seen to be a popular selection in Figure 5.9 for Sim 1 to Sim 4. Furthermore, based on suggestions from Breiman (2001*a*), both Menze *et al.* (2011) and Geurts *et al.* (2006) use $\xi = \lfloor\sqrt{p}\rfloor$. Finally, as was observed in Sim 5 to Sim 8 in Figure 5.9, when dealing with noisy data, larger subsets are often required. Therefore, $\xi = \lfloor p/2\rfloor$ was also included into grid. With regard to split points, Geurts *et al.* (2006) fixed s to be equal to one. Since values of s closer to N will result in ERF being very similar to Forest-RI, additional small values were chosen such as five and ten, with exception to $\lfloor N/2\rfloor$. In RotF, Rodriguez *et al.* (2006) fix the value for K at $\lfloor p/3\rfloor$ in their experiments. The values $\lfloor p/2\rfloor$ and $\lfloor p/4\rfloor$ were

added to the grid, thereby expanding the range of K. The values for B and ν, along with the interaction depth in boosting were identical to those used in Section 5.6.

Table 6.2: Estimated bias, variance, systematic and variance effects for random forest algorithms. Values in bold indicate row-wise minima.

Name	Data	Quantity	Forest-RI	ERF	RotF	ORF-log	Boosting
Sim 1	**mvnorm** $p = 15$, $\rho = 0.9$	Error	0.036	0.036	**0.034**	0.035	0.043
		Bayes Error	0.028	0.028	0.028	0.028	0.028
		Systematic Effect	0.003	0.001	**0**	0.003	0.005
		Variance Effect	0.005	0.007	0.006	**0.004**	0.010
		Bias	0.005	0.003	**0.002**	0.005	0.007
		Variance	0.016	0.017	0.014	**0.012**	0.026
Sim 2	**mvnorm** $p = 15$, $\rho = 0.5$	Error	0.060	0.058	**0.050**	0.055	0.068
		Bayes Error	0.040	0.040	0.040	0.040	0.040
		Systematic Effect	0.010	0.006	**0.004**	0.009	0.013
		Variance Effect	0.010	0.012	**0.006**	**0.006**	0.015
		Bias	0.024	0.012	**0.006**	0.015	0.033
		Variance	0.032	0.032	**0.019**	0.022	0.043
Sim 3	**mvnorm** $p = 15$, $\rho = 0.1$	Error	0.126	0.120	0.109	**0.107**	0.130
		Bayes Error	0.078	0.078	0.078	0.078	0.078
		Systematic Effect	0.021	**0.009**	0.011	0.014	0.026
		Variance Effect	0.027	0.033	0.020	**0.015**	0.026
		Bias	0.029	**0.015**	**0.015**	0.024	0.040
		Variance	0.080	0.076	0.057	**0.050**	0.084
Sim 4	**mvnorm** $p = 15$, $\rho = 0$	Error	0.214	0.209	**0.167**	0.176	0.200
		Bayes Error	0.141	0.141	0.141	0.141	0.141
		Systematic Effect	0.004	**0**	**0**	0.009	0.004
		Variance Effect	0.069	0.068	0.028	**0.026**	0.055
		Bias	0.044	**0.026**	**0.026**	0.053	0.060
		Variance	0.159	0.159	**0.094**	0.100	0.136
Sim 5	**Mease (2008)** $p = 30$, $J = 2$	Error	0.213	0.201	**0.197**	0.245	0.212
		Bayes Error	0.147	0.147	0.147	0.147	0.147
		Systematic Effect	**0.006**	0.009	0.008	0.041	0.017
		Variance Effect	0.060	0.045	**0.042**	0.057	0.048
		Bias	**0.006**	0.009	0.008	0.065	0.021
		Variance	0.095	0.076	**0.071**	0.130	0.092
Sim 6	**Mease (2008)** $p = 30$, $J = 5$	Error	0.272	0.264	**0.244**	0.272	0.259
		Bayes Error	0.143	0.143	0.143	0.143	0.143
		Systematic Effect	0.015	0.017	**0.008**	0.082	0.021
		Variance Effect	0.114	0.104	0.093	**0.047**	0.095
		Bias	0.029	0.029	**0.020**	0.110	0.037
		Variance	0.181	0.170	**0.141**	0.156	0.159
Sim 7	**Mease (2008)** $p = 30$, $J = 15$	Error	0.302	0.301	**0.260**	0.262	0.284
		Bayes Error	0.136	0.136	0.136	0.136	0.136
		Systematic Effect	0.031	0.024	**0.014**	0.057	0.020
		Variance Effect	0.135	0.141	0.110	**0.069**	0.128
		Bias	0.037	0.040	**0.022**	0.083	0.028
		Variance	0.226	0.225	0.170	**0.158**	0.201
Sim 8	**Mease (2008)** $p = 30$, $J = 20$	Error	0.310	0.306	**0.266**	0.273	0.290
		Bayes Error	0.134	0.134	0.134	0.134	0.134
		Systematic Effect	0.033	0.035	**0.020**	0.083	0.023
		Variance Effect	0.143	0.137	0.112	**0.056**	0.133
		Bias	0.049	0.047	**0.028**	0.109	0.031
		Variance	0.240	0.235	0.180	**0.168**	0.212

Table 6.3: Estimated bias, variance, systematic and variance effects for random forest algorithms. Values in bold indicate row-wise minima.

Name	Data	Quantity	Forest-RI	ERF	RotF	ORF-log	Boosting
Sim 9	**Two -norm** $p = 20$, $K = 2$	Error	0.032	0.030	**0.029**	**0.029**	0.040
		Bayes Error	0.024	0.024	0.024	0.024	0.024
		Systematic Effect	**0.001**	0.003	**0.001**	0.003	0.004
		Variance Effect	0.007	0.003	0.004	**0.002**	0.012
		Bias	0.013	**0.007**	**0.007**	0.011	0.014
		Variance	0.016	0.016	0.015	**0.011**	0.025
Sim 10	**Three -norm** $p = 20$, $K = 2$	Error	0.156	0.146	**0.145**	0.154	0.167
		Bayes Error	0.085	0.085	0.085	0.085	0.085
		Systematic Effect	0.041	**0.036**	0.040	0.055	0.048
		Variance Effect	0.030	0.025	0.020	**0.014**	0.034
		Bias	0.079	**0.070**	0.078	0.091	0.088
		Variance	0.090	0.084	0.068	**0.062**	0.102
Sim 11	**Ring -norm** $p = 20$, $K = 2$	Error	0.041	**0.034**	0.059	0.051	0.051
		Bayes Error	0.018	0.018	0.018	0.018	0.018
		Systematic Effect	**0.008**	**0.008**	0.012	0.017	0.019
		Variance Effect	0.015	**0.008**	0.029	0.016	0.014
		Bias	0.022	**0.018**	0.026	0.029	0.029
		Variance	0.029	**0.021**	0.044	0.032	0.034

The results from Sim 1 to Sim 8 are reported in Table 6.2, whereas the results for Sim 9 to Sim 11 are provided in Table 6.3. The two main findings in the analysis are the following:

- *Rotation forests seem to be the most effective at reducing the bias and the systematic effect, while maintaining a competitive level of variance and variance effect.* This is an interesting result given that the decision boundary for rotation forests on the mixture data (depicted in the top right panel of Figure 6.6), was relatively smooth. This is typically associated with higher bias and lower variance. Presumably, the larger input spaces (consisting of 15, 20 or 30 inputs) in Sim 1 to Sim 11 allowed improved data rotations that were able to closely fit each individual training set without paying too high a price in terms of an increase in variance.

- *Oblique random forests using logistic splits seem to be the most effective at reducing the variance and the variance effect, but at a cost to bias and the systematic effect (especially in noisy settings).* A possible explanation for this behaviour is that by enlarging the set of possible split directions at each node[10], a more diverse set of trees can be constructed, leading to a reduction in variance and the variance effect. However, oblique trees also tend to be more shallow than trees using orthogonal splits (Menze

[10]Since any multivariate model can still perform an orthogonal split at a node, the set of all possible splits for an oblique random forests is larger than that of a random forest using ordinary classification trees.

et al., 2011). In the presence of noise, these shallow trees could possibly incur an increase in bias which affects the overall bias of the ensemble.

Wins/ties are presented in Table 6.4, along with the corresponding p-values obtained from statistical comparisons. Boosting was omitted from the comparisons since it did not achieve a single "win" for any quantity in any of the data sets. To test the hypothesis that there is no difference between all of the algorithms, the Friedman aligned ranks test was used.

Table 6.4: Win/Tie analysis of bias, variance, systematic and variance effects for random forests. An asterisk indicates a significant p-value with $\alpha = 0.05$. Algorithm(s) in parentheses are not included in statistical comparison tests.

Quantity	**(Boosting)**	**Forest-RI**	**ERF**	**RotF**	**ORF-log**	**p-val**
Error	0/0	0/0	1/0	**8/1**	1/1	**0.0213***
Systematic Effect	0/0	0/0	2/2	**7/0**	0/0	**0***
Variance Effect	0/0	0/0	1/0	1/1	**8/1**	**0.0046***
Bias	0/0	1/0	2/3	**4/3**	0/0	**0.0001***
Variance	0/0	0/0	1/0	4/0	**6/0**	**0.0014***
Total	0/0	2/2	7/5	**25/4**	15/2	

Given that the aforementioned hypothesis was rejected, pairwise comparisons were conducted using the Shaffer static test. These are presented in Table 6.5 in the following order: (**1**) Forest-RI vs. ERF; (**2**) Forest-RI vs. RotF; (**3**) Forest-RI vs. ORF-log; (**4**) ERF vs. RotF; (**5**) ERF vs. ORF-log; (**6**) RotF vs. ORF-log.

Table 6.5: Adjusted p-values from the Shaffer static post-hoc test used for pairwise comparisons. An asterisk indicates a significant p-value with $\alpha = 0.05$.

Quantity	**1**	**2**	**3**	**4**	**5**	**6**
Error	0.08	**0***	0.12	0.17	0.87	0.17
Systematic Effect	0.48	0.17	0.14	0.47	**0.02***	**0***
Variance Effect	0.81	0.17	**0.02***	0.81	0.08	0.81
Bias	0.25	**0.04***	0.36	0.48	**0.01***	**0***
Variance	0.69	**0***	**0***	0.08	**0.03***	0.75

From Table 6.5 the following conclusion are drawn. Pairwise, RotF significantly outperformed Forest-RI in terms of error rate, bias and variance. However, the systematic and variance effects did not differ significantly between the two algorithms. Furthermore, it is interesting to note that the bias and the systematic effect of both ERF and RotF was significantly lower than those of ORF-log. While ORF-log outperformed ERF in terms of variance and the variance effect, the difference between ORF-log and RotF was not statistically

significant. This could explain why ultimately, as measured by tallied wins, RotF was able to succeed in more simulation configurations than ORF-log.

6.7 A Novel Framework: Oblique Random Rotation Forests

The bias-variance analysis presented in Section 5.6 as well as in Section 6.6, could serve as a way by which sensible (as opposed to ad hoc) proposals for novel random forest algorithms could be made. Based on observations made in the previous section, the following random rotation ensemble framework is proposed: *rotation forests using randomised oblique trees as base learners*. The rationale is simple: by combining rotations with oblique splitting, the hope is that the bias and variance, and their respective effects will simultaneously be reduced. In this section an empirical investigation of the above proposal is discussed. Specifically, the focus of the analysis was to estimate the bias, variance and their respective effects for oblique random rotation forests using logistic regression (ORRotF-log) for splitting at each node. The pre-specified parameter grid was taken as a combination of the grids for RotF and ORF-log, *viz.* $B = 200$, $\xi = \{1, \lfloor\sqrt{p}\rfloor, \lfloor p/2\rfloor\}$ and $K = \{\lfloor p/2\rfloor, \lfloor p/3\rfloor, \lfloor p/4\rfloor\}$. For detailed results, the reader is referred to Appendix A.

Table 6.6 provides a win/tie analysis of the results. The Friedman aligned ranks test was used to test the hypothesis that there is no difference between all the algorithms. The p-values from this test are reported in the final column. The test of no difference between the algorithms was rejected in the case of all of the measured quantities.

Table 6.6: Win/Tie analysis of bias, variance, systematic and variance effects for random forests, including random rotation forests. An asterisk indicates a significant p-value with $\alpha = 0.05$.

Quantity	Forest-RI	ERF	RotF	ORF-log	ORRotF-log	p-val
Error	0/0	1/0	**7/2**	0/1	1/2	**0.0304***
Systematic Effect	1/2	2/2	**5/2**	0/0	0/1	**0***
Variance Effect	0/0	1/0	1/0	2/1	**6/1**	**0.0002***
Bias	1/0	2/3	**5/3**	0/0	0/0	**0***
Variance	0/0	1/0	2/0	0/0	**8/0**	**0.0017***
Total	2/2	7/5	**20/7**	2/2	15/4	

Regarding pairwise comparisons, only ORRotF-log was compared with the remaining algorithms using the *Finner test* (see Section 7.2 for more details). The comparisons in Table 6.7 are therefore reported in the following order: (**1**) Forest-RI vs. ORRotF-log; (**2**) ERF vs. ORRotF-log; (**3**) RotF vs. ORRotF-log; (**4**) ORF-log vs. ORRotF-log.

Table 6.7: Adjusted p-values from the Finner test for comparing a control. An asterisk indicates a significant p-value with $\alpha = 0.05$.

Quantity	**1**	**2**	**3**	**4**
Error	0.17	0.42	0.29	0.76
Systematic Effect	**0.01***	**0***	**0***	0.71
Variance Effect	**0***	**0***	**0.02***	0.52
Bias	**0.02***	**0***	**0***	0.83
Variance	**0***	**0.01***	0.84	0.74

From Table 6.7 it is seen that in terms of bias and the systematic effect, ORRotF-log was significantly outperformed by Forest-RI, ERF and RotF. On the other hand, ORRotF-log outperformed Forest-RI, ERF and RotF in terms of the variance effect, and Forest-RI and ERF in terms of variance. Furthermore, no significant difference was detected between ORF-log and ORRotF-log. This suggests that the addition of oblique splitting rules to rotation forests resulted in the variance and the variance effect reduction observed in ORF-log to dominate the mechanisms decreasing the bias in RotF. Unfortunately, the overall result is not favourable: the combination of the two approaches failed to materialise into a complementary reduction in both bias and variance, and their respective effects.

6.8 Concluding Remarks

The design of a random forest consists of two main components, *viz.* sources of randomisation and deterministic modifications. The former can stem from data sampling, variable subsampling, random split-point selection and/or non-deterministic ensemble compilation. The latter focuses on some aspect of the algorithm such as preprocessing of the data, tree construction, ensemble creation and "smoothing".

Since the early developments of random forests, many proposals followed, attempting to improve performance by affecting one or both of the two components. Most of these novel additions can be applied in any classification setting. In addition, many proposals have focused exclusively on the high-dimensional setting. Furthermore, algorithms have been developed that closely resemble a random forest, but which are only superficially related.

Analysing the bias, variance and their respective effects for a subset of random forests revealed rotations to be effective at reducing the bias and the systematic effect, and oblique splitting rules to be effective at reducing the variance

and variance effect. Using these insights, a novel oblique random rotation forest framework which combines the two approaches, served as an example of a heuristically developed proposal, rather than a simple ad hoc combination of previously explored mechanisms. Although not entirely satisfactory, oblique random rotation forests using logistic regression splits represent only a special case of the framework. Other splits could also be investigated — possibly yielding better results.

Ultimately however, the main interest in classification algorithms lie with their performance on real world problems. In the next chapter, random forest algorithms are compared on real world benchmark data sets by means of a meta-analysis. In this regard, the road map forward is presented in Figure 6.8.

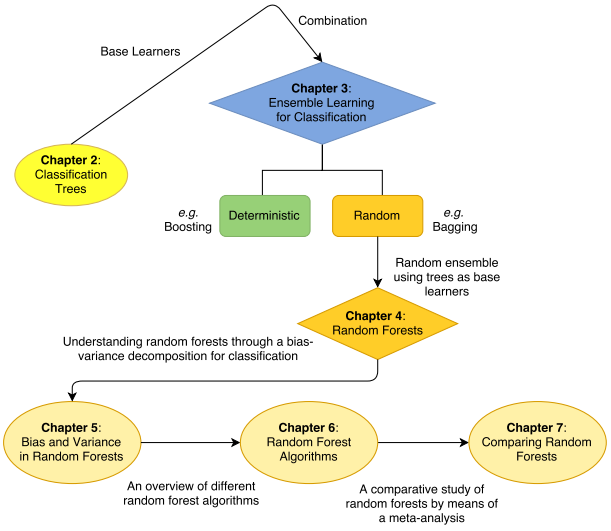

Figure 6.8: Road map to Chapter 7: A comparative study of random forest algorithms by means of a meta-analysis.

Chapter 7

Comparing Random Forests

What is of prime interest concerning the many random forest proposals found in the literature is how the different algorithms compare in terms of their classification performance on real world problems. To be able to compare the different proposals, a meta-analysis is conducted where the reported results from many papers are collected and analysed. The rationale for this approach is discussed in Section 7.1. However, to conduct a proper comparison, statistical hypothesis tests specific to the scenario of comparing multiple algorithms over multiple data sets have to be used. This is introduced in Section 7.2. An evaluation of some aspects of current random forests research is given in Section 7.3. In Section 7.4 a comparison of the different proposals is carried out and discussed. This is followed by concluding remarks in Section 7.5.

7.1 Introduction

As discussed in the previous chapter, many different random forest algorithms have been proposed in the literature. A logical extension of the discussion in Chapter 6 is an attempt at answering the question: which algorithm is best for real world applications? However, what is meant by "best" is highly dependent on the context of the problem at hand. In some cases, minimisation of the classification error is what constitutes an appropriate metric for measuring which algorithm is best. In other cases, a mixture of accuracy and interpretability may be viewed a better choice. Since the latter case is more susceptible to subjective opinion, the focus in this text will be on performance metrics solely associated with a quantitative measurement, removed from qualitative interpretations. This by no means implies that the author is of the opinion that the interpretability of an applied approach is not important. The ability of a classification algorithm to expose meaningful relationships between the response and inputs is often a crucial property. It however remains difficult to measure

differences in the level of interpretability provided by different classification algorithms objectively. Therefore this aspect is not further explored here.

One of the problems with a comparison between the different random forests presented earlier, is that access is required to the software as well as to the programmed algorithms used in the literature. Unfortunately, to fulfil this requirement turns out to be nearly impossible. Only a few authors (or open-source community contributors) create statistical packages or libraries that allow the general public access to different algorithms after publication. Table 7.1 lists the packages that implement some of the proposals in Chapter 6. These packages are available in the R language, which is probably the most popular statistical programming language at the moment.

Table 7.1: Available software for random forests in the *R* programming language.

Proposal	**R package**
Breiman (1996)	ipred
Breiman (2001)	randomForest
Rodriquez et al. (2006)	caret
Geurts et al. (2006)	extraTrees
Das et al. (2009)	party
Menze et al. (2011)	obliqueRF
Deng and Runger (2012)	RRF
Xu et al. (2012)	wsrf
Deng and Runger (2013)	RRF
Deng (2013)	RRF
Other (27 out of 37) proposals	unavailable

As can be seen in Table 7.1, the majority of proposals related to random forests are unavailable. Furthermore, other statistical software such as *SAS*, *SPSS*, *WEKA* and Python's *scikit-learn* machine learning library are either worse with respect to algorithm availability, or have a near complete overlap with the types of random forests already implemented in *R*. It is true that some of the proposals can easily be obtained by tweaking aspects of an algorithm that has already been programmed (such as sampling *without* replacement in the *randomForest* package to obtain the approach proposed by Ho (1998)), but for others this is not so easy (such as the dynamic integration proposed by Tsymbal *et al.* (2006)). Another option is to actually program these proposals from scratch, which is certainly possible but would be a highly time consuming and difficult undertaking.

An alternative approach and the one taken in this text, is to perform a comparison by way of a *meta-analysis*. This is done by amalgamating the comparisons

found in various papers in order to be able to compare *all* of the different approaches with each other, and not just a subset of approaches based on the availability of the code needed to implement them. The data set for the meta-analysis was collected from all 34 papers associated with research on random forests from 2001 to 2015 that could be found in the literature (the full list of papers is provided in Appendix B.1). Each observation in the data set included a specific algorithm's performance on a particular benchmark data set as recorded by a specific paper. In addition to performance, other aspects pertaining to the experimental design were also included, adding up to 19 different variables characterising each of the 4295 observations collected. Examples of these variables include the sizes of training and test data sets, the approach used to estimate the generalisation performance of each algorithm, and the method that was used to compare the different algorithms. More details regarding the meta-analysis data set can be found in Appendix B.2.

The immediate concern here is the degree to which results from different papers are commensurable (owing to different data subsetting, algorithm tuning, data sets used, and various other aspects of experimental design particular to a paper). This issue is dealt with in the following section, where it is shown not to be detrimental in a meta-analysis. More specifically, statistical comparisons between algorithms across different papers can be made given that the same data sets are used, and that certain reliability assumptions are met.

7.2 Statistical Comparisons over Multiple Data Sets

At the outset it is useful to clarify the distinction between a *classifier* and an *algorithm*. A classifier is a function that has been estimated using the training data, and which can then be used to classify future observations. An algorithm refers to the set of steps that characterises the way in which a classifier is to be obtained using the training data. For example, two Forest-RI models fit to the training data with different tuning parameter values are considered to be two different classifiers, although they make use of the same algorithm. Therefore, to be clear, the purpose of the proposed meta-analysis is to compare multiple *random forest algorithms* over multiple data sets. To contrast this with other approaches, Figure 7.1 shows different possible scenarios commonly found in research regarding comparisons of classification algorithms (Dietterich, 1998).

However, before comparing different algorithms it is important to consider what exactly should be measured to compare them. As already mentioned, the study will focus on measurements that can be represented as numeric quantities. This however still leaves many available options from which to

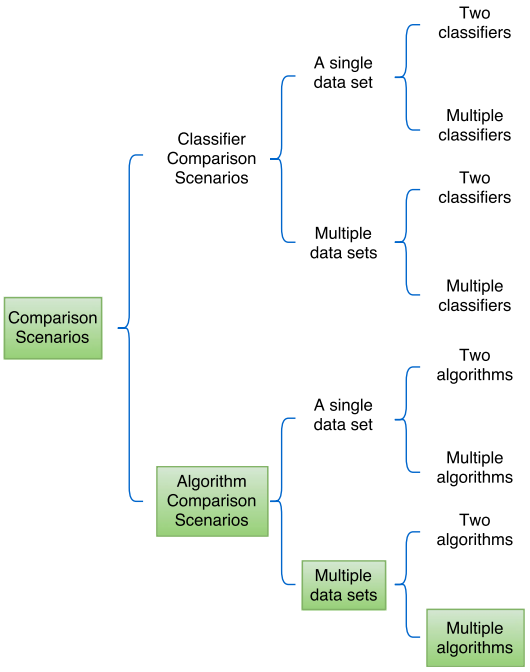

Figure 7.1: Comparison scenarios: *The green blocks correspond to the scenario associated with the meta-analysis in this text.*

choose. Furthermore, once one (or several) measures have been selected, it is almost always the case that this quantity can only be *estimated* from the data. Therefore, it is also important to decide exactly how to estimate the performance measure of choice. Finally, once selected performance measures have been estimated for each algorithm, differences between them can be evaluated using various *statistical tests*. Therefore, the following sections contain a brief discussion of different performance measures (Section 7.2.1), ways in which these performance measures can be estimated (Section 7.2.2), and how having estimated these measures, statistical tests can be performed to compare the algorithms (Section 7.2.3).

7.2.1 Algorithm Performance Measures

In binary classification most of the commonly used performance measures can be derived from a *confusion matrix*, as shown in Table 7.2. This is a 2×2 matrix where the cells contain a tally of the agreement or disagreement between the class labels predicted by an algorithm, and the true class labels.

Probably the most commonly used performance measure is classification error, *i.e.* $Err = \frac{FP+FN}{N}$, or equivalently classification accuracy, *viz.* $1 - Err = \frac{TP+TN}{N}$. However, some researchers have questioned the use of accuracy as an appropriate measure of algorithm performance (Provost *et al.*, 1998). Specifi-

Table 7.2: A confusion matrix for binary classification.

		Predicted $g = 1$	**Predicted** $g = 0$
Actual	$c = 1$	TP (True Positive)	FN (False Negative)
	$c = 0$	FP (False Positive)	TN (True Negative)
		P	N

cally, consider the following two points:

- Accuracy implicitly assumes symmetric loss (which is in many situations not the case);
- Maximising accuracy on a given data set assumes that the observed class distribution is in fact the true class distribution (which could also be false, especially if the training data set is small).

The counterargument to these remarks attempts to justify the use of accuracy as follows: an algorithm that is the most accurate (using symmetric loss and the above assumption regarding the observed class distribution), will also be the algorithm minimising asymmetric loss and/or the loss under a different class distribution. Note that the discussion is with regard to algorithms, and *not* classifiers. But Provost *et al.* (1998) state that the above justification is unreasonable since it is rare to have the knowledge necessary to redefine the loss associated with each class or its representation in order to suit accuracy as performance measure.

Instead *balanced* measures are preferred, such as an *ROC* curve where the trade-off between $sensitivity = \frac{TP}{TP+FN}$ (true positive rate) and $1-specificity = 1 - \frac{TN}{FP+TN}$ (false positive rate), is displayed for a grid of classification thresholds corresponding to different losses associated with each class. In fact, using *ROC* curves, Provost *et al.* (1998) show that on several data sets, a single algorithm rarely dominates over the entire range of classification thresholds. This was also observed by Bradley (1997). These observations once again put into question the use of accuracy as performance measure. However, graphical methods such as *ROC* curves can become difficult to interpret when many

algorithms have to be compared across many data sets.

Alternatively, the so-called *area-under-the-curve* (AUC) measure can be used. The AUC is computed as the area below the ROC curve and reduces the graphical information into a single numeric quantity. Unfortunately, the AUC has a major drawback: it treats the loss associated with each class differently, depending on the algorithm used. This property is inherent in the way that the AUC is defined. For more details, see Hand (2009), Hand (2010) and Hand and Anagnostopoulos (2013). Therefore, instead of using the AUC, Hand (2009) proposed the *H-measure*. This measure is based on a Bayesian approach. It specifies a beta prior distribution over the relative losses associated with each class, where the losses are independent of the employed algorithm. As with any prior, the parameters of the distribution can be set to incorporate expert knowledge regarding the problem. Alternatively, Hand and Anagnostopoulos (2014) recommend $\beta(\frac{TP+FN}{N}+1, \frac{FP+TN}{N}+1)$ as default choice.

Ultimately, the selection of an appropriate measure is highly dependent on the problem at hand and on what exactly is of interest to be measured. This complicates a comparison of algorithms over several benchmark data sets since each set essentially puts forward a different problem to be solved. Therefore, if feasible, it is typically a good idea to make use of several performance measures in a comparative study. Table 7.3 provides a list of algorithm performance measures, together with typical scenarios where each measure is considered appropriate (Santafe *et al.*, 2015).

Table 7.3: Algorithm performance measures for binary classification.

Performance measure	**Calculation**	**Appropriate scenario**
Error	$Err = \frac{FP+FN}{N}$	Balanced data
Accuracy	$Acc = \frac{TP+TN}{N}$	Balanced data
Sensitivity	$s = \frac{TP}{TP+FN}$	Skew data
Specificity	$sp = \frac{TN}{FP+TN}$	Skew data
Precision	$p = \frac{TP}{TP+FP}$	Skew data
Kappa	$\kappa = \frac{Acc-P_e}{1-P_e}$	Skew data
F-score	$F_\xi = \frac{(\xi^2+1)p \cdot s}{\xi^2 p+s}$	Skew data
H-measure	Based on segments of ROC	Balanced data/skew data

The *kappa statistic*, $\kappa = \frac{Acc-P_e}{1-P_e}$, where $P_e = (\frac{TP+FN}{N} \cdot \frac{TP+FP}{N}) + (\frac{TN+FP}{N} \cdot \frac{TN+FN}{N})$ can be seen as an adjusted accuracy rate. It measures the agreement between an algorithm and the truth while correcting for agreement that is observed by chance. The *F-score*, $F_\xi = \frac{(\xi^2+1)p \cdot s}{\xi^2 p+s}$, where ξ is a tuning parameter, is useful for skewed data. More specifically, the F-score can be used to measure the performance of an algorithm at correctly classifying observations

belonging to a minority class. Importantly, note that any binary classification measure can be turned into a measure for multiclass classification by using either the *one-versus-all* (Rifkin and Klautau, 2004) or the *one-versus-one* (Allwein *et al.*, 2000) approach. Hence the above remarks extend beyond the context of binary classification. Lastly, even after employing all of the measures mentioned in this section, an algorithm's computational efficiency, scalability, robustness, stability and level of interpretability remain unmeasured. These are all factors that should be borne in mind when drawing conclusions from a comparison.

7.2.2 Estimating Algorithm Performance

Among the most common ways of estimating the performance of an algorithm is the use of resampling methods. Two popular choices are the bootstrap and k-fold cross-validation. However, in any form of estimation, the bias-variance trade-off has to be taken into consideration.

In both approaches a high bias can be remedied using various *bias-correction* methods. For the bootstrap, Efron (1983) prosed the bias-corrected *.632 bootstrap*, which is a weighted average between the ordinary leave-one-out estimate and the training error. The drawback with this estimate is that in situations where the data is closely fitted, the contribution to the error will shift towards that of the leave-one-out estimate since the training error will tend towards zero. By down-weighting the error, the estimate will be too optimistic. To address this problem, Efron and Tibshirani (1997) introduced the *.632+ bootstrap* which involves incorporating into the estimate a measure of the amount of overfitting taking place. In cross-validation, the number of folds (the value for k) largely controls the bias-variance trade-off. Large values for k reduce the bias (by increasing the training set size used at each step), but increases the variance (with small test sets and averaging being more sensitive to changes in the data). In addition, proposals have been made to facilitate bias-correction in cross-validation using bias estimation methods similar to the *jackknife* (Efron and Efron, 1982). Approaches of this kind can be found in Burman (1989) and Fushiki (2011).

In terms of variance, both resampling methods produce a final estimate by averaging over the estimates obtained at each sampling step. This in turn reduces the variance of the estimation procedure. Extending this idea and in an attempt to further reduce the variance, it is possible to repeatedly perform a resampling method (such as ten runs of k-fold cross-validation), and to then average over the obtained result from each run. Furthermore, in cross-validation the value k can be adjusted to reduce the variance, but at a price of increased bias. A particularly interesting approach to reducing variance is that of *bolstered estimation* (Braga-Neto and Dougherty, 2004). This approach in-

volves placing densities around data points (identical for every point, but this can differ per class) and weighting misclassification error for a point by only using the proportion of its density that is on the wrong side of the decision boundary. Therefore, if an observation lies close to the decision boundary, only a part of its density will be counted — even if it lies on the wrong side. This renders the estimation approach more robust in terms of the obtained decision boundary, and less sensitive to changes in the data, which in turn aids in reducing the variance.

A fundamental problem with resampling methods occurs when a data set is small. This is often the case in certain fields, such as for example in genomics. In small data settings, further splitting at each step results in high variance. To capture this additional uncertainty, Isaksson *et al.* (2008) advise the use of confidence intervals, claiming resampling estimates to be unreliable. However, Kohavi (1995) studied both the bootstrap and cross-validation on moderately sized benchmark data sets and recommend ten-fold cross-validation as an appropriate accuracy estimation method.

7.2.3 Comparing Classification Algorithms

Suppose two algorithms are to be compared. The estimated performances from the algorithms can be treated as random variables P_1 and P_2. Associated with each of these random variables is an unknown probability distribution and therefore the true difference between the densities of P_1 and P_2 are also unknown. Formally, suppose H_0 denotes the *null hypothesis* which states that there is no difference between the performances of the two algorithms. Furthermore, let H_1 denote the *alternative hypothesis* stating that there is indeed a difference. Assuming that H_0 is true, a *test statistic* can be computed which attempts to convert as much of the information contained within the differences in the observed values from the two algorithms into a quantity that is associated with some *known* distribution. Using knowledge of this distribution it is possible (in some restricted sense) to ascertain whether one of the two algorithms is superior or not. This is usually done by computing a *p-value*, the probability of actually observing what was observed under the assumption that H_0 is true (using the derived distribution). If this probability is low, the researcher can choose to reject H_0, since the evidence from the observed data then questions the initial assumption of no difference.

There is a very important, but subtle point that needs to be made here. The described p-value is dependent on not only the chosen test statistic, but also on the observed values from the experiment. It does not contain any information regarding the likelihood that H_0 is actually true or not. The p-value is only capable of providing information about the probability of a realisation of what was *observed from the experiment* under the derived distribution. Therefore,

to reject the null hypothesis is ultimately up to the researcher who trusts that the experimental design providing the observed values as well as the employed test statistic are appropriate, and that the only conclusion left to be made is that the initial assumption (of no difference) was incorrect (Santafe *et al.*, 2015).

A hypothesis test in which multiple algorithms are compared, is known as an *omnibus test*[1]. In an omnibus test, H_0 is simply taken as the statement of no difference between *all* the approaches whereas H_1 states that at least one of the algorithms has a performance that is significantly different from the rest. Concretely, suppose a test has to be conducted where L algorithms were fit to D data sets. Let p_{ld} be the performance of algorithm $l \in L$ on data set $d \in D$. The purpose of the test is to ascertain whether there is a significant difference between the algorithms based on the values p_{ld}. As is typical for any hypothesis test, a decision threshold or *significance level* (α), is specified to decide when the obtained p-value is sufficiently small — indicating that the evidence against H_0 is strong enough to reject it. Figure 7.2 shows common comparison scenarios, together with appropriate statistical tests for each scenario (Santafe *et al.*, 2015).

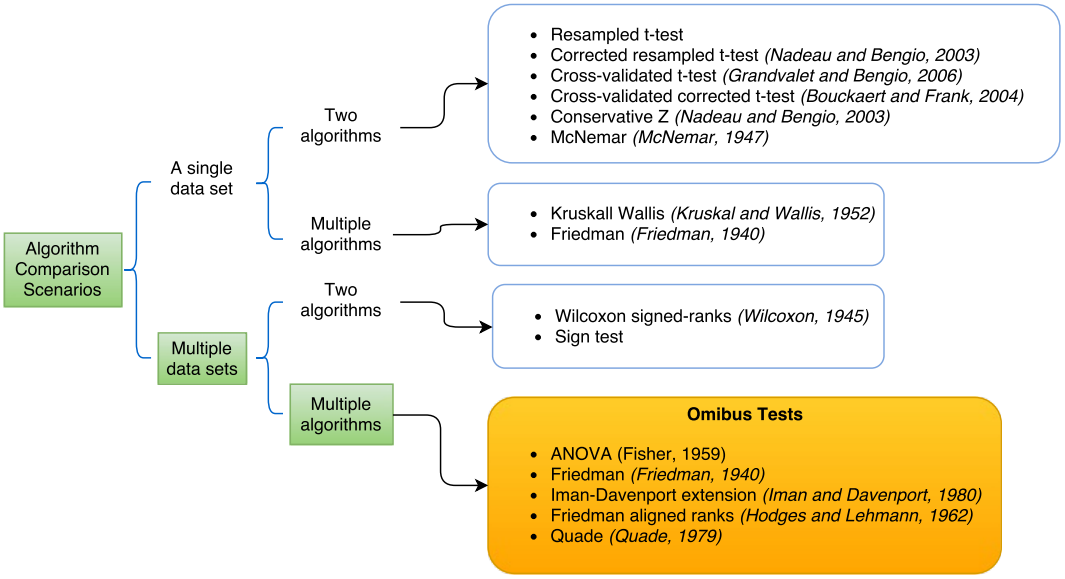

Figure 7.2: Omnibus statistical tests for comparing multiple classification algorithms over multiple data sets: *the orange block represents the tests appropriate in the meta-analysis.*

Among all of the omnibus tests an *ANOVA* (Fisher, 1955) is the only *parametric* test that relies on the following assumptions: observations (p_{ld}) are

[1]The definition of omnibus is "comprising of several items".

sampled from a normal distribution, and the associated random variables (P_{ld}) have equal variance. Both these assumptions are unlikely to hold when comparing different algorithms over a range of benchmark problems. Therefore it is advised to refrain from using an ANOVA in these situations (Demšar, 2006). Instead, *non-parametric* tests are preferred (Demšar, 2006; Garcia and Herrera, 2008; García *et al.*, 2010). In this regard, the following non-parametric omnibus tests are discussed.

- **Friedman** (Friedman, 1937): The Friedman test starts by ranking each algorithm based on performance, where ties are treated by assigning an average rank to the algorithms concerned. Let r_{ld} be the rank of algorithm l on data set d, then $\bar{R}_{l\cdot} = \frac{1}{D}\sum_{d=1}^{D} r_{ld}$ is the average rank of algorithm l over all the data sets $d \in D$. Under H_0, stating no difference, *i.e.* $\bar{R}_{1\cdot} = ... = \bar{R}_{L\cdot}$, the test statistic

$$\chi_F^2 = \frac{12D}{L(L+1)}\left(\sum_{l=1}^{L}\bar{R}_{l\cdot}^2 - \frac{L(L+1)^2}{4}\right)$$

  has a χ^2 distribution with $L-1$ degrees of freedom for large enough $L(>5)$ and $D(>10)$. However, Iman and Davenport (1980) have criticised the test based on empirically observing the inaccuracy of the χ^2 approximation. Hence they proposed a modified statistic.

- **Iman-Davenport extension** (Iman and Davenport, 1980): The extension uses the same ranking scheme as the Friedman test, but modifies the statistic as follows:

$$F_{ID} = \frac{(D-1)\chi_F^2}{D(L-1)-\chi_F^2}$$

  which is distributed according to an F distribution with $(L-1)$ and $(L-1)(D-1)$ degrees of freedom.

- **Friedman aligned ranks** (Hodges *et al.*, 1962): When the number of algorithms being tested is small, it might be of interest to incorporate the information regarding the difference in rank between the algorithms. The Friedman aligned rank test ranks each algorithm based on $p_{ld}^a = p_{ld} - \bar{p}_d$, where $\bar{p}_d = \frac{1}{L}\sum_{l=1}^{L} p_{ld}$ is the average performance on the d^{th} data set. Then the sum of ranks is computed over the algorithms, *i.e.* $R_{l\cdot} = \sum_{d=1}^{D} r_{ld}$, and over the data sets, *i.e.* $R_{\cdot d} = \sum_{l=1}^{L} r_{ld}$. Finally, the test statistic is obtained as follows:

$$F_{AR} = \frac{(L-1)\left[\sum_{l=1}^{L} R_{l\cdot}^2 - (LD^2/4)(LD+1)^2\right]}{\left[LD(LD+1)(2LD+1)\right]/6 - (1/L)\sum_{d=1}^{D} R_{\cdot d}^2}$$

which follows a χ^2 distribution with $L-1$ degrees of freedom.

- **Quade** (Quade, 1979): Some data sets may pose a more difficult challenge than others. The Quade test is an interesting approach where comparisons are made over data sets that are weighted based on the observed performances from the algorithms. The test starts by computing the ranks r_{ld} and performances p_{ld}. The difference between the best and worst performing algorithms on each data set is thereafter obtained as $p_d^* = max_l(p_{ld}) - min_l(p_{ld})$. Let $\widetilde{R}_{\cdot 1}, ..., \widetilde{R}_{\cdot D}$ represent the ranked data sets according to the previously calculated differences. The weighted rank for algorithm l is then $\widetilde{R}_{l\cdot} = \sum_{d=1}^{D} \widetilde{R}_{\cdot d}\big(r_{ld} - (L+1)/2\big)$. The test statistic is

$$F_Q = \frac{(D-1)B}{A-B},$$

where $A = D(D+1)(2D+1)L(L+1)(L-1)/72$ and $B = (1/D)\sum_{l=1}^{L} \widetilde{R}_{l\cdot}^2$. The F_Q statistic follows an F distribution with $(L-1)$ and $(L-1)(D-1)$ degrees of freedom.

Following an experimental framework proposed by Demšar (2006), García *et al.* (2010) conducted an analysis of the different tests in terms of their *power*, which is the probability of rejecting H_0 when it is indeed false. From their results, it was observed that the Friedman aligned ranks test and the Quade test had higher power and outperformed the Friedman and Iman-Davenport extension tests when the number of algorithms compared was small (< 5). However, when the number of algorithms increased, the power of the Iman-Davenport extension also increased, while the performance of the Friedman aligned ranks test dropped. The Quade test performed the best in terms of power, but was found to be very sensitive to changes in the selected benchmark data sets. As a result, recommendations of García *et al.* (2010) are as follows:

- To use the Friedman aligned ranks test or the Quade test in cases where the number of algorithms being compared is small;
- The Quade test can also be used when the number of algorithms is large, however the researcher must be aware of its sensitivity towards the choice of data sets. Therefore, the appropriateness of the selection and possible effects should be considered carefully;

– In situations where the number of algorithms compared is large (≥ 5), the Iman-Davenport test is a good alternative — because of its simplicity, increased power, and insensitivity to changes in data sets used.

Importantly, note that at this stage, none of the tests provide any information regarding which of the algorithms' performances differ significantly. Once a significant difference is detected, pairs of algorithms still need to be compared. The test statistics for pairwise comparisons between algorithms l_1 and l_2 in the different omnibus tests are given in Table 7.4.

Table 7.4: Pairwise comparisons in omnibus tests between Algorithms l_1 and l_2 (statistics follow a standard normal distribution).

Friedman/Iman-Davenport	**Friedman aligned ranks**	**Quade**
$Z_F = \frac{\bar{R}_{l_1\cdot} - \bar{R}_{l_2\cdot}}{\sqrt{\frac{L(L+1)}{6D}}}$	$Z_{AR} = \frac{(R_{l_1\cdot} - R_{l_2\cdot})/D}{\sqrt{\frac{L(LD+1)}{6}}}$	$Z_Q = \frac{T_{l_1\cdot} - T_{l_2\cdot}}{\sqrt{\frac{L(L+1)(2D+1)(L-1)}{18D(D+1)}}}$, where $T_{q\cdot} = \frac{\sum_{d=1}^{D} Q_{d\cdot} r_{qd}}{D(D+1)/2}$ with $q = l_1, l_2$.

When performing multiple comparisons (between all pairs of algorithms), a significant difference might be detected simply by chance. Concretely, the *Type I* error associated with a test is the probability of rejecting H_0 when in fact it is true. This is closely linked to the significance level (α), where essentially the specified threshold is the maximum probability of a Type I error that a researcher is willing to accept. When conducting multiple tests, it is not sufficient to control the Type I error for each comparison separately, since such an error might arise simply by chance. For example, with a single pairwise comparison, the probability of not making a Type I error is $1 - \alpha$. If L algorithms are compared, this amounts to $\binom{L}{2}$ comparisons. The probability of making at least one Type I error is then $1 - (1 - \alpha)^{\binom{L}{2}}$. This means that if only five algorithms are compared, this probability is 40%, which is much larger than 5%. Therefore, *post-hoc* tests aim to adjust α (or equivalently the p-values obtained from pairwise comparisons in Table 7.4) to control the *family-wise* error, which is the probability that at least one Type I error is made among multiple tests. Figure 7.3 shows different post-hoc tests for either comparing a novel algorithm (a control) against all others, or for conducting all pairwise comparisons (Santafe *et al.*, 2015).

When comparing a novel competitor, the above type of adjustments to the significance level can be done in a single step, in two steps, or through a more complicated multiple step-up or step-down procedure. Let H_{0l} denote the null hypothesis associated with the l^{th} comparison, and let pv_l denote the p-value.

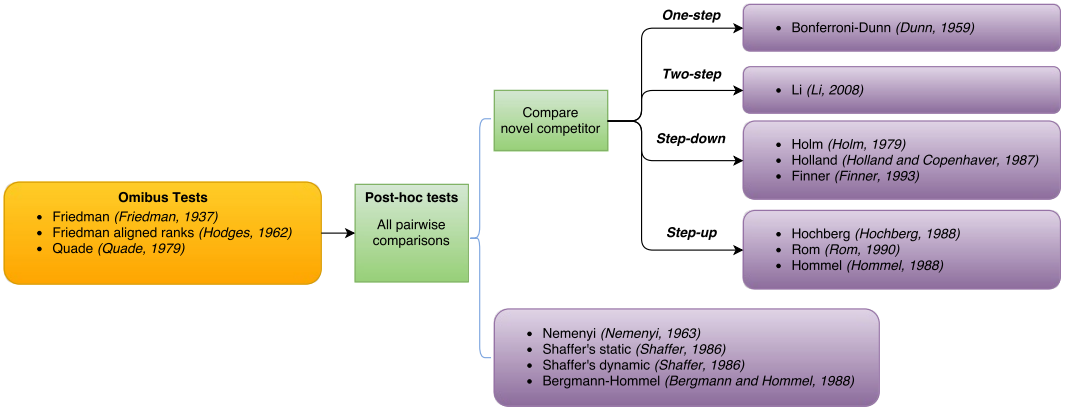

Figure 7.3: Post-hoc tests for comparing multiple classification algorithms over multiple data sets: *the purple blocks represent tests appropriate for the meta-analysis.*

Then each post-hoc test makes the following adjustment to α in multiple comparisons:

- **Comparing a novel competitor and all pairwise comparisons**
  - **One-step**
    - **Bonferroni-Dunn** (Dunn, 1959): $\forall l$ reject H_{0l} if $pv_l < \alpha/(L-1)$.
  - **Two-step**
    - **Li** (Li, 2008): *Step 1.* Reject $H_{0l} \forall l$, if $pv_{L-1} < \alpha$, where the p-values are ordered in ascending order (pv_{L-1} being the largest). Otherwise, do not reject $H_{0(L-1)}$ and proceed to *Step 2.* Reject $H_{0l} \forall l = 1, ..., L-2$ if $pv_l < \frac{(1-pv_{L-1})\alpha}{1-\alpha}$.
  - **Step-down**: with ordered p-values, reject $H_{01}, ..., H_{0l^*}$ where
    - **Holm** (Holm, 1979): $l^* = min\{l : pv_l > \alpha/(L-l+1)\}$.
    - **Holland** (Holland and Copenhaver, 1987): $l^* = min\{l : pv_l > 1-(1-\alpha)^{1/(L-l)}\}$.
    - **Finner** (Finner, 1993): $l^* = min\{l : pv_l > 1-(1-\alpha)^{l/(L-1)}\}$.
  - **Step-up**: with ordered p-values
    - **Hochberg** (Hochberg, 1988): Reject $H_{0(l^*+1)}, ..., H_{0(L-1)}$ where $l^* = max\{l : pv_l > \alpha/(L-l)\}$.
    - **Rom** (Rom, 1990): Reject $H_{0(l^*+1)}, ..., H_{0(L-1)}$ where $l^* = max\{l : pv_l > \alpha_l^R\}$ and $\alpha_{(L-1)}^R = \alpha, \alpha_{(L-2)}^R = \alpha/2$,

$$\alpha_{(L-j)}^R = \left(\sum_{l=1}^{j-1} \alpha^l - \sum_{l=1}^{j-2} \binom{j}{l} \cdot (\alpha_{(L-1)-l}^R)^{j-l} \right) / j,$$

with $j = 3, ..., L-1$.

- **Hommel** (Hommel, 1988): $\forall l$ reject H_{0l}, if $pv_l < \alpha/l^*$ where $l^* = max\{l : p_{(L-1)-l+m} > m\alpha/l, \forall m = 1, ..., l\}$.

- **All pairwise comparisons**
  - **Nemenyi** (Nemenyi, 1962): $\forall l \in \{1, ..., L-1\}$ reject H_{0l} if $pv_l < \frac{\alpha}{L(L-1)/2}$.
  - **Shaffer's static** (Shaffer, 1986): Using logical rules it can be determined which hypotheses may automatically be rejected, given that others have already been rejected. More specifically, let t_l be the maximum number of hypotheses $H_{0l'}, l' \in \{l, ..., L-1\}$, that can be true given that any $l-1$ of the hypotheses have already been rejected. Then reject $H_{01}, ..., H_{0(l^*-1)}$, where $l^* = min\{l : pv_l > \alpha/t_l\}$.
  - **Shaffer's dynamic** (Shaffer, 1986): The value t_l^* is computed sequentially (using ordered p-values), representing the maximum number of hypotheses that can be true, given that previously tested hypotheses in the sequence have already been rejected. Then reject $H_{01}, ..., H_{0(l^*-1)}$, where $l^* = min\{l : pv_l > \alpha/t_l^*\}$.
  - **Bergmann-Hommel** (Bergmann and Hommel, 1988): The procedure is based on finding all combinations of possible hypotheses (*exhaustive sets*) that could be true for the different comparisons. By forming the union of these sets, the *acceptance set* A is constructed, where every hypothesis not in A is rejected. The procedure is computationally intensive and the most complicated compared to the other post-hoc tests. For a more detailed exposition, together with illustrative examples, see Garcia and Herrera (2008).

As was the case with the omnibus tests, Garcia and Herrera (2008) and García *et al.* (2010) analysed the post-hoc tests in terms of their power. The following recommendations follow from their respective studies:

- The Bonferroni-Dunn and Nemenyi tests are the simplest to implement. However, both of these tests have little power;
- García *et al.* (2010) found that the Holm, Hochberg, Hommel, Holland and Rom tests have similar power, with the trend being that the more complicated tests (such as Rom and Hommel) are slightly more powerful;
- The simpler Finner test managed to outperform the others, with exception of the Li test in some cases. However, the Li test was found to be sensitive to changes in the selection of algorithms compared. Therefore the Finner test is preferred;

– In terms of pairwise comparisons, Garcia and Herrera (2008) found the Bergmann-Hommel test to perform the best, however they acknowledge the complexity of the test in terms of both implementation and explanation. It is also computationally intensive;

– Garcia and Herrera (2008) recommend the use of the Shaffer static test for all pairwise comparisons because of its simplicity and the benefit of additional information from logically related hypotheses.

Therefore in summary, to compare multiple classification algorithms over multiple data sets, the first step is to perform an omnibus test. The recommended omnibus test (given L is large, *i.e.* $L \geq 5$) is the Iman-Davenport test. If the associated null hypothesis is rejected, an all pairwise post-hoc test is performed in order to compare all the algorithms on a pairwise basis. The Shaffer static test is the recommended pairwise post-hoc test, since it is a simple test with high power.

7.3 An Evaluation of Random Forest Comparative Studies

In this section, as part of the meta-analysis, the existing research on random forests is evaluated with regard to the recommended experimental design and methodology for comparative studies discussed in the previous section.

7.3.1 An Evaluation of Performance Measure Selection

It has already been mentioned that misclassification error and classification accuracy are considered inappropriate performance measures, unless the data set on which the performance is recorded, is well balanced (Provost *et al.*, 1998). Since in practice the latter case is likely to occur less frequently, alternative measures are mostly preferred. However, *all* of the collected papers in the meta-analysis mainly made use of classification error or accuracy. To be fair, this observation is not unique to research on random forests. Demšar (2006) studied 157 papers that were accepted to the *International Conference on Machine Learning* from 1999 to 2003. Focusing on comparative studies, on average 75.8% of the papers included accuracy as performance measure, while 66.6% of the papers made use of accuracy *exclusively*.

7.3.2 An Evaluation of Performance Estimation

In Figure 7.4, boxplots of the error rates reported for Forest-RI in the meta-analysis papers are shown for the ten most popular data sets.[2]

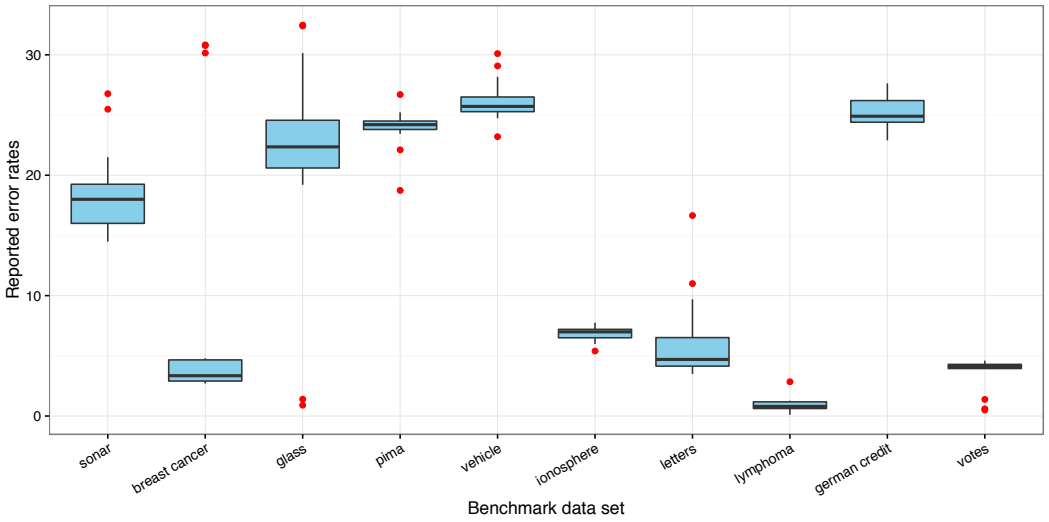

Figure 7.4: Reported error rates for Forest-RI for the ten most popular data sets in the meta-analysis papers.

The picture presented is one of a fair amount of variability, given that the same algorithm was used on the same data set in each case. Presumably much of the variation is due to differences in parameter tuning. Gross outliers are observed in the *breast cancer* and *glass* data sets and could possibly be attributed to human error during the reporting of results.

To investigate the variability of the error rates reported for Forest-RI in a more rigorous manner, the following strategy was used. The first step was to consider all combinations of ten data sets selected from the top fifteen most popular data sets in the meta-analysis papers. Thereafter, the combination of ten data sets corresponding to the largest number of papers that reported error rates for Forest-RI was found. This resulted in the data set combination that included the *sonar, glass, pima, vehicle, ionosphere, letters, german credit, votes, waveform* and *vowel* data sets that were found in four different papers (Breiman, 2001*a*; Cutler and Zhao, 2001; Robnik-Šikonja, 2004; Rodriguez *et al.*, 2006), each of which reported an error rate for Forest-RI for each data set. The reported Forest-RI performance (in each paper) was treated as the performance of a "unique" algorithm, and a comparison was made using an

[2]Details regarding benchmark data sets mentioned in this section can be found in Appendix B.3.

omnibus test. Therefore, the null hypothesis was that there is no difference in the reported error rates for the Forest-RI's across different papers.

The Iman-Davenport test resulted in a p-value equal to 0.0407, which with an $\alpha = 0.05$, rejected the null hypothesis. However, only four "algorithms" were compared. Hence the Friedman aligned ranks test was also conducted and resulted in a p-value $= 0.0217$ (more evidence against H_0). Lastly, Figure 7.4 pointed towards possible intrinsic differences in the difficulty presented between data sets, so the Quade test (which attempts to take these differences into account) was also used. This yielded a p-value equal to 0.0431. Thus, irrespective of which omnibus test was used, a statistically significant difference was detected in the reported error rates for Forest-RI on the same collection of data sets across the four different papers.

7.3.3 An Evaluation of Comparison Methods

In Figure 7.5, the methods used to compare the various algorithms in the papers included in the meta-analysis are depicted.

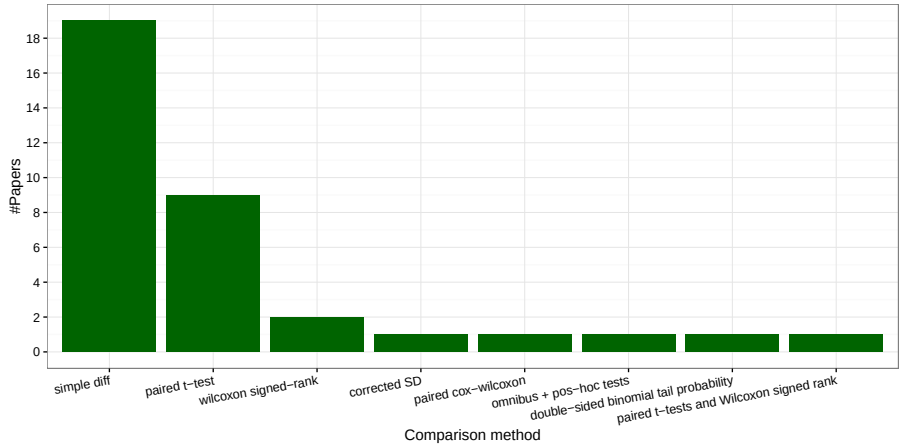

Figure 7.5: Methods used to compare different algorithms over multiple data sets in the papers considered for the meta-analysis.

The most common method (which is used in 56% of the papers) is *simple diff*, where differences in performance are simply taken at face value. The algorithm outperforming the rest is declared to be the best, without testing statistical significance. After simple diff, the *paired t-test* is also popular. Although the paired t-test does at least provide some measure of statistical significance, it does not control the family-wise error. Therefore, from a paired t-test, a statement such as that Algorithm A outperformed Algorithm B in all data sets and in all comparisons considered, is erroneous. Such statements can however be

found in many papers.[3]

To ascertain to what extent the conclusions from random forest comparative studies would change if statistical tests where performed, the following experiment was performed. Using the reported accuracy from each paper, an appropriate omnibus test (Iman-Davenport) was conducted. All of the statistical tests were performed using the *R* package *scmamp* (Calvo and Santafé, 2015). The results are shown in Figure 7.6.

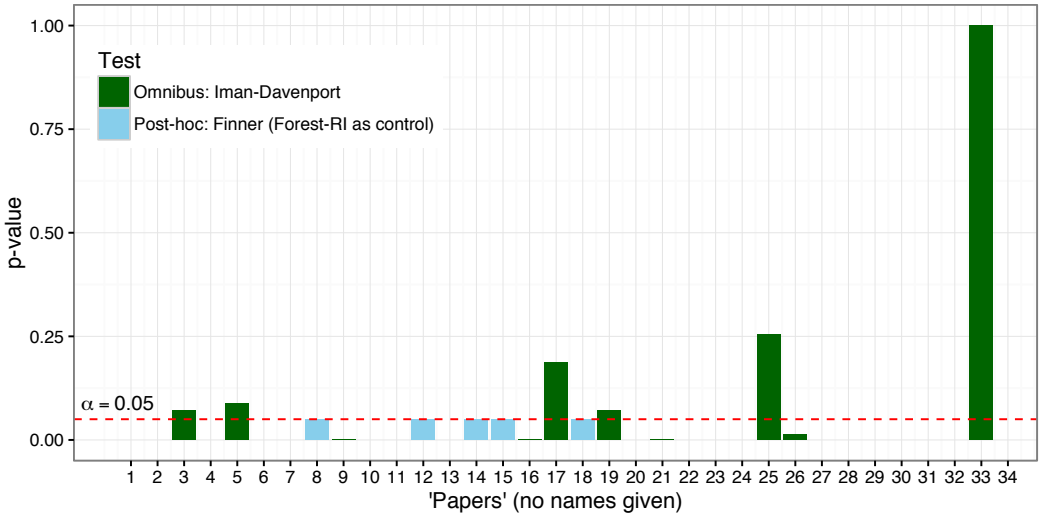

Figure 7.6: Omnibus and post-hoc test p-values from each paper considered for the meta-analysis. All p-values were computed using the reported accuracy from each paper.

Each dark green bar represents the p-value obtained from the omnibus test corresponding to a particular paper (the authors and paper titles are omitted, instead numbers are used to represent each paper). The dashed red line represents the significance level $\alpha = 0.05$. In Figure 7.6 it is seen that in fact six papers report results leading to no significant difference to be detected between *any* of the algorithms compared. Therefore in cases where the null hypothesis was rejected by the omnibus test, a post-hoc test (the Finner test to compare a control) was conducted, comparing (per paper) Forest-RI with all other algorithms in the paper. The five light blue bars in Figure 7.6 represent papers for which none of the compared algorithms were found to be significantly different from Forest-RI, with a pairwise p-value for every comparison to the control (Forest-RI) greater than 0.05. This means that in total, roughly a third ($11/34 = 32.4\%$) of the results from papers on random forests research

[3]In fact, even if the scenario is appropriate for using the t-test, Dietterich (1998) instead recommends using the McNemar test since it has higher power.

considered in the meta-analysis show no significant difference over and above Breiman's Forest-RI proposal.

While Breiman (2001*a*) was partly motivated to propose Forest-RI as a way to improve bagging trees in order to be competitive with AdaBoost (Freund and Schapire, 1995), the p-value for Breiman's own paper (Breiman, 2001*a*), using the Iman-Davenport test is 0.014 (which would not have rejected the null hypothesis at $\alpha = 0.01$). Furthermore, when performing the Finner post-hoc test for a pairwise comparison between a control (Forest-RI) and the other two algorithms in the paper, *viz.* AdaBoost and Forest-RC, the obtained p-values were 0.746 and 0.0457 respectively. This means that although Forest-RI outperformed Forest-RC, no statistically significant difference was actually observed with regard to AdaBoost. However, Forest-RI is clearly a useful contribution, for which the above remarks illustrate that improved performance on benchmark data sets should not be the only criterion employed when novel algorithms are being compared.

7.3.4 An Evaluation of Reproducibility

It has been alluded to in Chapter 1 that an increasingly important aspect of modern research is reproducibility (Peng, 2011). Research can be made reproducible by sharing the data and the associated code used to perform the analysis. Most of the benchmark data sets used in random forests research is easily obtainable via the UCI machine learning repository (Blake and Merz, 1998). This is an essential step towards increasing the number of reproducible comparative studies. However, as already discussed, the software and algorithm implementations are not always made available. To somewhat gauge the impact of actually having access to an implementation of a particular algorithm, Figure 7.7 shows the top (eleven) most popular algorithms (in terms of the number of papers in which the algorithm was compared) for the papers considered in the meta-analysis.

The number of times that an algorithm appears in the literature is of course dependent on when it was proposed, and/or on how substantial the proposal was, given the state of the field at that particular time and thereafter. However, it is interesting to note that the top five and in total seven out of the eleven most popular algorithms have an *R* package directly associated with it. The availability of software/code to implement algorithms seems to play a pertinent role in determining which algorithms are included in subsequent comparative studies. The main issue with this is that biases might exist at the outset of comparative studies, stemming purely from software availability.

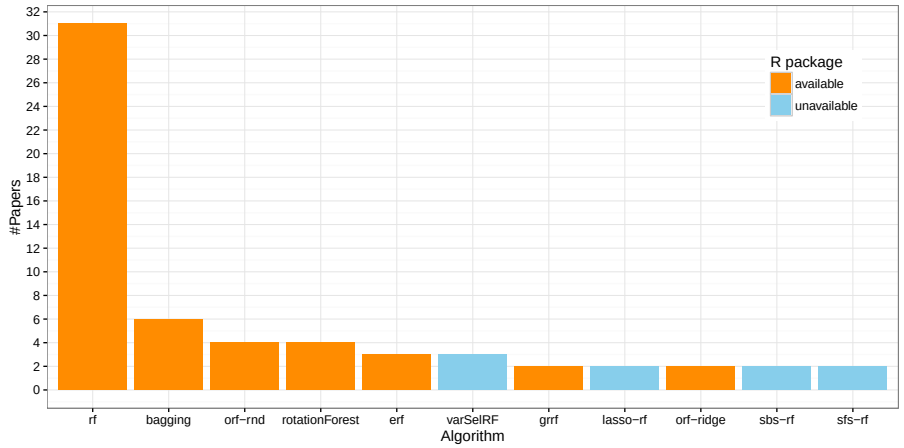

Figure 7.7: Popularity of algorithms in the meta-analysis papers, colour coded according to the availability of an implementation in *R*.

7.4 Comparing Classification Performance

In Section 7.1, the following question was posed: to what extent can it be established that a certain random forest is superior to others in terms of classification performance? This section is devoted to an attempt at answering the above question using the data collected in the meta-analysis. Before the results are presented, further comments and justifications for the meta-analysis are given with respect to the insights gained in the previous section.

- **Performance Measures**: Using data collected from academic papers imposes some restrictions on the way a comparison can be conducted. This means that the performance measure in the comparison is restricted to be classification accuracy (or equivalently, error) since this is the most commonly used performance measure in all of the papers considered.
- **Performance Estimation**: In the meta-analysis the performance of each algorithm will be averaged across the different papers in which the algorithm appeared, therefore reducing some of the variability resulting from different experimental designs. For example, the performance of Forest-RI on a particular data set will be averaged across all the papers that tested Forest-RI on that data set.
- **Comparison Methods**: As discussed in the previous section, more appropriate comparisons by means of omnibus and post-hoc statistical tests based on the reported accuracy rates are feasible in a meta-analysis.
- **Reproducibility**: In the meta-analysis, by using results in papers, an implementation of each algorithm is not required. This effectively re-

moves the bias that could exist when only a subset of algorithms are compared due to software being unavailable.

Following Demšar (2006), the only assumption in the meta-analysis, made to ensure that performance measures are commensurable, is that they are "reliable". Here, reliability refers to use of an appropriate estimation method, which includes a sufficient number of repetitions and/or resampling. In addition, it is preferable that across different papers, the data are split in an identical fashion. However, since each researcher is ultimately attempting to estimate the same quantity (the generalisation performance), a violation of this assumption is not seen as detrimental to the analysis. The various estimation methods and whether they were deemed reliable or not, is given in Figure 7.8. On the x-axis a label of "15 runs 70/30" (say) refers to the number of estimation runs, as well as the training and test split proportion, *i.e.* 70% of the data were used for training and the remaining 30% were used for testing.

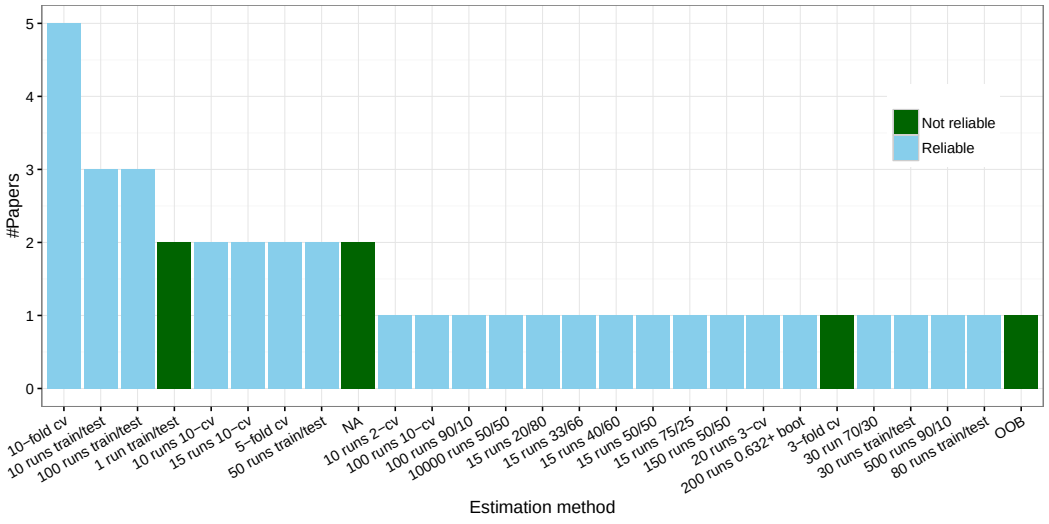

Figure 7.8: Performance estimation method used in the papers considered in the meta-analysis.

The most common approach in Figure 7.8 is ten-fold cross-validation, as was recommended by Kohavi (1995). Furthermore, most papers employ many runs of cross-validation, or of testing using a pre-specified train and test split proportion. However, six papers were deemed to use "unreliable" estimation methods. This was as a result of either too little detail given in the paper, or of the number of repetitions/folds not being sufficient. These papers were removed from the comparison. Furthermore, the ability to compare algorithms across different papers is dependent on the same data set being used for both

algorithms when their generalisation errors were estimated. In total, 134 algorithms (of which 94 are different random forests) were fitted to 201 different data sets. The question is, how does one compare all of these algorithms if the degree to which every algorithm overlaps the others is not always satisfactory?

A possible approach is to search over all combinations of $\binom{201}{D}$ data sets for different subset sizes of D (say $10, 20$ and 30), and to find the combination corresponding to the largest number of random forests that indeed overlap. However, the number of combinations is extremely large and even if this approach was computationally feasible, it would inevitably yield only a subset of random forests to be compared. Therefore, an alternative approach was considered.

The idea was to follow a two step procedure which firsts ranks the algorithms, and then performs a pairwise comparison between the top five algorithms. In the first step, a rank similar to the one in the Friedman test is computed for each algorithm. However, the rank is adjusted to take into account the issue of non-overlapping algorithms and data sets in the following ways:

1. The first adjustment involves dividing the rank for Algorithm l on data set d by the total number of algorithms L_d that are compared on data set d. The rationale behind this is that an algorithm ranked 2^{nd} (say) on data set d when compared to only two other algorithms should be considered less superior in comparison to an algorithm also ranked 2^{nd} compared to (say) twenty other algorithms on another data set d'. Therefore, multiplication by the weights $c_d = 1/L_d, d = 1, ..., D$ is referred to as the *competition factor* adjustment which adjusts the ranks according to the number of algorithms compared on each data set.

2. The second adjustment is to calculate a so-called *estimation spread factor*. This is defined as $es_l = \frac{D_l^C}{D} \in (0, 1]$, where D_l^C is the size of the complement of the collection of data sets over which the generalisation performance of Algorithm l was estimated. Here the rationale is that if Algorithm l is tested on more data sets, its average rank should bear more weight.

Concretely, the rank for Algorithm l, adjusted for competition and estimation spread, is computed as

$$\bar{R}_{l\cdot}^a = \frac{es_l}{D_l} \sum_{d=1}^{D_l} c_d r_{ld}, \tag{7.4.1}$$

where D_l is the size of the collection of data sets over which the generalisation performance of Algorithm l was estimated, and r_{ld} is the rank of Algorithm l on data set d. Since the above adjustments result in numeric quantities that are not as representative as traditional ranks, *i.e.* $(1, 2, 3...)$, the adjusted ranks are normalised to fall within the range $[1, L]$. Figure 7.9 displays the adjusted ranks for the *all-round* (not specific to high dimensional settings) algorithms. Due to the fact that many papers focus exclusively on the high-dimensional setting, these papers were analysed separately.

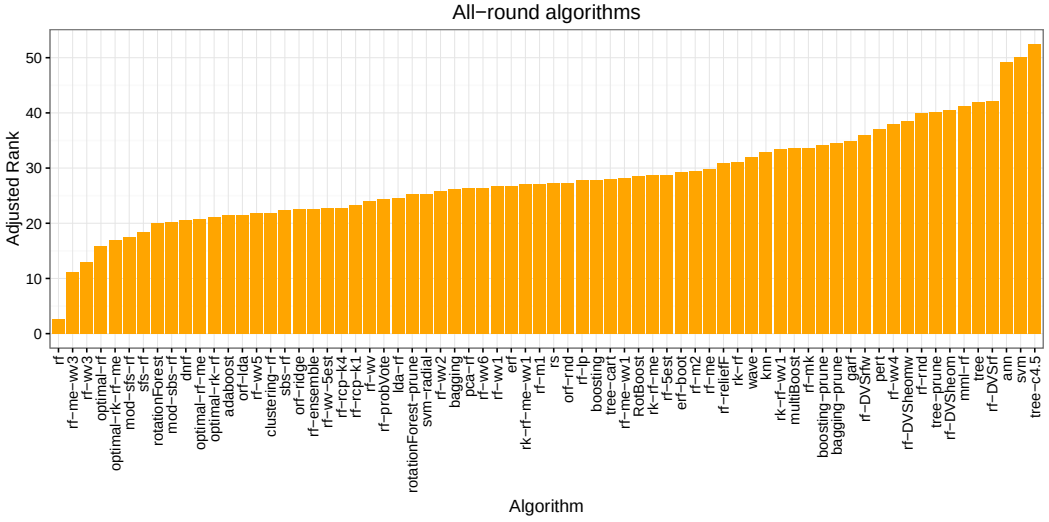

Figure 7.9: The adjusted ranks for all-round algorithms.

From Figure 7.9, the top five algorithms were *rf* (Forest-RI), *rf-me-wv3, rf-wv3, optimal-rf* and *optimal-rk-rf-me*. The top position went to Forest-RI, which is largely due to the algorithm's high estimation spread factor. The latter four algorithms are all modifications and/or combinations of modifications of Forest-RI, as proposed by Tripoliti *et al.* (2013). These contributions were very briefly discussed in Chapter 6. Specifically, *me* refers to adding the step of using a mixture of node impurity measures as was done in Robnik-Šikonja (2004); *rk* adds the step of randomly selecting the subset size $\xi \in \{1, ..., p\}$ at each node; and *wv*3 implies adding a weighted voting strategy using the *value difference metric*. The reader is referred to Wilson and Martinez (1997) and Payne and Edwards (1998) for more detail. Lastly, the *optimal* refers to adding a "search" step, where each time a tree is added to the ensemble, the algorithm checks whether the accuracy increases, or whether the correlation between the trees decreases. If this is not the case, it searches for the optimal combination of trees among all the trees that have been constructed thus far. The optimal combination of trees is the one associated with the largest

increase in accuracy, or with the largest decrease in correlation.

The next step in the comparison procedure was to compare the top five algorithms (identified in the previous step) on the largest possible overlap of data sets. This was done using appropriate statistical tests. The top left panel of Figure 7.10 provides an estimated kernel density plot based on the reported accuracy of each algorithm on the overlapping data collection. The largest possible overlap of data sets was found to consist of 24 data sets.[4] The densities alone already hinted at a possible association between the algorithms. Little difference was seen between rf-me-wv3 and rf-wv3, and between optimal-rf and optimal-rk-rf-me. However, differences were indeed perceived between the two previously mentioned groups and between Forest-RI and the other algorithms.

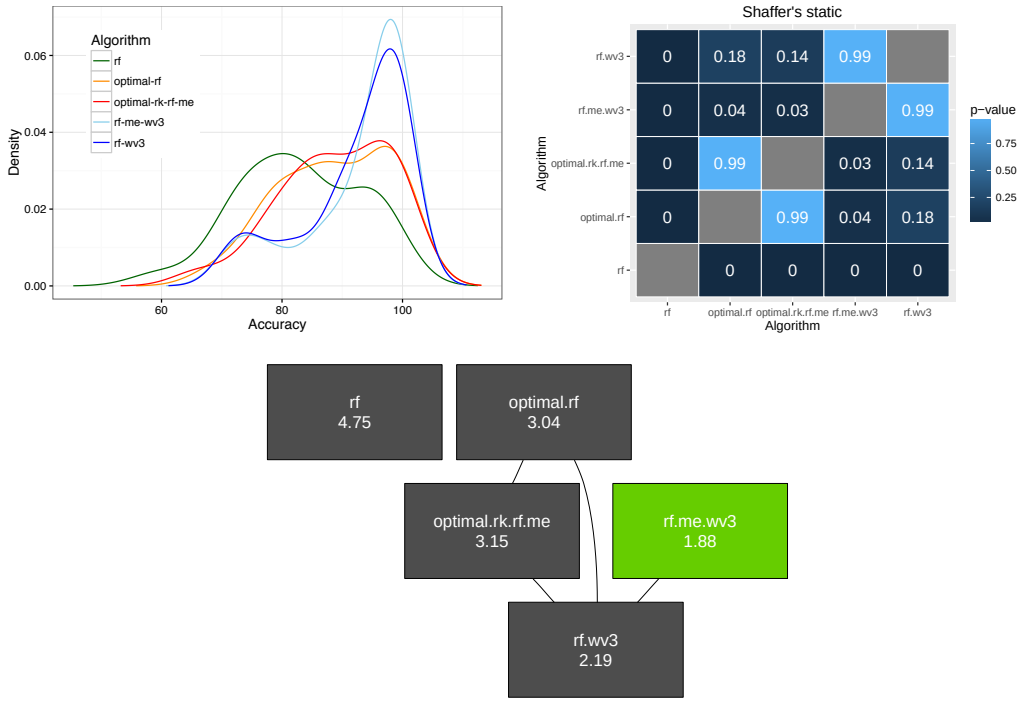

Figure 7.10: Results from comparing the top five all-round algorithms: Top left: *Kernel (Gaussian) density estimates of accuracies.* Top right: *Adjusted p-value matrix using the Shaffer static approach.* Bottom: *Pairwise comparisons plot.*

An Iman-Davenport omnibus test was performed and resulted in a p-value

[4]These included the *alzheimers, balance, breast, ecoli, glass, hays-roth, hepatitis, ionosphere, iris, mammo-mass, musk, parkinsons, pima, post-opt, sonar, spectf heart, survival, ta-eval, vehicle, votes, waveform, wdbc, wine* and *zoo* data sets.

equal to $3.08 \times 10^{-13} \approx 0$. Therefore, the null hypothesis that there is no difference between the top five algorithms can be rejected. Following the omnibus test, the Shaffer static post-hoc test was conducted. The adjusted p-values for pairwise comparisons are given in matrix form in the top right of Figure 7.10. Each cell represents a pairwise comparison, with the value in each cell corresponding to the adjusted p-value. For example, the top left cell is the pairwise comparison between rf-wv3 and rf with an adjusted p-value of practically zero. For ease of reading, the bottom of Figure 7.10 provides a quicker way to discern the pairwise relationship between the algorithms. It summarises the information in the matrix of adjusted p-values in the following way. Each block represents an algorithm, and a connection between two blocks symbolises no statistically significant difference between the two algorithms at a significance level of 0.05. The value in each block is the average rank of the algorithm as computed by the Friedman method (with the highest ranking algorithm shown in green).

From the bottom panel in Figure 7.10, it is clear that Forest-RI performed significantly worse than the other four algorithms. However, no significant difference was detected between the algorithms optimal-rf, optimal-rk-rf-me and rf-wv3. In addition, no significant difference was observed between rf-wv3 and rf-me-wv3. It is tempting to form a "chain" argument here, which states that there is no difference between optimal-rf, optimal-rk-rf-me and rf-me-wv3, linked by rf-wv3. Demšar (2006) warns against such a conclusion by saying that having an algorithm belong to two different groups is considered "statistical nonsense". This is because implicit within a comparison is the distribution obtained from the difference in the observed data. Therefore, to conclude that rf-wv3 belongs to both groups is equivalent to concluding that rf-wv3 belongs to two different distributions at the same time. Instead, the correct statement regarding rf-wv3 is that *the experimental data is not sufficient to reach any conclusion regarding group membership of rf-wv3* (Demšar, 2006).

Papers focusing on the high-dimensional setting were also analysed using the above two step procedure. That is, the adjusted ranks were computed and the top five algorithms were compared over a collection of maximum overlapping data sets. The results are summarised in Figure 7.11.

The top five high-dimensional algorithms were found to be *rf, xrf, svm-linear, shrunkCent.l* and *grf-rf*. The algorithm xrf is the modification to Forest-RI with input variable weighting, as proposed by Nguyen *et al.* (2015). Svm-linear denotes a support vector machine with a linear kernel, while shrunkCent.l denotes a nearest shrunken centroid method[5], proposed by Tibshirani

[5]Shrunken centroid methods are essentially regularised versions of LDA, see Hastie *et al.* (2009) for details.

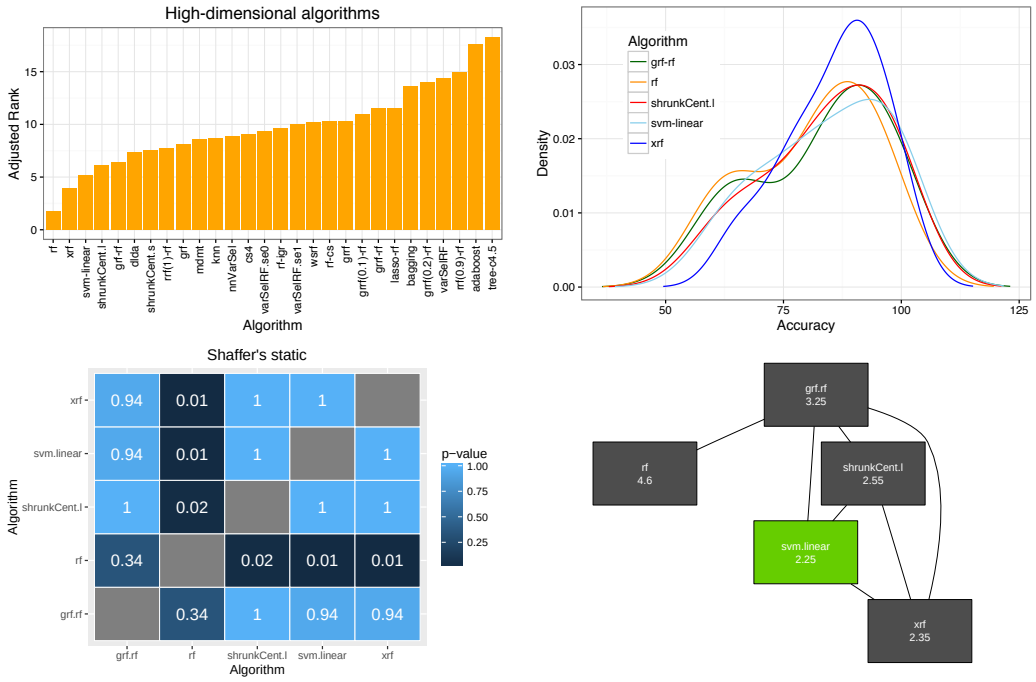

Figure 7.11: Results from comparing the top five high-dimensional algorithms: Top left: *Adjusted ranks.* Top right: *Kernel (Gaussian) density estimates of accuracies.* Bottom left: *Adjusted p-value matrix using the Shaffer static approach.* Bottom right: *Pairwise comparisons plot.*

et al. (2002) and used in Díaz-Uriarte and De Andres (2006). Lastly, the algorithm grf-rf is the guided random forest which uses variable importance scores as weights for input variables subsampled at each node (Deng and Runger, 2013). The overlapping collection of data sets (9 in total) on which the top five algorithms could be compared, included *adenocarcinoma, brain, breast.2.class, breast.3.class, colon, leukemia, lymphoma, nci60, prostate* and *srbct.* The estimated densities presented in the top right of Figure 7.11 show little difference between the algorithms, with xrf perhaps being the exception.

An Iman-Davenport test was performed and resulted in a p-value equal to 0.0014. Therefore, the null hypothesis that there is no difference between the top five high-dimensional algorithms can be rejected. The bottom left of Figure 7.11 provides the adjusted p-value matrix for pairwise comparisons from a Shaffer static test, with the accompanied summary given in the bottom right. No statistically significant pairwise differences were detected between xrf, svm-linear, shrunkCent.l and grf-rf. The only algorithm that did not differ from Forest-RI was grf-rf. However, as was the case with rf-wv3, there is not sufficient evidence to reach a conclusion about the group membership of grf-rf.

7.4.1 Comparing Oblique Random Rotation Forests

In this section, the performance of oblique random rotation forests is compared using 12 data sets from the UCI machine learning repository.[6] The generalisation performance of each algorithm was estimated as follows. Using a 70%/30% split, the data set was randomly divided into training and test data. Using only the training data, ten-fold cross-validation was performed in order to find the optimal tuning parameter values amongst a pre-specified grid. Following this tuning step, the entire training set was then used to train a classifier using the obtained parameter values. The performance of the algorithm was estimated using the test data. This procedure was repeated ten times, with the pre-specified parameter grids identical to those in Section 6.6. The results are given in Table 7.5 in the form of tallied wins/ties for each performance measure estimated. The final column provides the omnibus p-values from a Friedman aligned ranks test, which was conducted for each performance measure. More detailed results for each data set can be found in Appendix C.

Table 7.5: Win/Tie analysis of benchmark performances for random forests.

Measure	**Forest-RI**	**RotF**	**ORF-log**	**ORRotF-log**	p**-val**
Accuracy	**5/1**	1/1	0/1	4/1	0.4779
Sensitivity	2/3	0/2	**4/4**	2/4	0.2480
Specificity	3/1	**3/2**	2/2	0/3	0.3892
Precision	**4/1**	2/2	1/2	1/3	0.4173
Kappa	**5/1**	1/1	0/1	4/1	0.7975
F-score	**4/1**	2/0	1/1	**4/1**	0.8437
H-measure	**5/0**	4/1	0/1	2/1	0.6996
Total	**28/8**	13/9	8/11	17/14	

In Table 7.5, the Friedman aligned ranks test failed to reject the null hypothesis of no difference for all of the performance measures considered. Therefore, although the tallied wins/ties favoured Forest-RI (with ORRotF-log in second place), no statistically significant differences were detected between the performances of the algorithms compared.

7.5 Concluding Remarks

A meta-analysis of all (34) 2001 to 2015 papers that could be found in which a novel random forest algorithm was proposed and compared to already existing random forests was conducted. By evaluating random forest comparative studies and how they adhere to appropriate experimental design and

[6]The data sets used in the comparison were *adult, bank, bank note, breast cancer, german credit, pima, popfailure, saheart, sonar, spam, votes* and *wdbc*. For more details, *cf.* Appendix C.

methodology as is recommended in the literature, false discoveries along with general concerns were identified. These include concerns regarding the performance measures used in a comparison, the way in which these measures were estimated, and the methodology used to compare multiple algorithms over multiple data sets. In fact, even though it has been argued that classification accuracy (or equivalently error) is an inappropriate performance measure, unless the data set on which the performance is measured is well balanced, all of the collected papers in the meta-analysis used accuracy or error as the main measure for comparing algorithms. Furthermore, statistically significant differences were detected between different papers where the same algorithm was tested on the same data set, indicating the degree to which experimental design can influence results. Finally, it is shown that in almost a third of the results from random forest research papers, no significant improvement over the performance of Forest-RI is actually found when comparison are made using appropriate statistical tests.

Using the reported accuracies in each paper, the random forest algorithms found in the literature could be compared. This was done using a novel two step procedure. The first step computed an adjusted rank for each algorithm, with the second step performing appropriate statistical comparisons between the top five ranked algorithms. A particular weighted voting strategy, the so-called *value difference metric* showed considerable promise for improving the performance of Forest-RI. However, Wilson and Martinez (1997) comment on the high computational requirements of this distance metric. As an example, Figure 7.12 shows the time taken to make predictions as a function of the number of test observations (left panel), and the number of input variables (right panel), for a naive R implementation of rf-wv3, when compared to Forest-RI.

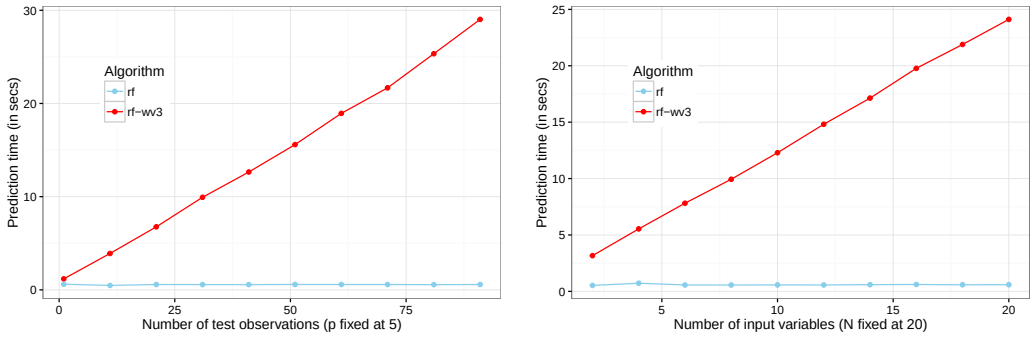

Figure 7.12: Prediction time comparisons between Forest-RI and rf-wv3. Left: *Prediction time as a function of the number of test observations.* Right: *Prediction time for twenty test observations for different sizes of the input space.*

The time differences in Figure 7.12 are stark, even for a small number of ob-

servations and/or input variables. In this regard, a recurring message seems to be that, although classification performance can be measured using several different performance measures, estimated appropriately and compared following the recommended statistical comparison methodology, it ultimately only serves as an evaluation of one aspect of an algorithm. Classification performance can however initiate interest, but an algorithm's computational efficiency, scalability, robustness, stability and level of interpretability remain important for its overall success and widespread adoption.

In high-dimensional scenarios, variable weighting strategies proved successful in improving the performance of Forest-RI. However, compared to other approaches, such as support vector machines and shrunken nearest neighbours methods, no statistically significant differences were detected.

Finally, oblique random rotation forests using logistic regression splits showed no statistically significant improvement over Forest-RI, rotation forests, or oblique random forests using logistic regression splits, when compared on 12 benchmark data sets. However, as previously mentioned, the framework allows for different oblique splitting rules to be applied at each node, of which logistic regression is only a special case. Therefore, other rules remain to be compared and could potentially fair better.

In the next and final chapter, a summary of the findings in the text is provided, along with suggestions for future work.

Chapter 8

Conclusion

This thesis set out to investigate random forests for classification. More specifically, three main focal points were of interest, *viz.* to provide an overview of earlier and later contributions and to conceptualise their connections, to analyse the relative performances of the most important random forest algorithms, and if possible, to investigate heuristically motivated novel random forest proposals. In this final chapter of the thesis, a summary is provided, along with thoughts and interpretations of important aspects of the text. Subsequently, some suggestions are made regarding areas of potential interest in future research on random forests for classification.

8.1 Summary

In **Chapter 2**, classification trees were discussed, together with the CART algorithm for recursive binary partitioning. Trees were shown to be sensitive to changes in the data, which in turn caused them to have high variance. As a way of mitigating the shortcomings of single tree classifiers, ensemble learning schemes were discussed in **Chapter 3**. In order to create diverse ensembles and thereby reducing variance, an ensemble learning algorithm can either rely on deterministic adaptation strategies, or on different ways of inducing randomness into the base classifiers. Random forests, which are discussed in **Chapter 4**, are ensemble learning algorithms that exclusively make use of independently constructed randomised trees as base learners. Their generalisation error is bounded by the strength of each tree in the ensemble, as well as by the correlation between the trees. By increasing the former, and reducing the latter, an improved ensemble classifier can be obtained.

An investigation of random forests from a bias-variance perspective for classification followed in **Chapter 5**. A decomposition of the expected prediction error into bias and variance components is useful when investigating the accuracy of a predictor. However, in classification such decompositions are not

as straightforward as is the case for squared-error loss in regression. Hence various definitions of bias and variance for classification were discussed. By means of an empirical study of bias and variance, and their respective effects on generalisation performance, it was found that the mechanisms employed by random forests, *viz.* randomisation and aggregation, tend to decrease both the variance and its effect. Furthermore, the bias and systematic effect either remained unchanged, or were also reduced.

Chapter 6 provided an overview of the literature regarding novel random forest proposals. A taxonomy of the most important contributions was constructed, by way of which each random forest can be characterised in terms of its source(s) of randomisation and in terms of deterministic modifications used. Using this framework, new modifications and/or combinations of previously explored mechanisms can relatively easily be conceptualised. For the purpose of constructing novel algorithms, the use of bias-variance analyses was proposed as a way to find potentially useful combinations of random forest mechanisms. Following the bias-variance analysis approach, based on the complementary performances of rotation forests and oblique random forests, oblique random rotation forests were proposed. Using logistic regression to obtain the splits, the accuracy of this novel proposal was however not entirely satisfactory.

In **Chapter 7**, a meta-analysis of research on random forests was conducted, followed by a comparative study based on the reported results from each paper. A comparison of multiple algorithms over multiple data sets require sound experimental design and appropriate methodology. By way of a meta-analysis of all the papers that could be found in which a novel random forest algorithm was proposed and compared to already existing random forest algorithms, an evaluation of the state of random forests research was performed. The results revealed comparative studies to predominantly use accuracy to measure algorithm performance. This is despite the fact that the setting in which accuracy is appropriate does not always hold. Furthermore, statistically significant differences were detected between different papers in which the same algorithm was evaluated on the same data set. This points to a possible need for standardisation of experimental designs for comparative studies in classification contexts. It was also shown that no significant improvements over the performance of Forest-RI can actually be found in almost a third of the results from random forests research papers when comparisons are made using appropriate statistical tests.

Using the reported accuracies in each paper, the different random forests could be compared. The approach taken in the text was a two step procedure. In the first step, an adjusted rank was computed for each algorithm, where the rank according to the Friedman method was adjusted for the level of competition

and estimation spread. Once the algorithms were ranked, the top five were compared using appropriate statistical tests. The results showed the value difference metric weighted voting strategy to be particularly useful, however it comes at a high price in terms of computation. In the high-dimensional setting, variable weighting proved useful for Forest-RI, but did not significantly outperform all the other approaches. Finally, oblique random rotation forests using logistic regression splits showed no statistically significant improvement over Forest-RI, rotation forests, or oblique random forests using logistic regression splits, when compared on 12 benchmark data sets.

Ultimately however, even if classification performance can be measured using several different performance measures, estimated appropriately and compared following the recommended statistical comparison methodology, it only serves as an evaluation of one aspect of an algorithm. Impressive classification performance can surely initiate interest in a particular approach, but an algorithm's computational efficiency, scalability, robustness, stability and level of interpretability remain important for its widespread adoption and overall success.

8.2 Avenues for Further Research

The more heuristic-based approach of using a bias-variance analysis to find sensible suggestions for novel classification algorithms, proposed in this thesis, is not limited to random forests. This approach can be applied in the context of any class of algorithms. Also the two-step procedure used to compare algorithms is completely general purpose, given that the papers considered in a meta-analysis all report performances using the same performance measure. Therefore both these analyses can be ported to research on other learning algorithms for classification.

Furthermore, it seems that more insight into research on random forests can still be gained by means of additional mining of the meta-analysis data set, which is publicly available (*cf.* Section 1.2).

Finally, the proposed framework for oblique random rotation forests can further be explored. In this regard, investigation of different rules for node splitting, and the effect that splitting rules have on variable importance measures and on proximity plots may be of interest. With such further evaluations in mind, the *RRotF* package in R has been made available. For more detail the reader is referred to Section 1.2 and Appendix D.1.

Appendices

Appendix A

Bias-Variance Analysis of Oblique Random Rotation Forests

The results for the bias-variance analysis of oblique random rotation forests are given in Table A.1.

Table A.1: Estimated bias, variance, systematic and variance effects for oblique random rotation forests.

Name	Data	Quantity	Forest-RI	ERF	RotF	ORF-log	ORRotF-log
Sim 1	**mvnorm** $p = 15$, $\rho = 0.9$	Error	0.036	0.036	**0.034**	0.035	0.035
		Bayes Error	0.028	0.028	0.028	0.028	0.028
		Systematic Effect	0.003	0.001	**0**	0.003	0.003
		Variance Effect	0.005	0.007	0.006	**0.004**	**0.004**
		Bias	0.005	0.003	**0.002**	0.005	0.005
		Variance	0.016	0.017	0.014	0.012	**0.011**
Sim 2	**mvnorm** $p = 15$, $\rho = 0.5$	Error	0.060	0.058	**0.050**	0.055	0.052
		Bayes Error	0.040	0.040	0.040	0.040	0.040
		Systematic Effect	0.010	0.006	**0.004**	0.009	0.008
		Variance Effect	0.010	0.012	0.006	0.006	**0.004**
		Bias	0.024	0.012	**0.006**	0.015	0.010
		Variance	0.032	0.032	0.019	0.022	**0.018**
Sim 3	**mvnorm** $p = 15$, $\rho = 0.1$	Error	0.126	0.120	0.109	0.107	**0.103**
		Bayes Error	0.078	0.078	0.078	0.078	0.078
		Systematic Effect	0.021	**0.009**	0.011	0.014	0.013
		Variance Effect	0.027	0.033	0.020	0.015	**0.012**
		Bias	0.029	**0.015**	**0.015**	0.024	0.025
		Variance	0.080	0.076	0.057	0.050	**0.044**
Sim 4	**mvnorm** $p = 15$, $\rho = 0$	Error	0.214	0.209	**0.167**	0.176	**0.167**
		Bayes Error	0.141	0.141	0.141	0.141	0.141
		Systematic Effect	0.004	**0**	**0**	0.009	**0**
		Variance Effect	0.069	0.068	0.028	**0.026**	0.032
		Bias	0.044	**0.026**	**0.026**	0.053	0.040
		Variance	0.159	0.159	0.094	0.100	**0.090**

Name	Data	Quantity	Forest-RI	ERF	RotF	ORF-log	ORRotF-log
Sim 5	**Mease-Wyner (2008)** $p = 30$, $J = 2$	Error	0.213	0.201	**0.197**	0.245	0.252
		Bayes Error	0.147	0.147	0.147	0.147	0.147
		Systematic Effect	**0.006**	0.009	0.008	0.041	0.056
		Variance Effect	0.060	0.045	**0.042**	0.057	0.049
		Bias	**0.006**	0.009	0.008	0.065	0.082
		Variance	0.095	0.076	**0.071**	0.130	0.138
Sim 6	**Mease-Wyner (2008)** $p = 30$, $J = 5$	Error	0.272	0.264	**0.244**	0.272	0.270
		Bayes Error	0.143	0.143	0.143	0.143	0.143
		Systematic Effect	0.015	0.017	**0.008**	0.082	0.081
		Variance Effect	0.114	0.104	0.093	0.047	**0.046**
		Bias	0.029	0.029	**0.020**	0.110	0.109
		Variance	0.181	0.170	**0.141**	0.156	0.156
Sim 7	**Mease-Wyner (2008)** $p = 30$, $J = 15$	Error	0.302	0.301	**0.260**	0.262	0.261
		Bayes Error	0.136	0.136	0.136	0.136	0.136
		Systematic Effect	0.031	0.024	**0.014**	0.057	0.059
		Variance Effect	0.135	0.141	0.110	0.069	**0.066**
		Bias	0.037	0.040	**0.022**	0.083	0.077
		Variance	0.226	0.225	0.170	0.158	**0.156**
Sim 8	**Mease-Wyner (2008)** $p = 30$, $J = 20$	Error	0.310	0.306	**0.266**	0.273	0.270
		Bayes Error	0.134	0.134	0.134	0.134	0.134
		Systematic Effect	0.033	0.035	**0.020**	0.083	0.077
		Variance Effect	0.143	0.137	0.112	**0.056**	0.059
		Bias	0.049	0.047	**0.028**	0.109	0.099
		Variance	0.240	0.235	0.180	0.168	**0.166**
Sim 9	**Two -norm** $p = 20$, $K = 2$	Error	0.032	0.030	**0.029**	**0.029**	**0.029**
		Bayes Error	0.024	0.024	0.024	0.024	0.024
		Systematic Effect	**0.001**	0.003	**0.001**	0.003	0.005
		Variance Effect	0.007	0.003	0.004	0.002	**-0.0005**
		Bias	0.013	**0.007**	**0.007**	0.011	0.011
		Variance	0.016	0.016	0.015	0.011	**0.010**
Sim 10	**Three -norm** $p = 20$, $K = 2$	Error	0.156	0.146	**0.145**	0.154	0.156
		Bayes Error	0.085	0.085	0.085	0.085	0.085
		Systematic Effect	0.041	**0.036**	0.040	0.055	0.058
		Variance Effect	0.030	0.025	0.020	0.014	**0.013**
		Bias	0.079	**0.070**	0.078	0.091	0.100
		Variance	0.090	0.084	0.068	0.062	**0.061**
Sim 11	**Ring -norm** $p = 20$, $K = 2$	Error	0.041	**0.034**	0.059	0.051	0.049
		Bayes Error	0.018	0.018	0.018	0.018	0.018
		Systematic Effect	**0.008**	**0.008**	0.012	0.017	0.019
		Variance Effect	0.015	**0.008**	0.029	0.016	0.012
		Bias	0.022	**0.018**	0.026	0.029	0.031
		Variance	0.029	**0.021**	0.044	0.032	0.029

Appendix B

Meta-Analysis

This section contains the details of the data collected for the meta-analysis conducted in Chapter 7.

B.1 Papers Considered

Table B.1 provides the list of papers from which reported results were collected and analysed.

Table B.1: Papers considered in the meta-analysis.

Author(s)	Paper title
Breiman (2001*a*)	Random Forest
Latinne *et al.* (2001)	Limiting the Number of Trees in Random Forests
Cutler and Zhao (2001)	PERT - Perfect Random Tree Ensembles
Robnik-Šikonja (2004)	Improving Random Forests
Geurts *et al.* (2006)	Extremely Randomized Trees
Rodriguez *et al.* (2006)	Rotation Forest
Tsymbal *et al.* (2006)	Dynamic Integration with Random Forests
Lin and Jeon (2006)	Random Forests and Adaptive Nearest Neighbors
Tan and Dowe (2006)	Decision Forests with Oblique Decision Trees
Díaz-Uriarte and De Andres (2006)	Gene Selection and Classification of Microarray Data using Random Forests
Hu *et al.* (2006)	Maximum Diversified Multiple Decision Tree Algorithm for Microarray Data
Bostrom (2007)	Estimating Class Probabilities in Random Forests
Amaratunga *et al.* (2008)	Enriched Random Forests
Boinee *et al.* (2008)	Meta Random Forests
Zhang and Zhang (2008)	RotBoost: A technique for Combining Rotation Forest and AdaBoost
Bernard *et al.* (2009)	On the Selection of Decision Trees in Random Forests
Saffari *et al.* (2009)	On-line Random Forests
Menze *et al.* (2011)	On Oblique Random Forests
Kim *et al.* (2011)	A Weight-adjusted Voting Algorithm for Ensembles of Classifiers
Deng and Runger (2012)	Feature Selection via Regularised Trees
Genuer *et al.* (2010)	Variable Selection using Random Forests
Xu *et al.* (2012)	Classifying Very High-dimensional Data with Random Forests Built from Small Subspaces
Bader-El-Den and Gaber (2012)	GARF: Towards Self-optimised Random Forests
Deng and Runger (2013)	Gene Selection with Guided Regularised Random Forest
Deng (2013)	Guided Random Forest in the RRF package
Tripoliti *et al.* (2013)	Modifications of the Construction and Voting mechanisms of the Random Forest Algorithm
Ye *et al.* (2013)	Stratified Sampling for Feature Subspace Selection in Random Forests for High-dimensional Data
Deng (2014)	Interpreting Tree Ensembles with inTrees
Zhang and Suganthan (2014)	Random Forests with Ensemble of Feature Spaces
Fawagreh *et al.* (2015)	On Extreme Pruning of Random Forest Ensembles
Welbl (2014)	Casting Random Forests as Artificial Neural Networks (and Profiting from it)
Nguyen *et al.* (2015)	Unbiased Feature Selection in Learning Random Forests for High-dimensional Data
Blaser and Fryzlewicz (2015)	Random Rotation Ensembles
Seyedhosseini and Tasdizen (2015)	Disjunctive Normal Random Forests

B.2 Meta-Analysis Data Set

Table B.2 gives a description of each variable in the data set compiled from reported results of random forest comparison studies used in the meta-analysis.

Table B.2: Variables in the meta-analysis data set.

Variable	Description
paper_title	Title of the paper.
author	Author(s) of the paper.
year	Year the paper was published.
journal	Journal the paper appeared in.
dataset	Data set name.
dataset_size	Total size of the data set.
num_inputs	Number of input variables.
classes	Number of classes for the response.
train_size	Size of the training set.
test_size	Size of the test set size.
method	Algorithm used to make predictions.
situation	Focus of the paper, high-dimensional setting or all-round.
error	Reported error rate for the algorithm on the data set.
error_sd	Reported standard deviation of the error.
tuning	Method used to tune the algorithm.
ntree	Ensemble size, if the method is an ensemble method.
mtry	Variable subsample size, if a Random Forests is used.
evaluation	Method used to estimate generalisation performance.
comparison	Method used to compare algorithms.

B.3 Benchmark Data Sets

Table B.3 provides the characteristics of popular benchmark data sets from the UCI machine learning repository.

Table B.3: Characteristics of popular benchmark data sets from the UCI machine learning repository.

Dataset	Size	Number of inputs	Classes
adenocarcinoma	76	9868	2
alzheimers	108	14	2
balance	625	4	3
brain	42	5597	5
breast	106	10	6
breast.2.class	78	4869	2
breast.3.class	96	4869	3
colon	62	2000	2
ecoli	336	8	8
german credit	1000	20	2
glass	214	10	7
hays-roth	160	5	3
hepatitis	155	19	2
ionosphere	351	34	2
iris	150	4	3
letters	20000	16	26
leukemia	38	3051	2

Dataset	Size	Number of inputs	Classes
lymphoma	62	4026	3
mammo-mass	961	5	2
musk	6598	168	2
nci60	61	5244	8
parkinsons	197	23	2
pima	768	8	2
post-opt	90	8	3
prostate	102	6033	2
sonar	208	60	2
spect heart	267	22	2
srbct	63	2308	4
survival	306	3	2
ta-eval	151	5	3
vehicle	946	18	4
votes	435	16	2
vowel	990	10	11
waveform	5000	40	3
wdbc	569	30	2
wine	178	13	3
zoo	101	16	7

B.4 Detail Regarding Algorithms

In Table B.4, a list is provided of all the algorithms in the meta-analysis, as well as the corresponding paper in which the algorithm appeared.

Table B.4: A list of algorithms in the meta-analysis, along with the paper in which each algorithm appeared.

Algorithm	Appeared In
adaboost	Breiman (2001a)
adaboost	Lin and Jeon (2006)
adaboost	Tan and Dowe (2006)
adaboost	Cutler and Zhao (2001)
adaboost	Menze *et al.* (2011)
adaboost	Bader-el-den and Gaber (2012)
adaboost	Hu *et al.* (2006)
adaboost	Zhang and Zhang (2008)
adamenn	Lin and Jeon (2006)
ann	Seyedhosseini and Tasdizen (2015)
ann	Welbl (2014)
ann-rf-relaxed	Welbl (2014)
ann-rf-sparse	Welbl (2014)
ann-rf-vote	Welbl (2014)

Algorithm	**Appeared In**
bag-rs	Latinne *et al.* (2001)
baggedRF	Boinee *et al.* (2008)
bagging	Geurts *et al.* (2006)
bagging	Rodriguez *et al.* (2006)
bagging	Latinne *et al.* (2001)
bagging	Hu *et al.* (2006)
bagging	Kim *et al.* (2011)
bagging	Zhang and Zhang (2008)
bagging-prune	Rodriguez *et al.* (2006)
boostedRF	Boinee *et al.* (2008)
boosting	Rodriguez *et al.* (2006)
boosting	Seyedhosseini and Tasdizen (2015)
boosting	Kim *et al.* (2011)
boosting-prune	Rodriguez *et al.* (2006)
CFS-select	Deng and Runger (2012)
club-drf	Fawagreh *et al.* (2014)
clustering-rf	Tripoliti *et al.* (2013)
cs4	Hu *et al.* (2006)
dann	Lin and Jeon (2006)
dlda	Diaz-Uriarte and Andres (2006)
dndt	Seyedhosseini and Tasdizen (2015)
dnrf	Seyedhosseini and Tasdizen (2015)
enrichRF(chi)	Ye *et al.* (2013)
enrichRF(Ct)	Amaratunga *et al.* (2008)
enrichRF(lda)	Ye *et al.* (2013)
enrichRF(t)	Amaratunga *et al.* (2008)
enrichRFcv(Ct)	Amaratunga *et al.* (2008)
enrichRFcv(t)	Amaratunga *et al.* (2008)
erf	Geurts *et al.* (2006)
erf	Ye *et al.* (2013)
erf	Blaser and Fryzlewicz (2015)
erf-boot	Geurts *et al.* (2006)
FCBF-select	Deng and Runger (2012)
garf	Bader-el-den and Gaber (2012)
gd-MCboost	Seyedhosseini and Tasdizen (2015)
grf	Deng (2013)
grf-rf	Deng (2013)
grrf	Deng (2013)
grrf	Nguyen *et al.* (2015)
grrf-rf	Deng (2013)
grrf(0.1)-rf	Deng and Runger (2013)
grrf(0.2)-rf	Deng and Runger (2013)
inTreeStel	Deng (2014)
knn	Geurts *et al.* (2006)
knn	Diaz-Uriarte and Andres (2006)
knn	Menze *et al.* (2011)
lasso-rf	Deng and Runger (2013)
lasso-rf	Nguyen *et al.* (2015)
lda-rf	Zhang and Suganthan (2014)
logitboost	Lin and Jeon (2006)
mdmt	Hu *et al.* (2006)
mml-rf	Tan and Dowe (2006)
mod-sbs-rf	Tripoliti *et al.* (2013)
mod-sfs-rf	Tripoliti *et al.* (2013)
multiBoost	Zhang and Zhang (2008)
nnVarSel	Diaz-Uriarte and Andres (2006)
no info	Diaz-Uriarte and Andres (2006)

Algorithm	**Appeared In**
online-adaboost	Saffari *et al.* (2009)
online-logitboost	Saffari *et al.* (2009)
online-rf	Saffari *et al.* (2009)
online-savage	Saffari *et al.* (2009)
optimal-rf	Tripoliti *et al.* (2013)
optimal-rf-me	Tripoliti *et al.* (2013)
optimal-rk-rf	Tripoliti *et al.* (2013)
optimal-rk-rf-me	Tripoliti *et al.* (2013)
orf-lda	Menze *et al.* (2011)
orf-log	Seyedhosseini and Tasdizen (2015)
orf-ridge	Menze *et al.* (2011)
orf-ridge	Seyedhosseini and Tasdizen (2015)
orf-rnd	Breiman (2001a)
orf-rnd	Lin and Jeon (2006)
orf-rnd	Menze *et al.* (2011)
orf-rnd	Ye *et al.* (2013)
orf-rnd-k4	Lin and Jeon (2006)
orf-svm	Seyedhosseini and Tasdizen (2015)
pca-rf	Zhang and Suganthan (2014)
pert	Cutler and Zhao (2001)
rboost-select	Deng and Runger (2012)
rf	Breiman (2001a)
rf	Geurts *et al.* (2006)
rf	Deng and Runger (2012)
rf	Deng and Runger (2013)
rf	Rodriguez *et al.* (2006)
rf	Fawagreh *et al.* (2014)
rf	Genuer *et al.* (2012)
rf	Xu *et al.* (2012)
rf	Latinne *et al.* (2001)
rf	Robnik-Sikonja (2004)
rf	Tsymbal *et al.* (2006)
rf	Tan and Dowe (2006)
rf	Cutler and Zhao (2001)
rf	Diaz-Uriarte and Andres (2006)
rf	Amaratunga *et al.* (2008)
rf	Saffari *et al.* (2009)
rf	Boinee *et al.* (2008)
rf	Menze *et al.* (2011)
rf	Bader-el-den and Gaber (2012)
rf	Deng (2013)
rf	Tripoliti *et al.* (2013)
rf	Bostrom (2007)
rf	Hu *et al.* (2006)
rf	Ye *et al.* (2013)
rf	Bernard *et al.* (2008)
rf	Zhang and Suganthan (2014)
rf	Nguyen *et al.* (2015)
rf	Blaser and Fryzlewicz (2015)
rf	Seyedhosseini and Tasdizen (2015)
rf	Welbl (2014)
rf	Kim *et al.* (2011)
rf-5est	Robnik-Sikonja (2004)
rf-cs	Xu *et al.* (2012)
rf-DVSheom	Tsymbal *et al.* (2006)
rf-DVSheomw	Tsymbal *et al.* (2006)
rf-DVSrf	Tsymbal *et al.* (2006)
rf-DVSrfw	Tsymbal *et al.* (2006)
rf-ensemble	Zhang and Suganthan (2014)

Algorithm	Appeared In
rf-igr	Xu *et al.* (2012)
rf-lp	Bostrom (2007)
rf-m1	Bostrom (2007)
rf-m2	Bostrom (2007)
rf-me	Tripoliti *et al.* (2013)
rf-me-wv1	Tripoliti *et al.* (2013)
rf-me-wv3	Tripoliti *et al.* (2013)
rf-mk	Bostrom (2007)
rf-probVote	Bostrom (2007)
rf-pvalFilter	Amaratunga *et al.* (2008)
rf-rcp-k1	Lin and Jeon (2006)
rf-rcp-k4	Lin and Jeon (2006)
rf-reliefF	Tripoliti *et al.* (2013)
rf-rnd	Menze *et al.* (2011)
rf-wv	Robnik-Sikonja (2004)
rf-wv-5est	Robnik-Sikonja (2004)
rf-wv1	Tripoliti *et al.* (2013)
rf-wv2	Tripoliti *et al.* (2013)
rf-wv3	Tripoliti *et al.* (2013)
rf-wv4	Tripoliti *et al.* (2013)
rf-wv5	Tripoliti *et al.* (2013)
rf-wv6	Tripoliti *et al.* (2013)
rk-rf	Tripoliti *et al.* (2013)
rk-rf-me	Tripoliti *et al.* (2013)
rk-rf-me-wv1	Tripoliti *et al.* (2013)
rk-rf-wv1	Tripoliti *et al.* (2013)
rotationForest	Rodriguez *et al.* (2006)
rotationForest	Tripoliti *et al.* (2013)
rotationForest	Seyedhosseini and Tasdizen (2015)
rotationForest	Zhang and Zhang (2008)
rotationForest-prune	Rodriguez *et al.* (2006)
RotBoost	Zhang and Zhang (2008)
rr-erf	Blaser and Fryzlewicz (2015)
rr-rf	Blaser and Fryzlewicz (2015)
rrf	Deng and Runger (2012)
rrf(0.9)-rf	Deng and Runger (2013)
rrf(1)-rf	Deng and Runger (2013)
rs	Geurts *et al.* (2006)
rs	Latinne *et al.* (2001)
sbs-rf	Tripoliti *et al.* (2013)
sbs-rf	Bernard *et al.* (2008)
sfs-rf	Tripoliti *et al.* (2013)
sfs-rf	Bernard *et al.* (2008)
shrunkCent.l	Diaz-Uriarte and Andres (2006)
shrunkCent.s	Diaz-Uriarte and Andres (2006)
soft tree	Seyedhosseini and Tasdizen (2015)
space part	Seyedhosseini and Tasdizen (2015)
srf	Ye *et al.* (2013)
srf(chi)	Ye *et al.* (2013)
srf(lda)	Ye *et al.* (2013)
svm	Bader-el-den and Gaber (2012)
svm-linear	Diaz-Uriarte and Andres (2006)
svm-radial	Menze *et al.* (2011)
svm-radial	Seyedhosseini and Tasdizen (2015)
tree	Geurts *et al.* (2006)
tree	Rodriguez *et al.* (2006)
tree	Seyedhosseini and Tasdizen (2015)
tree-c4.5	Latinne *et al.* (2001)
tree-c4.5	Bader-el-den and Gaber (2012)
tree-c4.5	Hu *et al.* (2006)
tree-cart	Menze *et al.* (2011)
tree-cart	Deng (2014)
tree-cart	Kim *et al.* (2011)
tree-cart	Zhang and Zhang (2008)
tree-prune	Rodriguez *et al.* (2006)

Algorithm	**Appeared In**
varSelRF	Deng and Runger (2013)
varSelRF	Genuer *et al.* (2012)
varSelRF	Nguyen *et al.* (2015)
varSelRF.se0	Diaz-Uriarte and Andres (2006)
varSelRF.se1	Diaz-Uriarte and Andres (2006)
varSelRFboot	Amaratunga *et al.* (2008)
varSelRFcv	Amaratunga *et al.* (2008)
wave	Kim *et al.* (2011)
wsrf	Nguyen *et al.* (2015)
xrf	Nguyen *et al.* (2015)

Appendix C

Benchmark Comparison of Oblique Random Rotation Forests

Table C.1 provides the results for the comparison study of oblique random rotation forests. Performances computed using the *hmeasure* R package.

Table C.1: Results of oblique random rotation forest comparison study

Data set	PerfMeasure	Forest-RI	RotF	ORF-log	ORRotF-log
Adult *N = 32561, p=14*	Accuracy	**0.865**	0.844	0.847	0.849
	Sensitivity	0.628	0.513	**0.725**	0.612
	Specificity	0.940	**0.950**	0.886	0.924
	Precision	**0.770**	0.764	0.668	0.719
	Kappa	**0.607**	0.521	0.594	0.565
	F-score	0.692	0.614	**0.696**	0.661
	H-measure	**0.524**	0.477	0.521	0.510
Bank *N = 45211, p=16*	Accuracy	**0.905**	0.900	0.786	0.903
	Sensitivity	0.409	0.376	**0.830**	0.356
	Specificity	0.970	0.970	**0.780**	0.976
	Precision	0.647	0.623	0.333	**0.663**
	Kappa	**0.454**	0.418	0.370	0.415
	F-score	**0.501**	0.469	0.476	0.463
	H-measure	**0.594**	0.466	0.449	0.550
Bank Note *N = 1372, p=4*	Accuracy	0.993	0.998	**1**	**1**
	Sensitivity	**1**	**1**	**1**	**1**
	Specificity	0.987	0.996	**1**	**1**
	Precision	0.984	0.995	**1**	**1**
	Kappa	0.985	0.995	**1**	**1**
	F-score	0.992	0.997	**1**	**1**
	H-measure	0.995	**1**	**1**	**1**
Breast Cancer *N=683, p=9*	Accuracy	0.956	0.975	0.980	**0.985**
	Sensitivity	0.958	0.972	0.972	**0.986**
	Specificity	0.955	0.977	**0.985**	**0.985**
	Precision	0.919	0.958	**0.972**	**0.972**
	Kappa	0.904	0.946	0.957	**0.968**
	F-score	0.938	0.965	0.972	**0.979**
	H-measure	0.922	0.932	0.959	**0.963**

Data set	PerfMeasure	Forest-RI	RotF	ORF-log	ORRotF-log
German Credit *N=1000, p=61*	Accuracy	0.717	**0.747**	0.730	0.737
	Sensitivity	0.475	0.938	**0.971**	0.962
	Specificity	**0.847**	0.300	0.167	0.211
	Precision	0.623	**0.758**	0.731	0.740
	Kappa	**0.341**	0.283	0.177	0.216
	F-score	0.539	**0.838**	0.834	0.836
	H-measure	0.271	**0.279**	0.264	0.259
Pima *N=768, p=8*	Accuracy	**0.787**	0.752	0.735	0.730
	Sensitivity	**0.933**	0.612	0.700	0.662
	Specificity	0.444	**0.827**	0.753	0.767
	Precision	**0.797**	0.653	0.602	0.602
	Kappa	0.427	**0.446**	0.437	0.419
	F-score	**0.860**	0.632	0.647	0.631
	H-measure	0.322	**0.393**	0.318	0.324
Pop Failure *N=540, p=20*	Accuracy	**0.950**	0.938	0.919	0.944
	Sensitivity	**1**	**1**	**1**	**1**
	Specificity	**0.385**	0.231	0	0.308
	Precision	**0.949**	0.937	0.919	0.943
	Kappa	**0.535**	0.355	0	0.450
	F-score	**0.974**	0.967	0.958	0.970
	H-measure	0.641	**0.696**	0.594	0.658
SA-heart *N=462, p=9*	Accuracy	0.725	0.768	0.775	**0.783**
	Sensitivity	**0.542**	0.458	**0.542**	**0.542**
	Specificity	0.822	**0.933**	0.900	0.911
	Precision	0.619	**0.786**	0.743	0.765
	Kappa	0.375	0.434	0.471	**0.486**
	F-score	0.578	0.579	0.627	**0.634**
	H-measure	0.312	**0.417**	0.365	0.368
Sonar *N=208, p=60*	Accuracy	**0.790**	**0.790**	0.742	0.726
	Sensitivity	0.690	0.690	**0.793**	0.759
	Specificity	**0.879**	**0.879**	0.697	0.697
	Precision	**0.833**	**0.833**	0.697	0.688
	Kappa	**0.574**	**0.574**	0.486	0.453
	F-score	**0.755**	**0.755**	0.742	0.721
	H-measure	**0.692**	0.532	0.547	0.521
Spam *N=4601, p=57*	Accuracy	0.949	0.931	0.939	**0.951**
	Sensitivity	0.908	0.875	0.871	**0.910**
	Specificity	0.975	0.968	**0.983**	0.977
	Precision	0.959	0.946	**0.971**	0.963
	Kappa	0.891	0.854	0.870	**0.896**
	F-score	0.933	0.909	0.918	**0.936**
	H-measure	**0.873**	0.811	0.850	0.871
Votes *N=232, p=16*	Accuracy	**0.986**	0.942	0.928	0.928
	Sensitivity	**1**	0.969	0.938	0.938
	Specificity	**0.973**	0.919	0.919	0.919
	Precision	**0.970**	0.912	0.909	0.909
	Kappa	**0.971**	0.884	0.855	0.855
	F-score	**0.985**	0.939	0.923	0.923
	H-measure	**0.957**	0.830	0.856	0.861
wdbc *N=569, p=31*	Accuracy	0.959	0.971	0.971	**0.976**
	Sensitivity	0.937	0.952	**0.968**	**0.968**
	Specificity	0.972	**0.981**	0.972	**0.981**
	Precision	0.952	**0.968**	0.953	**0.968**
	Kappa	0.911	0.937	0.937	**0.950**
	F-score	0.944	0.960	0.961	**0.968**
	H-measure	0.925	0.912	0.944	**0.956**

Appendix D

Source Code

D.1 Chapter 1 Code: Random Rotation Forest R Package

R Code D.1: Source Code: Random Rotation Forest R Package

```
#####################################################
# Chapter 1: Random Rotation Forest R package
#####################################################

# Check for missing packages and install if missing
list.of.packages <- c("devtools", "caret", "randomForest", "obliqueRF")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()
    [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

# download and load random rotation forests package
if("RRotF" %in% installed.packages()[,"Package"] == FALSE){
      library(devtools)
      # Github profile: Arnu Pretorius
      install_github("arnupretorius/RRotF")
}
library(RRotF)

#####################################
# Random Rotation Forest (RRotF)
#####################################
# The RRotF function extensively makes use of code from the RotationForest
# R package.
# ------------------------------------------------------------------------------
RRotF <- function (x, y, K = round(ncol(x)/3, 0), L = 10, mtry=floor(sqrt(ncol
    (x))), model="log", ...){

      require(randomForest)
      require(obliqueRF)
      x <- data.frame(sapply(x, as.numeric))
      y <- factor(as.numeric(y)-1)
      while (ncol(x)%%K != 0) {
            K <- K - 1
      }
      M <- round(ncol(x)/K)
```

```
37        predicted <- list()
38        fit <- numeric()
39        Ri <- list()
40        Ria <- list()
41        fit <- list()
42        predicted <- matrix(NA, nrow = nrow(x), ncol = L)
43        subsets <- list()
44        SelectedClass <- list()
45        IndependentsClassSubset <- list()
46        IndependentsClassSubsetBoot <- list()
47        pcdata <- list()
48        loadings <- list()
49        for (i in 1:L) {
50              Independents <- x[, sample(1:ncol(x), ncol(x))]
51              n <- 0
52              subsets[[i]] <- list()
53              SelectedClass[[i]] <- list()
54              IndependentsClassSubset[[i]] <- list()
55              IndependentsClassSubsetBoot[[i]] <- list()
56              pcdata[[i]] <- list()
57              loadings[[i]] <- list()
58              for (j in seq(1, K)) {
59                    n <- n + M
60                    subsets[[i]][[j]] <-
61                       data.frame(Independents[, (n-(M - 1)):n], y)
62                    SelectedClass[[i]][[j]] <- as.integer(sample(levels(as.
                          factor(y)), 1))
63                    IndependentsClassSubset[[i]][[j]] <-
64                       subsets[[i]][[j]][subsets[[i]][[j]]$y == SelectedClass[[i
                             ]][[j]], ]
65                    IndependentsClassSubsetBoot[[i]][[j]] <-
66                       IndependentsClassSubset[[i]][[j]][sample(1:dim(
                             IndependentsClassSubset[[i]][[j]])[1],
67                          round(0.75 * nrow(IndependentsClassSubset[[i]][[j]])),
                                replace = TRUE), ]
68                    pcdata[[i]][[j]] <-
69                       princomp(IndependentsClassSubsetBoot[[i]][[j]][,!colnames
                             (IndependentsClassSubsetBoot[[i]][[j]]) %in% "y"])
70                    loadings[[i]][[j]] <- pcdata[[i]][[j]]$loadings[,]
71                    colnames(loadings[[i]][[j]]) <- dimnames(loadings[[i]][[j]])
                          [[1]]
72                    loadings[[i]][[j]] <- data.frame(dimnames(loadings[[i]][[j
                          ]])[[1]],
73                                                     loadings[[i]][[j]])
74                    colnames(loadings[[i]][[j]])[1] <- "rowID"
75              }
76              Ri[[i]] <- Reduce(function(x, y) merge(x, y, by = "rowID",
77                                                     all = TRUE), loadings[[i]])
78              Ri[[i]][is.na(Ri[[i]])] <- 0
79              Ria[[i]] <- Ri[[i]][order(match(Ri[[i]]$rowID, colnames(x))),
80                                  order(match(colnames(Ri[[i]]), colnames(x)))]
81              rownames(Ria[[i]]) <- Ria[[i]]$rowID
82              Ria[[i]]$rowID <- NULL
83              finalx <- data.frame(as.matrix(x) %*% as.matrix(Ria[[i]]))
84              final <- data.frame(finalx, y)
85              if(model=="rf"){
86                    fit[[i]] <- randomForest(y ~ ., data = final, mtry=mtry,
                          ntree=1,
87                                             ...)
88              } else if(model %in% c("ridge", "pls", "log", "svm", "rnd")){
89                    capture.output(fit[[i]] <- obliqueRF(x = as.matrix(finalx),
                          y=as.numeric(y), mtry=mtry, ntree=1,
90                                                         training_method=model,
                                                               verbose = FALSE,
                                                               ...))
91              } else {
```

```
92                  stop("Argument 'model' not a valid model type.")
93              }
94        }
95        res <- list(models = fit, loadings = Ria, columnnames = colnames(x), mod
              =model)
96        class(res) <- "RRotF"
97        res
98  }
99
100 # prediction function
101 predict.RRotF <- function (object, newdata, type="class"){
102       if(class(object) != "RRotF"){
103             stop("Object must be of class 'RRotF'")
104       }
105       newdata <- data.frame(sapply(newdata, as.numeric))
106       if (!identical(colnames(newdata), object$columnnames))
107             stop("Variable names and/or order of variables in newdata is not
                  identical to training set. Please check if variables are
                  exactly the same in both sets.")
108       predicted <- matrix(NA, nrow = nrow(newdata), ncol = length(object$
              models))
109       for (i in 1:length(object$models)) {
110             final <- data.frame(as.matrix(newdata) %*% as.matrix(object$
                  loadings[[i]]))
111             predicted[, i] <- predict(object$models[[i]], final,
112                                       type = "prob")[, 2]
113       }
114       if (type=="class") {
115             ifelse(rowMeans(predicted) > 0.5, 1, 0)
116       }
117       else if(type=="prob") {
118             rowMeans(predicted)
119       } else {
120             stop("Argument 'type' must be either 'class' or 'prob'")
121       }
122 }
123
124 # parameter tuning function
125 findOptimalTuning <- function(x, y, k=10, paraGrid, verboset=TRUE,  ...){
126       CVerrorVec <- NULL
127       nconfig <- nrow(paraGrid)
128       for(i in 1:nconfig){
129             if(verboset){
130                   print(paste("para config",i, "out of", nconfig))
131             }
132             K <- paraGrid[i,1]
133             L <- paraGrid[i,2]
134             mtry <- paraGrid[i,3]
135             CVerrorVec[i] <- kFoldRun(x=x, y=y, k=k, K=K, L=L, mtry=mtry, ...)
                  [[1]]
136       }
137       optParaIndex <- which(CVerrorVec == min(CVerrorVec))[1]
138       list(optTuneVals=paraGrid[optParaIndex,], tuneValErrors=data.frame(
              paraGrid,CVerrorVec))
139 }
140
141 # k-fold cross validation function
142 kFoldRun <- function(x,y,k=10, seed=1, verbose=TRUE, ...){
143       # Test preparations
144       library(caret)
145       foldError <- NULL
146       CVError <- NULL
147
148       # create folds
149       set.seed(seed)
150       folds <- createFolds(y, k=k, list = FALSE)
```

```
151
152         # perform k-fold CV
153         for(i in 1:k){
154                 if(verbose){
155                         print(paste("fold:", i))
156                 }
157                 trainFolds <- folds != i
158                 model <- RRotF(x=x[trainFolds,],y=y[trainFolds], ...)
159                 preds <- predict.RRotF(model, x[!trainFolds,])
160                 foldError[i] <- mean(preds != as.numeric(factor(y[!trainFolds]))
                        -1)
161         }
162         CVError <- mean(foldError)
163         return(list(avgCVError=CVError, perFoldError=foldError))
164 }
165 # ------------------------------------------------------------------------------
```

D.2 Chapter 2 Code: Classification Trees

R Code D.2: Source Code: Classification Trees

```
#####################################
# CHAPTER 2: Classification Trees
#####################################

# Check for missing packages and install if missing
list.of.packages <- c("latex2exp", "MASS", "class", "caret",
                      "ggplot2", "lattice", "rpart", "rpart.plot")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()
    [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

#####################################################
# Figure 2.1: Recursive binary partitioning
#####################################################
# ---------------------------------------------------------------------------
# create empty plot
par(mar=c(0,0,0,0))
plot(0:22, 0:22, type="n", xlab="", ylab="",
     xlim=c(0, 22), ylim=c(-1, 22),
     main="", axes=FALSE)

# draw partitioned input space
# draw box
lines(c(1,1), c(1, 20))
lines(c(1,10), c(1, 1))
lines(c(1,10), c(20, 20))
lines(c(10,10), c(1, 20))
# draw partitions
lines(c(3,3), c(1,20))
lines(c(1,3), c(5,5), col="brown")
lines(c(3,3), c(5,1), col="brown")
lines(c(3,10), c(14, 14))
lines(c(3,7), c(14,14), col="brown")
lines(c(7,7), c(1, 14), col="brown")
lines(c(3,3), c(14,20), col="brown")

# generate random points
set.seed(1)
# R1
points(runif(4, min=1.5, max=2.5), runif(4, min = 1.5, max = 4.5), col="blue")
points(runif(1, min=1.5, max=2.5), runif(1, min = 1.5, max = 4.5), col="orange
    ", pch=2)
# R2
points(runif(12, min=1.5, max=2.5), runif(12, min = 5.5, max = 19.5), col="
    orange", pch=2)
points(runif(3, min=1.5, max=2.5), runif(3, min = 5.5, max = 19.5), col="blue"
    )
# R3
points(runif(17, min=3.5, max=9.5), runif(17, min = 14.5, max = 19.5), col="
    blue")
points(runif(1.5, min=3.5, max=9.5), runif(1, min = 14.5, max = 19.5), col="
    orange", pch=2)
# R4
points(runif(30, min=3.5, max=6.5), runif(30, min = 1.5, max = 13.5), col="
    orange", pch=2)
points(runif(5, min=3.5, max=6.5), runif(5, min = 1.5, max = 13.5), col="blue"
    )
# R5
```

```
54 points(runif(9, min=7.5, max=9.5), runif(9, min = 1.5, max = 13.5), col="blue"
       )
55 points(runif(2, min=7.5, max=9.5), runif(2, min = 1.5, max = 13.5), col="
       orange", pch=2)
56
57 # add text
58 # axis
59 text(0, 10.5, TeX("$X_2$"))
60 text(5.5, -1, TeX("$X_1$"))
61 # split points
62 text(3, -0.1, TeX("$s_1$"))
63 text(0.6, 5, TeX("$s_2$"))
64 text(10.4, 14, TeX("$s_3$"))
65 text(7, -0.1, TeX("$s_4$"))
66 # regions
67 text(2, 2.5, TeX("$R_1$"), col="blue", cex=1.5)
68 text(2, 12, TeX("$R_2$"), col="darkorange", cex=1.5)
69 text(6.5, 17, TeX("$R_3$"), col="blue", cex=1.5)
70 text(5, 7.5, TeX("$R_4$"), col="darkorange", cex=1.5)
71 text(8.5, 7.5, TeX("$R_5$"), col="blue", cex=1.5)
72 # create tree
73 text(16.5, 20, TeX("$X_1 \\leq s_1$"))
74 text(15.5, 18.5, "<< Yes", col="green")
75 text(17.5, 18.5, "No >>", col="red")
76 lines(c(16.5, 16.5), c(18.5, 19.5))
77 lines(c(14, 19),c(19, 19))
78 # splits
79 lines(c(14, 14), c(19, 16))
80 lines(c(19, 19), c(19, 16))
81 # internal nodes
82 text(14, 15, TeX("$X_2 \\leq s_2$"))
83 text(19, 15, TeX("$X_2 \\leq s_3$"))
84 lines(c(14, 14), c(13.5, 14.5))
85 lines(c(13, 15), c(14, 14))
86 lines(c(19, 19), c(13.5, 14.5))
87 lines(c(18, 20), c(14, 14))
88 # split internal node 1
89 lines(c(13, 13), c(14, 9))
90 lines(c(15, 15), c(14, 9))
91 # split internal node 2
92 lines(c(18, 18), c(14, 9))
93 lines(c(20, 20), c(14, 9))
94 # root nodes 1, 2, 3
95 text(13, 8, TeX("$R_1$"), col="blue", cex=1.5)
96 text(15, 8, TeX("$R_2$"), col="darkorange", cex=1.5)
97 text(20, 8, TeX("$R_3$"), col="blue", cex=1.5)
98 # internal node 3
99 text(18, 8, TeX("$X_1 \\leq s_4$"))
100 lines(c(18, 18), c(6.5, 7.5))
101 lines(c(17, 19), c(7, 7))
102 lines(c(17, 17), c(7, 3))
103 lines(c(19, 19), c(7, 3))
104 # root node 4 and 5
105 text(17, 2, TeX("$R_4$"), col="darkorange", cex=1.5)
106 text(19, 2, TeX("$R_5$"), col="blue", cex=1.5)
107
108 # ------------------------------------------------------------------------------
109 ##############################################
110 # Figure 2.2: Simulated mixture data
111 ##############################################
112 # ------------------------------------------------------------------------------
113 # Generate training data
114 set.seed(1)
115 mBlue <- mvrnorm(n=10, mu = c(1,0),Sigma = diag(1,2,2))
116 mOrange <- mvrnorm(n=10, mu = c(0,1),Sigma = diag(1,2,2))
117 B <- matrix(0,nrow=100,ncol=2)
```

```
O <- matrix(0,nrow=100,ncol=2)

for(i in 1:100){
        sample1 = sample(1:10, 1)
        sample2 = sample(1:10, 1)
        meanB = mBlue[sample1,]
        meanO = mOrange[sample2,]
        B[i,] = mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] = mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btrain <- cbind(B[1:100,],matrix(0,100,1))
Otrain <- cbind(O[1:100,],matrix(1,100,1))
datatrain <- rbind(Btrain,Otrain)
Xtrain <- datatrain[,1:2]
Ytrain <- datatrain[,3]
train <- data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])

# create decision boundary plotting grid
x1min <- min(Xtrain[,1])
x1max <- max(Xtrain[,1])
x2min <- min(Xtrain[,2])
x2max <- max(Xtrain[,2])
x1seq <- seq(from=x1min,to=x1max,length=100)
x2seq <- seq(from=x2min,to=x2max,length=100)
plotGrid <- data.frame(as.matrix(expand.grid(x1seq,x2seq)))
colnames(plotGrid) <- colnames(train)[2:3]

# create test set
B <- matrix(0,nrow=5000,ncol=2)
O <- matrix(0,nrow=5000,ncol=2)
for(i in 1:5000){
        sample1 <- sample(1:10, 1)
        sample2 <- sample(1:10, 1)
        meanB <- mBlue[sample1,]
        meanO <- mOrange[sample2,]
        B[i,] <- mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] <- mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btest <- cbind(B[1:5000,],matrix(0,5000,1))
Otest <- cbind(O[1:5000,],matrix(1,5000,1))
datatest <- rbind(Btest,Otest)
Xtest <- datatest[,1:2]
Ytest <- datatest[,3]
test <- data.frame(y=factor(Ytest), X1=Xtest[,1], X2=Xtest[,2])

# plot data
color <- ifelse(train$y == 0, "blue", "darkorange")
# Bayes decision boundary
p <- function(x) {
        s <- sqrt(1/5)
        p0 <- mean(dnorm(x[1], mBlue[,1], s) * dnorm(x[2], mBlue[,2], s))
        p1 <- mean(dnorm(x[1], mOrange[,1], s) * dnorm(x[2], mOrange[,2], s))
        p1/(p0+p1)
}

bayesrule <- apply(plotGrid, 1, p)
bayesPr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
    x1seq)),
                z=as.vector(bayesrule))
gd <- expand.grid(x=x1seq, y=x2seq)
bayesPlot <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])
    , aes(x=X1, y=X2) ) +
      geom_contour(data=bayesPr, aes(x=x, y=y, z=z), breaks=c(0,.5), col="
          purple",
```

```
                    linetype=2)+
      geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
                 col=ifelse(bayesrule<0.5, "skyblue", "orange"))+
      theme_bw()+
      geom_point(size = 3, pch = train$y, col=color) +
      ggtitle("Bayes decision boundary: Mixture data")
bayesPlot
bayesProbs <- apply(test[,2:3], 1, p)
bayesError <- sum(as.numeric(test$y != factor(ifelse(bayesProbs>0.5, 1, 0))))/
    nrow(test)
# ---------------------------------------------------------------------
#####################################################################
# Figure 2.3: Classification tree fitted to the mixture data
#####################################################################
# ---------------------------------------------------------------------
# fit a classification tree to the data
tree.fit <- train(y~., data=train, method="rpart")

# plot fit
prp(tree.fit$finalModel, type=3, varlen=0, faclen=0, fallen.leaves=TRUE, box.
    col=c("orange", "blue"))

# compute training and test error
treeTrainingPreds <- predict(tree.fit)
treeTrainingError <- sum(as.numeric(train$y != treeTrainingPreds))/nrow(train)

# Compute test error
treeTestPreds <- predict(tree.fit, test)
treeTestError <- sum(as.numeric(test$y != treeTestPreds))/nrow(test)

# construct decision boundary plot
treeProbs <- predict(tree.fit, plotGrid, type="prob")[,2]
pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
               z=as.vector(treeProbs))
gd <- expand.grid(x=x1seq, y=x2seq)
gtree <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]),
    aes(x=X1, y=X2)) +
      geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
                 col=ifelse(treeProbs<0.5, "skyblue", "orange")) +
      geom_point(size = 3, pch = train$y, col=color) +
      geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
      geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
          , breaks=c(0,.5)) +
      theme_bw()+
      theme(legend.position="top")+
      scale_color_manual(name="Tree decision boundary:",values=c("purple", "
          brown"),
                         labels = c('Bayes','Tree'))+
      scale_linetype_manual(name = 'Tree decision boundary:', values = c("
          dashed", "solid"),
                          labels = c('Bayes','Tree'))+
    annotate("text", x = 2.2, y = -1.6, size=3,
             label = paste("Training error:", round(treeTrainingError, 3),
                           "\nTest error:", round(treeTestError,3),
                           "\nBayes error:", round(bayesError,3)), hjust=0)
gtree
# ---------------------------------------------------------------------
#####################################################################################
# Figure 2.4: Changes in decision boundary as a result of changes in the data
#####################################################################################
# ---------------------------------------------------------------------
for(i in 1:3){
      # Generate training data
      set.seed(i+2)
      mBlue <- mvrnorm(n=10, mu = c(1,0),Sigma = diag(1,2,2))
```

```
240         mOrange <- mvrnorm(n=10, mu = c(0,1),Sigma = diag(1,2,2))
241         B <- matrix(0,nrow=100,ncol=2)
242         O <- matrix(0,nrow=100,ncol=2)
243
244         for(i in 1:100){
245               sample1 = sample(1:10, 1)
246               sample2 = sample(1:10, 1)
247               meanB = mBlue[sample1,]
248               meanO = mOrange[sample2,]
249               B[i,] = mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
250               O[i,] = mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
251         }
252
253         Btrain <- cbind(B[1:100,],matrix(0,100,1))
254         Otrain <- cbind(O[1:100,],matrix(1,100,1))
255         datatrain <- rbind(Btrain,Otrain)
256         Xtrain <- datatrain[,1:2]
257         Ytrain <- datatrain[,3]
258         trainTemp <- data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])
259
260         # create decision boundary plotting grid
261         x1min <- min(Xtrain[,1])
262         x1max <- max(Xtrain[,1])
263         x2min <- min(Xtrain[,2])
264         x2max <- max(Xtrain[,2])
265         x1seq <- seq(from=x1min,to=x1max,length=100)
266         x2seq <- seq(from=x2min,to=x2max,length=100)
267         plotGrid <- data.frame(as.matrix(expand.grid(x1seq,x2seq)))
268         colnames(plotGrid) <- colnames(trainTemp)[2:3]
269
270         # create test set
271         B <- matrix(0,nrow=5000,ncol=2)
272         O <- matrix(0,nrow=5000,ncol=2)
273         for(i in 1:5000){
274               sample1 <- sample(1:10, 1)
275               sample2 <- sample(1:10, 1)
276               meanB <- mBlue[sample1,]
277               meanO <- mOrange[sample2,]
278               B[i,] <- mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
279               O[i,] <- mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
280         }
281
282         Btest <- cbind(B[1:5000,],matrix(0,5000,1))
283         Otest <- cbind(O[1:5000,],matrix(1,5000,1))
284         datatest <- rbind(Btest,Otest)
285         Xtest <- datatest[,1:2]
286         Ytest <- datatest[,3]
287         testTemp <- data.frame(y=factor(Ytest), X1=Xtest[,1], X2=Xtest[,2])
288
289         # Compute Bayes related quantities
290         color <- ifelse(trainTemp$y == 0, "blue", "darkorange")
291         # Bayes decision boundary
292         p <- function(x) {
293               s <- sqrt(1/5)
294               p0 <- mean(dnorm(x[1], mBlue[,1], s) * dnorm(x[2], mBlue[,2], s))
295               p1 <- mean(dnorm(x[1], mOrange[,1], s) * dnorm(x[2], mOrange[,2],
                        s))
296               p1/(p0+p1)
297         }
298
299         bayesrule <- apply(plotGrid, 1, p)
300         bayesPr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=
                length(x1seq)),
301                             z=as.vector(bayesrule))
302         gd <- expand.grid(x=x1seq, y=x2seq)
303         bayesProbs <- apply(testTemp[,2:3], 1, p)
```

```
304         bayesError <- sum(as.numeric(testTemp$y != factor(ifelse(bayesProbs>0.5,
                1, 0))))/nrow(testTemp)
305
306         # Fit fully grown tree
307         contr <- rpart.control(minsplit = 2, minbucket = 1, cp = 0, maxdepth =
                30)
308         tree.fit <- rpart(y~., data=trainTemp, control=contr)
309
310         # construct decision boundary plot for tree
311         treeProbs <- predict(tree.fit, plotGrid, type="prob")[,2]
312         pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
                x1seq)),
313                        z=as.vector(treeProbs))
314         gd <- expand.grid(x=x1seq, y=x2seq)
315         gtree <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain
                [,2]), aes(x=X1, y=X2)) +
316           geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
317                      col=ifelse(treeProbs<0.5, "skyblue", "orange")) +
318           geom_point(size = 3, pch = trainTemp$y, col=color) +
319           geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown",
                    linetype="dashed"), breaks=c(0,.5))+
320           geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="
                    solid"), breaks=c(0,.5)) +
321           theme_bw()+
322           theme(legend.position="none")+
323           scale_color_manual(name="tree decision boundary:",values=c("purple
                    ", "brown"),
324                              labels = c('Bayes','Tree'))+
325           scale_linetype_manual(name = 'tree decision boundary:', values = c
                    ("dashed", "solid"),
326                                 labels = c('Bayes','Tree'))
327         print(gtree)
328
329         # Fit fully grown treePrune
330         treePrune.fit <- train(y~., data=trainTemp, method="rpart")
331
332         # construct decision boundary plot for treePrune
333         treePruneProbs <- predict(treePrune.fit, plotGrid, type="prob")[,2]
334         pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
                x1seq)),
335                        z=as.vector(treePruneProbs))
336         gd <- expand.grid(x=x1seq, y=x2seq)
337         gtreePrune <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=
                Xtrain[,2]), aes(x=X1, y=X2)) +
338           geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
339                      col=ifelse(treePruneProbs<0.5, "skyblue", "orange")) +
340           geom_point(size = 3, pch = trainTemp$y, col=color) +
341           geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown",
                    linetype="dashed"), breaks=c(0,.5))+
342           geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="
                    solid"), breaks=c(0,.5)) +
343           theme_bw()+
344           theme(legend.position="none")+
345           scale_color_manual(name="treePrune decision boundary:",values=c("
                    purple", "brown"),
346                              labels = c('Bayes','treePrune'))+
347           scale_linetype_manual(name = 'treePrune decision boundary:',
                    values = c("dashed", "solid"),
348                                 labels = c('Bayes','treePrune'))
349         print(gtreePrune)
350
351         # Fit logistic regression model
352         lr.fit <- glm(y ~.,family=binomial(link='logit'),data=trainTemp)
353
354         # construct decision boundary plot for LR
355         LRProbs <- predict(lr.fit, plotGrid, type="response")
```

```
356         pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
                x1seq)),
357                        z=as.vector(LRProbs))
358         gd <- expand.grid(x=x1seq, y=x2seq)
359         gLR <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])
                , aes(x=X1, y=X2)) +
360           geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
361                      col=ifelse(LRProbs<0.5, "skyblue", "orange")) +
362           geom_point(size = 3, pch = trainTemp$y, col=color) +
363           geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown",
                  linetype="dashed"), breaks=c(0,.5))+
364           geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="
                  solid"), breaks=c(0,.5)) +
365           theme_bw()+
366           theme(legend.position="none")+
367           scale_color_manual(name="LR decision boundary:",values=c("purple",
                 "brown"),
368                              labels = c('Bayes','LR'))+
369           scale_linetype_manual(name = 'LR decision boundary:', values = c("
                  dashed", "solid"),
370                                 labels = c('Bayes','LR'))
371         print(gLR)
372 }
373 # ------------------------------------------------------------------------------
```

D.3 Chapter 3 Code: Ensemble Learning for Classification

R Code D.3: Source Code: Ensemble Learning for Classification

```
#######################################################
# CHAPTER 3: Ensemble Learning for Classification
#######################################################

# Check for missing packages and install if missing
list.of.packages <- c("ggplot2", "gridExtra", "rpart", "caret",
                      "MASS", "class", "reshape2", "rpart.plot", "gbm",
                      "survival", "splines", "parallel", "plyr", "lattice",
                      "ipred")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()
    [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

#############################################################################
# Figure 3.1: Improving the accuracy of trees with the AdaBoost algorithm
#############################################################################
# ------------------------------------------------------------------------
# plot contribution curve
Err <- seq(0.5, 0.01, len=100)
alpha <- log((1-Err)/Err)
contributionData <- data.frame(Err=Err, alpha=alpha)
weightUpdateData <- data.frame(Err=Err, weight=exp(alpha)/sum(exp(alpha)))
p1 <- ggplot(contributionData, aes(Err, alpha)) + geom_line(color="blue") +
      geom_point(color="blue")+
      xlab(expression(paste("Error"[b]))) + ylab(expression(paste(alpha[b])))+
      theme_bw()

p2 <- ggplot(weightUpdateData, aes(Err, weight)) + geom_line(color="purple")+
      geom_point(color="purple")+
      xlab(expression(paste("Error"[b]))) + ylab("Misclassified observation
          weight")+
      theme_bw()
grid.arrange(p1, p2, ncol=2)
# ------------------------------------------------------------------------
#############################################################################
# Figure 3.2: Test Error rates on elemStat data for a stump, a fully grown
# tree and for AdaBoost.
#############################################################################
# ------------------------------------------------------------------------
# Simulate elemStat data
set.seed(3)
X <- NULL
for(i in 1:10){
      X <- cbind(X, rnorm(12000))
}
y <- factor(apply(X, 1, function(x){ ifelse(sum(x^2) > 9.34, 1, -1)}))
trainHastie <- data.frame(y=y[1:2000], x=X[1:2000, ])
colnames(trainHastie) <- c("y", paste("X", 1:10, sep=""))
testHastie <- data.frame(y=y[2001:12000], x=X[2001:12000, ])
colnames(testHastie) <- c("y", paste("X", 1:10, sep=""))

# Fit tree stump
fit <- rpart(y~., data=trainHastie, control=rpart.control(maxdepth=1))

# Test error of tree stump
preds <- predict(fit, testHastie, type="class")
```

```
errorStump <- sum(as.numeric(preds != testHastie$y))/length(testHastie$y)

# Fit full tree
fit <- rpart(y~., data=trainHastie)

# Test error of full tree
preds <- predict(fit, testHastie, type="class")
errorFullTree <- sum(as.numeric(preds != testHastie$y))/length(testHastie$y)

# Fit adaBoost model
trainErrorVec <- NULL
testErrorVec <- NULL
expLoss <- NULL
minPlus <- function(x){
      x <- as.numeric(x)
      x <- x-1
      x[x == 0] <- -1
      return(x)
}
fitControl <- trainControl(method = "none")
M <- 600
for(i in 1:M){
      fit <- train(y~., data = trainHastie, method = "gbm", distribution="
          adaboost", trControl = fitControl, verbose = FALSE,
                  tuneGrid = data.frame(interaction.depth = 1,
                                        n.trees = i,
                                        shrinkage = 1,
                                        n.minobsinnode = 20))
      predsTest <- predict(fit, testHastie)
      predsTrain <- predict(fit, trainHastie)
      expLoss[i] <- mean(exp(minPlus(trainHastie$y)*predict(fit$finalModel,
          trainHastie[,-1], n.trees=i)))
      trainErrorVec[i] <- sum(as.numeric(predsTrain != trainHastie$y))/length(
          trainHastie$y)
      testErrorVec[i] <- sum(as.numeric(predsTest != testHastie$y))/length(
          testHastie$y)
}

# Plot errors
TestErrors <- data.frame(x=1:M, y=testErrorVec)
TrainErrors <- data.frame(x=1:M, tr=trainErrorVec, expl=expLoss)
ggplot(TestErrors, aes(x=x, y=y)) + geom_line(color="red") +
      geom_hline(yintercept=errorFullTree, linetype="dashed", color="orange",
          show.legend = TRUE, size=1.2)+
      geom_hline(yintercept=errorStump, linetype="dashed", color="green", show
          .legend = TRUE, size=1.2)+
      ylab("Test error") + xlab("Number of boosting iterations")+
      theme_bw() +
      annotate("text", x = 150, y = 0.425, label = "Stump")+
      annotate("text", x = 300, y = 0.25, label = "Tree")+
      annotate("text", x = 450, y = 0.12, label = "Boosting")
# ------------------------------------------------------------------------------
################################################################################
# Figure 3.3: Top: AdaBoost compared to bagging using 100 classification
# trees fitted to the mixture data. Bottom: A random sample of three
# classification trees from the bagged ensemble
################################################################################
# ------------------------------------------------------------------------------
# Generate training data
set.seed(1)
mBlue <- mvrnorm(n=10, mu = c(1,0),Sigma = diag(1,2,2))
mOrange <- mvrnorm(n=10, mu = c(0,1),Sigma = diag(1,2,2))
B <- matrix(0,nrow=100,ncol=2)
O <- matrix(0,nrow=100,ncol=2)

for(i in 1:100){
```

```
        sample1 = sample(1:10, 1)
        sample2 = sample(1:10, 1)
        meanB = mBlue[sample1,]
        meanO = mOrange[sample2,]
        B[i,] = mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] = mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btrain <- cbind(B[1:100,],matrix(0,100,1))
Otrain <- cbind(O[1:100,],matrix(1,100,1))
datatrain <- rbind(Btrain,Otrain)
Xtrain <- datatrain[,1:2]
Ytrain <- datatrain[,3]
train <- data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])

# create decision boundary plotting grid
x1min <- min(Xtrain[,1])
x1max <- max(Xtrain[,1])
x2min <- min(Xtrain[,2])
x2max <- max(Xtrain[,2])
x1seq <- seq(from=x1min,to=x1max,length=100)
x2seq <- seq(from=x2min,to=x2max,length=100)
plotGrid <- data.frame(as.matrix(expand.grid(x1seq,x2seq)))
colnames(plotGrid) <- colnames(train)[2:3]

# create test set
B <- matrix(0,nrow=5000,ncol=2)
O <- matrix(0,nrow=5000,ncol=2)
for(i in 1:5000){
        sample1 <- sample(1:10, 1)
        sample2 <- sample(1:10, 1)
        meanB <- mBlue[sample1,]
        meanO <- mOrange[sample2,]
        B[i,] <- mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] <- mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btest <- cbind(B[1:5000,],matrix(0,5000,1))
Otest <- cbind(O[1:5000,],matrix(1,5000,1))
datatest <- rbind(Btest,Otest)
Xtest <- datatest[,1:2]
Ytest <- datatest[,3]
test <- data.frame(y=factor(Ytest), X1=Xtest[,1], X2=Xtest[,2])

# fit boosted model
boost.fit <- train(y~., data = train, method = "gbm", distribution="adaboost")
# compute training and test error
boostTrainPreds <- predict(boost.fit)
boostTrainingError <- sum(as.numeric(train$y != boostTrainPreds))/nrow(train)

# Compute test error
boostTestPreds <- predict(boost.fit, test)
boostTestError <- sum(as.numeric(test$y != boostTestPreds))/nrow(test)

# construct decision boundary plot
color <- ifelse(train$y == 0, "blue", "darkorange")
# Bayes decision boundary
p <- function(x) {
        s <- sqrt(1/5)
        p0 <- mean(dnorm(x[1], mBlue[,1], s) * dnorm(x[2], mBlue[,2], s))
        p1 <- mean(dnorm(x[1], mOrange[,1], s) * dnorm(x[2], mOrange[,2], s))
        p1/(p0+p1)
}
bayesrule <- apply(plotGrid, 1, p)
bayesPr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
    x1seq)),
```

```
                    z=as.vector(bayesrule))
bayesProbs <- apply(test[,2:3], 1, p)
bayesError <- sum(as.numeric(test$y != factor(ifelse(bayesProbs>0.5, 1, 0))))/
    nrow(test)
# boosting probabilities
boostProbs <- predict(boost.fit, plotGrid, type="prob")[,2]
pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
               z=as.vector(boostProbs))
gd <- expand.grid(x=x1seq, y=x2seq)
gboost <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]),
    aes(x=X1, y=X2)) +
      geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
                 col=ifelse(boostProbs<0.5, "skyblue", "orange")) +
      geom_point(size = 3, pch = train$y, col=color) +
      geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
      geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
          , breaks=c(0,.5)) +
      theme_bw() +
      theme(legend.position="top")+
      scale_color_manual(name="AdaBoost decision boundary:",values=c("purple",
           "brown"),
                         labels = c('Bayes','AdaBoost'))+
      scale_linetype_manual(name = 'AdaBoost decision boundary:', values = c("
          dashed", "solid"),
                            labels = c('Bayes','AdaBoost'))+
      annotate("text", x = 2.2, y = -1.6, size=3,
               label = paste("Training error:",
                             round(boostTrainingError, 3),
                              "\nTest error:", round(boostTestError,3),
                              "\nBayes error:", round(bayesError,3)), hjust=0)
gboost

# fit bagging to the data
set.seed(11)
bagging.fit <- train(y~., data=train, method="treebag",
                     nbagg=100, control=
                           rpart.control(minsplit=2, cp=0.044444, xval=0))

# plot three randomly selected trees from the bagging model
set.seed(6)
treeIndex <- sample(1:100, 3)
t1 <- bagging.fit$finalModel$mtrees[[treeIndex[1]]]$btree
t2 <- bagging.fit$finalModel$mtrees[[treeIndex[2]]]$btree
t3 <- bagging.fit$finalModel$mtrees[[treeIndex[3]]]$btree
prp(t1, type=3, varlen=0, faclen=0, fallen.leaves=TRUE, box.col=ifelse(t1$
    frame$yval == 1, "blue", "orange"))
prp(t2, type=3, varlen=0, faclen=0, fallen.leaves=TRUE, box.col=ifelse(t2$
    frame$yval == 1, "blue", "orange"))
prp(t3, type=3, varlen=0, faclen=0, fallen.leaves=TRUE, box.col=ifelse(t3$
    frame$yval == 1, "blue", "orange"))

# compute training and test error
baggingPreds <- predict(bagging.fit)
baggingTrainingError <- sum(as.numeric(train$y != baggingPreds))/nrow(test)

# Compute test error
baggingTestPreds <- predict(bagging.fit, test)
baggingTestError <- sum(as.numeric(test$y != baggingTestPreds))/nrow(test)

# construct decision boundary plot
baggingProbs <- predict(bagging.fit, plotGrid, type="prob")[,2]
pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
               z=as.vector(baggingProbs))
gd <- expand.grid(x=x1seq, y=x2seq)
gbagging <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]),
```

```
      aes(x=X1, y=X2)) +
240    geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
241               col=ifelse(baggingProbs<0.5, "skyblue", "orange")) +
242    geom_point(size = 3, pch = train$y, col=color) +
243    geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
           dashed"), breaks=c(0,.5))+
244    geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
           , breaks=c(0,.5)) +
245    theme_bw()+
246    theme(legend.position="top")+
247    scale_color_manual(name="Bagging decision boundary:",values=c("purple",
           "brown"),
248                       labels = c('Bayes','Bagging'))+
249    scale_linetype_manual(name = 'Bagging decision boundary:', values = c("
           dashed", "solid"),
250                          labels = c('Bayes','Bagging'))+
251    annotate("text", x = 2.2, y = -1.6, size=3,
252             label = paste("Training error:",
253                           round(baggingTrainingError, 3),
254                           "\nTest error:", round(baggingTestError,3),
255                           "\nBayes error:", round(bayesError,3)), hjust=0)
256 gbagging
257 # ------------------------------------------------------------------------
```

D.4 Chapter 4 Code: Random Forests

R Code D.4: Source Code: Random Forests

```
##############################
# CHAPTER 4: Random Forests
##############################

# Check for missing packages and install if missing
list.of.packages <- c("gridExtra", "ggplot2", "caret", "reshape2", "kernlab", "randomForest",
                      "randomForestSRC", "ggRandomForests", "latex2exp", "ROCR", "lattice",
                      "grid", "MASS", "ipred", "plyr", "e1071")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

#################################################################################
# Figure 4.1: 10-fold cross-validation errors per additional 10 trees for a
# random forest fit on the mixture data.
#################################################################################
# ------------------------------------------------------------------------------
# Simulate data
set.seed(3)
X <- NULL
for(i in 1:10){
      X <- cbind(X, rnorm(12000))
}
y <- factor(apply(X, 1, function(x){ ifelse(sum(x^2) > 9.34, 1, -1)}))
trainHastie <- data.frame(y=y[1:2000], x=X[1:2000, ])
colnames(trainHastie) <- c("y", paste("X", 1:10, sep=""))
testHastie <- data.frame(y=y[2001:12000], x=X[2001:12000, ])
colnames(testHastie) <- c("y", paste("X", 1:10, sep=""))

# learning curve mixture data with standard error bars
tuneControl <- data.frame(mtry=1)
fitControl <- trainControl(method="cv", number=10)
overError <- NULL
overSD <- NULL
for(i in 1:50){
      rf.fit <- train(y~., data=trainHastie, method="rf", trControl=fitControl,
                      tuneGrid=tuneControl, ntree=i*10)
      overError[i] <- 1-rf.fit$result[2]
      overSD[i] <- rf.fit$result[4]
}

# plot errors with SD for RF fit to the mixture data
fraction <- 1:50*10
errorRF.fit <- unlist(overError)
sdRF.fit <- unlist(overSD)
RF.fiterror <- data.frame(x=fraction, y=errorRF.fit)
RF.fitsdPlus <- data.frame(x=fraction, y=errorRF.fit+sdRF.fit)
RF.fitsdMin <- data.frame(x=fraction, y=errorRF.fit-sdRF.fit)
RFPlot <- ggplot(data=RF.fiterror, aes(x, y)) + geom_line(color="orange", size=1) +
      geom_errorbar(aes(ymax=errorRF.fit+sdRF.fit, ymin=errorRF.fit-sdRF.fit),
            width=5, col="blue")+
      geom_point(col="orange", size=4) + xlab("Number of trees") + ylab("Error (ten-fold CV)") +
      ggtitle("Random Forest Learning: elemStat data")+
```

```
        theme_bw()+
        geom_vline(xintercept=which(errorRF.fit == min(errorRF.fit))*10, col="
            purple", linetype="dashed")
RFPlot
# ------------------------------------------------------------------------------
################################################################################
# Figure 4.2: A Forest-RI fitted to the mixture data: The decision boundary
# is represented by the solid brown line.
################################################################################
# ------------------------------------------------------------------------------
# Generate training data
set.seed(1)
mBlue <- mvrnorm(n=10, mu = c(1,0),Sigma = diag(1,2,2))
mOrange <- mvrnorm(n=10, mu = c(0,1),Sigma = diag(1,2,2))
B <- matrix(0,nrow=100,ncol=2)
O <- matrix(0,nrow=100,ncol=2)

for(i in 1:100){
        sample1 = sample(1:10, 1)
        sample2 = sample(1:10, 1)
        meanB = mBlue[sample1,]
        meanO = mOrange[sample2,]
        B[i,] = mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] = mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btrain <- cbind(B[1:100,],matrix(0,100,1))
Otrain <- cbind(O[1:100,],matrix(1,100,1))
datatrain <- rbind(Btrain,Otrain)
Xtrain <- datatrain[,1:2]
Ytrain <- datatrain[,3]
train <- data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])

# create decision boundary plotting grid
x1min <- min(Xtrain[,1])
x1max <- max(Xtrain[,1])
x2min <- min(Xtrain[,2])
x2max <- max(Xtrain[,2])
x1seq <- seq(from=x1min,to=x1max,length=100)
x2seq <- seq(from=x2min,to=x2max,length=100)
plotGrid <- data.frame(as.matrix(expand.grid(x1seq,x2seq)))
colnames(plotGrid) <- colnames(train)[2:3]

# create test set
B <- matrix(0,nrow=5000,ncol=2)
O <- matrix(0,nrow=5000,ncol=2)
for(i in 1:5000){
        sample1 <- sample(1:10, 1)
        sample2 <- sample(1:10, 1)
        meanB <- mBlue[sample1,]
        meanO <- mOrange[sample2,]
        B[i,] <- mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
        O[i,] <- mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
}

Btest <- cbind(B[1:5000,],matrix(0,5000,1))
Otest <- cbind(O[1:5000,],matrix(1,5000,1))
datatest <- rbind(Btest,Otest)
Xtest <- datatest[,1:2]
Ytest <- datatest[,3]
test <- data.frame(y=factor(Ytest), X1=Xtest[,1], X2=Xtest[,2])

# construct decision boundary plot
color <- ifelse(train$y == 0, "blue", "darkorange")
# Bayes decision boundary
p <- function(x) {
```

```
        s <- sqrt(1/5)
        p0 <- mean(dnorm(x[1], mBlue[,1], s) * dnorm(x[2], mBlue[,2], s))
        p1 <- mean(dnorm(x[1], mOrange[,1], s) * dnorm(x[2], mOrange[,2], s))
        p1/(p0+p1)
}
bayesrule <- apply(plotGrid, 1, p)
bayesPr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
                    z=as.vector(bayesrule))
bayesProbs <- apply(test[,2:3], 1, p)
bayesError <- sum(as.numeric(test$y != factor(ifelse(bayesProbs>0.5, 1, 0))))/nrow(test)

# fit a random forest to the data
fitControl <- trainControl(method="none")
tuneControl <- data.frame(mtry=1)
set.seed(13)
rf.fit <- train(y~., data=train, method="rf", trControl=fitControl,
                tuneGrid=tuneControl, ntree=100, proximity=TRUE)
rfProx <- rf.fit

# compute training and test error
rfTrainPreds <- predict(rf.fit)
rfTrainingError <- sum(as.numeric(train$y != rfTrainPreds))/nrow(train)

# Compute test error
rfTestPreds <- predict(rf.fit, test)
rfTestError <- sum(as.numeric(test$y != rfTestPreds))/nrow(test)

# construct decision boundary plot
rfProbs <- predict(rf.fit, plotGrid, type="prob")[,2]
pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
               z=as.vector(rfProbs))
gd <- expand.grid(x=x1seq, y=x2seq)
grf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]), aes(x=X1, y=X2)) +
  geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
             col=ifelse(rfProbs<0.5, "skyblue", "orange")) +
  geom_point(size = 3, pch = train$y, col=color) +
  geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="dashed"), breaks=c(0,.5))+
  geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid"), breaks=c(0,.5)) +
  theme_bw() +
  theme(legend.position="top")+
  scale_color_manual(name="Forest-RI decision boundary:",values=c("purple", "brown"),
                     labels = c('Bayes','Forest-RI'))+
  scale_linetype_manual(name = 'Forest-RI decision boundary:', values = c("dashed", "solid"),
                        labels = c('Bayes','Forest-RI'))+
  annotate("text", x = 2.2, y = -1.6, size=3,
           label = paste("Training error:", round(rfTrainingError, 3),
                         "\nTest error:", round(rfTestError,3),
                         "\nBayes error:", round(bayesError,3)), hjust=0)
grf

# Simulate elemStat data
set.seed(3)
X <- NULL
for(i in 1:10){
    X <- cbind(X, rnorm(12000))
}
y <- factor(apply(X, 1, function(x){ ifelse(sum(x^2) > 9.34, 1, -1)}))
trainHastie <- data.frame(y=y[1:2000], x=X[1:2000,])
colnames(trainHastie) <- c("y", paste("X", 1:10, sep=""))
```

```
testHastie <- data.frame(y=y[2001:12000], x=X[2001:12000, ])
colnames(testHastie) <- c("y", paste("X", 1:10, sep=""))

# trees learning curves for bagging
M <- 50
trainErrorBag <- NULL
testErrorBag <- NULL
for(i in 1:M){
      fit <- train(y~., data = trainHastie, method = "treebag", trControl =
          fitControl, verbose = FALSE, nbagg=i*10)
      predsTest <- predict(fit, testHastie)
      predsTrain <- predict(fit, trainHastie)
      trainErrorBag[i] <- sum(as.numeric(predsTrain != trainHastie$y))/length(
          trainHastie$y)
      testErrorBag[i] <- sum(as.numeric(predsTest != testHastie$y))/length(
          testHastie$y)
}

# trees learning curves for random forest
trainErrorRF <- NULL
testErrorRF <- NULL
for(i in 1:M){
      fit <- train(y~., data = trainHastie, method = "rf", trControl =
          fitControl, verbose = FALSE,
                  tuneGrid = data.frame(mtry=3), ntree=i*10)
      predsTest <- predict(fit, testHastie)
      predsTrain <- predict(fit, trainHastie)
      trainErrorRF[i] <- sum(as.numeric(predsTrain != trainHastie$y))/length(
          trainHastie$y)
      testErrorRF[i] <- sum(as.numeric(predsTest != testHastie$y))/length(
          testHastie$y)
}

# Plot error curves
errors <- data.frame(x=seq(from=10, to=500, by=10), teBag=testErrorBag, teRF=
    testErrorRF)
mel <- melt(errors, id.var="x")
ggplot(mel, aes(x=x, y=value, col=variable)) + geom_line() +
      geom_point()+
      geom_hline(yintercept=mean(testErrorBag), col="green", linetype="dashed"
          )+
      geom_hline(yintercept=mean(testErrorRF), col="orange", linetype="dashed"
          )+
      theme_bw()+
      theme(legend.position="top")+
      xlab("Number of trees") + ylab("Test Error") +
      scale_colour_manual(name = 'elemStat Data Fit:', values=c("green", "
          orange"), labels = c("Bagging","Forest-RI"))
# ------------------------------------------------------------------------------
################################################################################
# Figure 4.3: OOB error computed on the Spam training data, compared to the
# test error.
################################################################################
# ------------------------------------------------------------------------------
# trees learning curves for random forest
M <- 50
trainErrorRF <- NULL
testErrorRF <- NULL
for(i in 1:M){
      fit <- train(y~., data = trainHastie, method = "rf", trControl =
          fitControl, verbose = FALSE,
                  tuneGrid = data.frame(mtry=3), ntree=i*10)
      predsTest <- predict(fit, testHastie)
      predsTrain <- predict(fit, trainHastie)
      trainErrorRF[i] <- sum(as.numeric(predsTrain != trainHastie$y))/length(
          trainHastie$y)
```

```
233         testErrorRF[i] <- sum(as.numeric(predsTest != testHastie$y))/length(
                testHastie$y)
234 }
235
236 # Compute out of bag error rates for spam data and plot with test error rate
237 fitControl <- trainControl(method="none")
238 rf.fit <- train(y~., data = trainHastie, method = "rf", trControl = fitControl
        , verbose = FALSE,
239                     tuneGrid = data.frame(mtry=3), ntree=500)
240 OOBErrorRF <- rf.fit$finalModel$err.rate[seq(from=10, to=500, by=10),1]
241 OOBTesterrors <- data.frame(x=seq(from=10, to=500, by=10), teRF=testErrorRF,
        oobRF=OOBErrorRF)
242 mel <- melt(OOBTesterrors, id.var="x")
243 ggplot(mel, aes(x=x, y=value, col=variable)) + geom_line() +
244         theme_bw()+
245         geom_point()+
246         theme(legend.position="top")+
247         geom_hline(yintercept=mean(testErrorRF), col="red", linetype="dashed")+
248         geom_hline(yintercept=mean(OOBErrorRF), col="blue", linetype="dashed")+
249         xlab("Number of trees") + ylab("Error") +
250         scale_colour_manual(name = 'OOB vs Test elemStat Data:', values=c("red",
              "blue"), labels = c("Test Error","OOB Error"))
251 # ------------------------------------------------------------------------------
252 ################################################################
253 # Figure 4.4: Variable importance for the spam data
254 ################################################################
255 # ------------------------------------------------------------------------------
256 # load data
257 data(spam)
258 spamData <- data.frame(y=spam$type, spam[,-58])
259
260 # split into training and test
261 set.seed(3)
262 trainIndex <- createDataPartition(spamData$y, p=0.6, list=FALSE)
263 spamTrain <- spamData[trainIndex,]
264 spamTest <- spamData[-trainIndex,]
265
266 # RF variable importance
267 set.seed(123)
268 rf <- rfsrc(y~., data=spamTrain, importance="TRUE")
269
270 # compute prediction error
271 rfPreds <- predict(rf, spamTest, type="prob")
272 rfClassPreds <- rfPreds$class
273 rfMisclassError <- mean(rfClassPreds != spamTest$y)
274
275 # compute variable importance
276 vimp <- gg_vimp(rf, which.outcome="all")
277 plot(vimp) + theme_bw() +theme(legend.position="none")
278 # ------------------------------------------------------------------------------
279 ##############################################################################################
280 # Figure 4.5: Spam data variable exploration plot: The top two rows
281 # correspond to the eight most important variables and the bottom two rows
282 # the least important.
283 ##############################################################################################
284 # ------------------------------------------------------------------------------
285 # plot variable relationships
286 ylabel <- TeX("$\\hat{P}(spam|\\underline{x})$")
287 gg_v <- gg_variable(rf)
288 xvar <- vimp$vars[c(1:3, 55:57)]
289 plot(gg_v, xvar=xvar, panel=TRUE) + scale_colour_manual(values = c("skyblue",
        "red")) +
290         theme_bw()+theme(legend.position="bottom", legend.title=element_blank())
              + ylab(ylabel)
291 # ------------------------------------------------------------------------------
292 ##############################################################################################
```

```
# Table 4.1: Significant predictors from the logistic regression fit to the
# spam data.
###############################################################################
# ------------------------------------------------------------------------------
# fitting a logistic regression model to the spam data
lrSpam <- glm(y ~.,family=binomial(link='logit'),data=spamTrain)

# Compute prediction error
lrPredsProbs <- predict(lrSpam, spamTest, type='response')
lrPreds <- ifelse(lrPredsProbs > 0.5,"spam","nonspam")
LRmisClasificError <- mean(lrPreds != spamTest$y)

# find significant variables
modCoefs <- summary(lrSpam)$coefficients
sigVarIndex <- which(modCoefs[,4] < 0.05)
sigVar <- modCoefs[sigVarIndex,]
sigVarOrd <- round(sigVar[order(sigVar[,4]),], 4)
sigVarOrd
# ------------------------------------------------------------------------------
#################################################################
# In text: Rank correlation between VIMP and p-values
#################################################################
# ------------------------------------------------------------------------------
# rank correlations based on p-value for all predictors
vimpVars <- vimp$vars
lrVars <- sapply(rownames(modCoefs[order(modCoefs[,4]),])[-1], function(x)
    which(vimpVars == x))
rankCorrelation <- cor(1:57, lrVars, method="spearman")
# ------------------------------------------------------------------------------
###############################################################################################
# Figure 4.6: Random Forest partial dependence plot: Left: Partial dependence
# for the word "free". Right: Partial dependence for the word "george".
###############################################################################################
# ------------------------------------------------------------------------------
# rf partial dependence plots
partialFree <- plot.variable(rf, xvar.names="free", partial=TRUE)
partialGeorge <- plot.variable(rf, xvar.names="george", partial=TRUE)
freeData <- gg_partial(partialFree)
georgeData <- gg_partial(partialGeorge)
fplotDat <- data.frame(y = 1-freeData$yhat, x=freeData$free)
gplotDat <- data.frame(y = 1-georgeData$yhat, x=georgeData$george)
parFree <- ggplot(fplotDat, aes(x=x, y=y)) + geom_line() + geom_point(col="
    darkgreen", size=3)+
      theme_bw()+
      ylab(TeX("$\\hat{P}(spam|\\underline{x})$")) + xlab("Percentage of the
          word 'free' in email") +
      geom_rug(sides="b", col="blue")
parGeorge <- ggplot(gplotDat, aes(x=x, y=y)) + geom_line() + geom_point(col="
    darkgreen", size=3)+
      theme_bw()+
      ylab(TeX("$\\hat{P}(spam|\\underline{x})$")) + xlab("Percentage of the
          word 'george' in email") +
      geom_rug(sides="b", col="blue")
grid.arrange(parFree, parGeorge, ncol=2)
# ------------------------------------------------------------------------------
###############################################################################################
# Table 4.2: Model confusion matrices (logistic regression abbreviated as LR)
###############################################################################################
# ------------------------------------------------------------------------------
# helper function to compute probabilities from odds
computeProb <- function(coef){
      odds <- exp(coef)
      prob <- odds/(1+odds)
      return(prob)
}

```

```
354 # compare confusion matrices of the two models
355 rfConfMat <- confusionMatrix(rfClassPreds, spamTest$y, dnn=c("Predicted", "
        Actual"))[[2]]
356 lrConfMat <- confusionMatrix(lrPreds, spamTest$y, dnn=c("Predicted", "Actual")
        )[[2]]
357 # ------------------------------------------------------------------------------
358 ################################################################################
359 # Figure 4.7: ROC curve for a Random Forest and logistic regression fit to
360 # the spam data.
361 ################################################################################
362 # ------------------------------------------------------------------------------
363 # compare using ROC curves
364 # calculating the values for ROC curve
365 pred <- prediction(predictions=data.frame(rf=1-rfPreds$predicted[,1], lr=
        lrPredsProbs),labels=data.frame(rf=as.numeric(spamTest$y),lr=as.numeric(
        spamTest$y)))
366 perf <- performance(pred,"tpr","fpr")
367
368 # plotting the ROC curve
369 plot(perf,colorize=TRUE, main="ROC spam data: RF vs Logistic regression")
370 text(0.01, 0.92, "RF")
371 text(0.15, 0.8, "Logistic regression")
372 # ------------------------------------------------------------------------------
373 ################################################################################
374 # Figure 4.8: Random Forest proximity plots: a comparison of a proximity
375 # plot with RF decision boundary.
376 ################################################################################
377 # ------------------------------------------------------------------------------
378 # proximity plots
379 nrf <- rfProx$finalModel
380 mdsPoints <- MDSplot(nrf, fac=train$y)$points
381 labelPoints <- mdsPoints[c(4, 74, 87, 116, 155, 186),]
382 grfProx1 <- ggplot(data.frame(x=mdsPoints[,1], y=mdsPoints[,2], response=train
        $y), aes(x=x, y=y, col=response))+
383       geom_point()+xlab("Dimension 1") + ylab("Dimension 2") +
384       theme_bw() +
385       theme(legend.position="none")+
386       scale_colour_manual(name="Proximity Plot", values=c("blue", "orange", "
            purple")) +
387       geom_label(data=data.frame(x=labelPoints[,1], y=labelPoints[,2],label=
            paste(1:6)), aes(x=x,y=y,label=label), col="darkgreen", size=5)+
388       ggtitle("Proximity Plot")
389
390 # decision boundary with 2 dimensions
391 labelPoints2 <- train[c(4, 74, 87, 116, 155, 186),2:3]
392 proxRF <- nrf
393 rfProbs <- predict(proxRF, newdata=plotGrid, type="prob")[,2]
394 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
395                z=as.vector(rfProbs))
396 gd <- expand.grid(x=x1seq, y=x2seq)
397 grfProx2 <- ggplot(data.frame(y=factor(Ytrain), x1=Xtrain[,1], x2=Xtrain[,2]),
        aes(x=x1, y=x2)) +
398       geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
399                  col=ifelse(rfProbs<0.5, "skyblue", "orange")) +
400       geom_point(size = 3, pch = train$y, col=color) +
401       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple"), breaks=c(0,.5))
            +
402       theme_bw()+
403       theme(legend.position="none")+
404       scale_color_manual(name="Forest-RI decision boundary:",values=c("brown")
            ,
405                          labels = c(''))+ xlab("X1") + ylab("X2")+
406       geom_label(data=data.frame(x=labelPoints2[,1], y=labelPoints2[,2],label=
            paste(1:6)), aes(x=x,y=y,label=label), col="darkgreen", size=5)+
407       ggtitle("Forest-RI Decision Boundary")
408
```

```
409 # compare proximity plot with decision boundary
410 grid.arrange(grfProx1, grfProx2, ncol=2)
411 # ------------------------------------------------------------------------------
```

D.5 Chapter 5 Code: Bias and Variance in Random Forests

R Code D.5: Source Code: Bias and Variance in Random Forests

```
#####################################################
# CHAPTER 5: BIAS AND VARIANCE IN RANDOM FORESTS
#####################################################

# Check for missing packages and install if missing
list.of.packages <- c("latex2exp", "mlbench", "ggplot2", "caret", "doSNOW", "lattice",
                      "gridExtra", "grid", "stargazer", "ipred", "gbm", "randomForest",
                      "rpart")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

#####################################################
# Figure 5.1: Bias and variance in regression
# ###################################################
# ---------------------------------------------------
# Specifiy means and standard deviation
dmar <- par()$mar
par(mar=c(0,0,0,0))
mean1=65; sd1=15
lb1=80; ub1=120
mean2=100; sd2=25
lb2=80; ub2=120

# simulate data
x1 <- seq(-4,4,length=100)*sd1 + mean1
hx1 <- dnorm(x1,mean1,sd1)
x2 <- seq(-4,4,length=100)*sd2 + mean2
hx2 <- dnorm(x2,mean2,sd2)

# create empty plot
plot(x1, hx1, type="n", xlab="", ylab="",
     xlim=c(0, 200), ylim=c(-0.003, 0.04),
     main="", axes=FALSE)

# Draw distributions
lines(x1, hx1)
lines(x2, hx2)

# make arrows and mean lines
arrows(0, 0, 0, 0.04, xpd = TRUE, length = 0.1)
arrows(0, 0, 200, 0, xpd = TRUE, length = 0.1)
arrows(mean1-sd1, max(hx1), mean1+sd1, max(hx1), length = 0.05, code=3, col="darkorange")
arrows(mean2-sd2, max(hx2), mean2+sd2, max(hx2), length = 0.05, code=3, col="darkorange")
arrows(mean1, 0.029, mean2, 0.029, length = 0.05, code=3, col="darkgreen")
lines(c(mean1,mean1), c(0, 0.035), lty=2, col="blue")
lines(c(mean2, mean2), c(0, 0.035), lty=2, col="blue")

# add text to the plot
proby <- TeX("$P(Y|\\underline{x})$")
probtry <- TeX("$P_{\\Omega_{tr}}(Y|\\underline{x})$")
bayes <- TeX("$Irreducible Error$")
```

```
var <- TeX("$Variance$")
bias <- TeX("$Bias$")
bayesModel <- TeX("$f_B(\\underline{x})$")
avgModel <- TeX("$\\bar{f}(\\underline{x})$")
text(30, 0.005,proby,cex=0.8)
text(150, 0.005, probtry, cex=0.8)
text(35, max(hx1), bayes, cex=0.8)
text(135, max(hx2), var, cex=0.8)
text((mean1+mean2)/2, 0.032, bias, cex=0.8)
text(mean1, -0.0029, bayesModel, cex=0.8)
text(mean2, -0.0029, avgModel, cex=0.8)
text(-6, 0.038, "P")
text(190, -0.0025, "Y")
# ------------------------------------------------------------------------------
################################################################################
# Figure 5.2: Bias and variance of an estimated distribution: Left: Large
# bias and small variance. Right: Small bias and large variance.
################################################################################
# ------------------------------------------------------------------------------
# Large bias, small variance
# Specify means and standard deviations
mean1=85; sd1=15
lb1=80; ub1=120
mean2=140; sd2=5
lb2=80; ub2=120

# simulate data
x1 <- seq(-4,4,length=100)*sd1 + mean1
hx1 <- dnorm(x1,mean1,sd1)
x2 <- seq(-4,4,length=100)*sd2 + mean2
hx2 <- dnorm(x2,mean2,sd2)

# create empty plot
plot(x1, hx1, type="n", xlab="", ylab="",
     xlim=c(0, 200), ylim=c(-0.003, 0.08),
     main="", axes=FALSE)

# draw distributions
lines(x1, hx1)
lines(x2, hx2)
arrows(0, 0, 0, 0.08, xpd = TRUE, length = 0.1)
arrows(0, 0, 200, 0, xpd = TRUE, length = 0.1)
arrows(mean2-sd2, max(hx2), mean2+sd2, max(hx2), length = 0.05, code=3, col="
    darkorange")
arrows(mean1-sd1, max(hx1), mean1+sd1, max(hx1), length = 0.05, code=3, col="
    darkorange")
arrows(mean1, 0.5*max(hx2), mean2, 0.5*max(hx2), length = 0.05, code=3, col="
    darkgreen")
lines(c(mean1,mean1), c(0, 0.08), lty=2, col="blue")
lines(c(mean2, mean2), c(0, 0.08), lty=2, col="blue")

# add text to the plot
proby <- TeX("$P(y|\\underline{x})$")
probtry <- TeX("$P_{\\Omega_{tr}}(y|\\underline{x})$")
bayesModel <- TeX("$f_B(\\underline{x})$")
avgModel <- TeX("$\\bar{f}(\\underline{x})$")
text(65, 0.02,proby,cex=0.9)
text(165, 0.005, probtry, cex=0.9)
text((mean1+mean2)/2, 0.5*max(hx2)+0.003, "Large bias", cex=0.9)
text(mean2+sd2+17, max(hx2), "Small variance", cex=0.9)
text(mean1, -0.0029, bayesModel, cex=0.9)
text(mean2, -0.0029, avgModel, cex=0.9)
text(-5, 0.078, "P")
text(190, -0.0025, "Y")
# Small bias, large variance
# specify means and standard deviations
```

```
mean1=85; sd1=15
lb1=80; ub1=120
mean2=90; sd2=25
lb2=80; ub2=120

# simulate data
x1 <- seq(-4,4,length=100)*sd1 + mean1
hx1 <- dnorm(x1,mean1,sd1)
x2 <- seq(-3,4,length=100)*sd2 + mean2
hx2 <- dnorm(x2,mean2,sd2)

# create empty plot
plot(x1, hx1, type="n", xlab="", ylab="",
     xlim=c(0, 200), ylim=c(-0.003, 0.08),
     main="", axes=FALSE)

# draw distributions
lines(x1, hx1)
lines(x2, hx2)
arrows(0, 0, 0, 0.08, xpd = TRUE, length = 0.1)
arrows(0, 0, 200, 0, xpd = TRUE, length = 0.1)
arrows(mean1-sd1, max(hx1), mean1+sd1, max(hx1), length = 0.05, code=3, col="
    darkorange")
arrows(mean2-sd2, max(hx2), mean2+sd2, max(hx2), length = 0.05, code=3, col="
    darkorange")
arrows(mean1, 0.05, mean2, 0.05, length = 0.05, code=3, col="darkgreen")
lines(c(mean1,mean1), c(0, 0.08), lty=2, col="blue")
lines(c(mean2, mean2), c(0, 0.08), lty=2, col="blue")

# add text
proby <- TeX("$P(y|\\underline{x})$")
probtry <- TeX("$P_{\\Omega_{tr}}(y|\\underline{x})$")
bayesModel <- TeX("$f_B(\\underline{x})$")
avgModel <- TeX("$\\bar{f}(\\underline{x})$")
text(65, 0.02,proby,cex=0.9)
text(140, 0.007, probtry, cex=0.9)
text(mean2+15, 0.05, "Small bias", cex=0.9)
text(mean2+sd2+18, max(hx2), "Large variance", cex=0.9)
text(mean1-2, -0.0029, bayesModel, cex=0.9)
text(mean2+2, -0.0029, avgModel, cex=0.9)
text(-4, 0.078, "P")
text(190, -0.0025, "Y")
# ------------------------------------------------------------------------------
################################################################################
# Figure 5.3: The effect of decreasing the variance of probability estimates
# on classification when f > 0.5 and E(PÎ©T R ) > 0.5.
################################################################################
# ------------------------------------------------------------------------------
# specify means and standard deviations
mean1=720; sd1=15
lb1=700; ub1=720
mean2=220; sd2=40
lb2=200; ub2=220

# simulate data
x1 <- seq(-4,4,length=100)*sd1 + mean1
hx1 <- dnorm(x1,mean1,sd1)
x2 <- seq(-4,4,length=100)*sd2 + mean2
hx2 <- dnorm(x2,mean2,sd2)

# create empty plot
plot(c(x1, x2), c(hx1, hx2), type="n", xlab="", ylab="",
     xlim=c(0, 900), ylim=c(-0.003, 0.04),
     main="", axes=FALSE)

# draw distributions
```

```
lines(x1, hx1)
lines(x2, hx2)
# draw probabilit areas
i <- x1 >= lb1
polygon(c(lb1,x1[i],ub1), c(0,hx1[i],0), col="red")
j <- x2 >= lb2
polygon(c(lb2,x2[j],ub2), c(0,hx2[j],0), col="red")

# make arrows and decision boundary/mean lines
arrows(0, 0, 0, 0.04, xpd = TRUE, length = 0.05)
arrows(500, 0, 500, 0.04, xpd = TRUE, length = 0.05)
arrows(0, 0, 400, 0, xpd = TRUE, length = 0.05)
arrows(500, 0, 900, 0, xpd = TRUE, length = 0.05)
arrows(350, 0.02, 490, 0.02, xpd = TRUE, length = 0.2, lwd=2, col="purple")
arrows(mean1-sd1, max(hx1), mean1+sd1, max(hx1), length = 0.05, code=3, col="
    darkorange")
arrows(310, 0.01, 250, 0.003, length = 0.05)
arrows(mean2-sd2, max(hx2), mean2+sd2, max(hx2), length = 0.05, code=3, col="
    darkorange")
arrows(820, 0.01, 740, 0.003, length = 0.05)
lines(c(200,200), c(0, 0.035), lty=2, col="green")
lines(c(700, 700), c(0, 0.035), lty=2, col="green")
lines(c(mean1, mean1), c(0, 0.03), lty=2, col="blue")
lines(c(mean2, mean2), c(0, 0.03), lty=2, col="blue")

# add text
text(-10, 0.038, "P")
text(490, 0.038, "P")
text(0, -0.002, "0.0", cex=0.8)
text(200, -0.002, "0.5", cex=0.8)
text(400, -0.002, "1.0", cex=0.8)
text(500, -0.002, "0.0", cex=0.8)
text(700, -0.002, "0.5", cex=0.8)
text(900, -0.002, "1.0", cex=0.8)
text(415, 0.025, "Decrease Variance", cex = 0.8)
text(200, 0.037, "Decision threshold", cex=0.7)
text(700, 0.037, "Decision threshold", cex=0.7)
text(mean1+5, 0.032, TeX("$E(P_{\\Omega_{TR}})$"), cex=0.8)
text(mean2+5, 0.032, TeX("$E(P_{\\Omega_{TR}})$"), cex=0.8)
text(mean1+sd1+45, max(hx1), TeX("$Var(P_{\\Omega_{TR}})$"), cex=0.8)
text(mean2-sd2-45, max(hx2), TeX("$Var(P_{\\Omega_{TR}})$"), cex=0.8)
text(310, 0.012, TeX("$P(\\bar{g}(\\underline{x}) = g_B(\\underline{x}))$"),
    cex=0.8)
text(820, 0.012, TeX("$P(\\bar{g}(\\underline{x}) = g_B(\\underline{x}))$"),
    cex=0.8)
# ------------------------------------------------------------------------
################################################################################
# Figure 5.4: The effect of increasing the variance of probability estimates
# on classification when f > 0.5 and E(PÎ©T R ) < 0.5.
################################################################################
# ------------------------------------------------------------------------
# specify means and standard deviations
mean1=180; sd1=15
lb1=200; ub1=220
mean2=680; sd2=40
lb2=700; ub2=720

# simulate data
x1 <- seq(-4,4,length=100)*sd1 + mean1
hx1 <- dnorm(x1,mean1,sd1)
x2 <- seq(-4,4,length=100)*sd2 + mean2
hx2 <- dnorm(x2,mean2,sd2)

# create empty plot
plot(c(x1, x2), c(hx1, hx2), type="n", xlab="", ylab="",
     xlim=c(0, 900), ylim=c(-0.003, 0.04),
```

```
244       main="", axes=FALSE)
245
246 # draw distributions
247 lines(x1, hx1)
248 lines(x2, hx2)
249 # draw probabilit areas
250 i <- x1 >= lb1
251 polygon(c(lb1,x1[i],ub1), c(0,hx1[i],0), col="red")
252 j <- x2 >= lb2
253 polygon(c(lb2,x2[j],ub2), c(0,hx2[j],0), col="red")
254
255 # make arrows and decision boundary, mean lines
256 arrows(0, 0, 0, 0.04, xpd = TRUE, length = 0.05)
257 arrows(500, 0, 500, 0.04, xpd = TRUE, length = 0.05)
258 arrows(0, 0, 400, 0, xpd = TRUE, length = 0.05)
259 arrows(500, 0, 900, 0, xpd = TRUE, length = 0.05)
260 arrows(350, 0.02, 490, 0.02, xpd = TRUE, length = 0.2, lwd=2, col="purple")
261 arrows(mean1-sd1, max(hx1), mean1+sd1, max(hx1), length = 0.05, code=3, col="
        darkorange")
262 arrows(260, 0.01, 208, 0.003, length = 0.05)
263 arrows(mean2-sd2, max(hx2), mean2+sd2, max(hx2), length = 0.05, code=3, col="
        darkorange")
264 arrows(800, 0.005, 720, 0.003, length = 0.05)
265 lines(c(200,200), c(0, 0.035), lty=2, col="green")
266 lines(c(700, 700), c(0, 0.035), lty=2, col="green")
267 lines(c(180, 180), c(0, 0.03), lty=2, col="blue")
268 lines(c(680, 680), c(0, 0.03), lty=2, col="blue")
269
270 # add text
271 text(-10, 0.038, "P")
272 text(490, 0.038, "P")
273 text(0, -0.002, "0.0", cex=0.8)
274 text(200, -0.002, "0.5", cex=0.8)
275 text(400, -0.002, "1.0", cex=0.8)
276 text(500, -0.002, "0.0", cex=0.8)
277 text(700, -0.002, "0.5", cex=0.8)
278 text(900, -0.002, "1.0", cex=0.8)
279 text(415, 0.025, "Increase Variance", cex=0.8)
280 text(200, 0.037, "Decision threshold", cex=0.7)
281 text(700, 0.037, "Decision threshold", cex=0.7)
282 text(179, 0.032, TeX("$E(P_{\\Omega_{TR}})$"), cex=0.8)
283 text(679, 0.032, TeX("$E(P_{\\Omega_{TR}})$"), cex=0.8)
284 text(mean1-sd1-45, max(hx1), TeX("$Var(P_{\\Omega_{TR}})$"), cex=0.8)
285 text(mean2+sd2+45, max(hx2), TeX("$Var(P_{\\Omega_{TR}})$"), cex=0.8)
286 text(270, 0.012, TeX("$P(\\bar{g}(\\underline{x}) = g_B(\\underline{x}))$"),
        cex=0.8)
287 text(800, 0.006, TeX("$P(\\bar{g}(\\underline{x}) = g_B(\\underline{x}))$"),
        cex=0.8)
288 # ------------------------------------------------------------------------------
289 ################################################################################
290 # Figure 5.5: Class distributions for a three class classification task:
291 # Left: The true distribution. Middle: Class distribution over training set
292 # samples for the first classifier. Right: Class distribution over training
293 # set samples for the second classifier.
294 ################################################################################
295 # ------------------------------------------------------------------------------
296 # create empty plot
297 plot(1:12, 1:12, type="n", xlab="", ylab="",
298       xlim=c(0, 12), ylim=c(-0.04, 1),
299       main="", axes=FALSE)
300
301 # add arrows
302 arrows(0, 0, 0, 1, xpd = TRUE, length = 0.1)
303 arrows(0, 0, 12, 0, xpd = TRUE, length = 0.1)
304 # first distribution
305 lines(c(0,1), c(0.6, 0.6), col="darkgreen")
```

```
lines(c(1,1), c(0.6, 0), col="darkgreen")
lines(c(1,2), c(0.3, 0.3), col="darkorange")
lines(c(2,2), c(0.3, 0), col="darkorange")
lines(c(2,3), c(0.1, 0.1), col="skyblue")
lines(c(3,3), c(0.1, 0), col="skyblue")
# second dist
lines(c(4,4), c(0, 0.1), col="darkgreen")
lines(c(4,5), c(0.1, 0.1), col="darkgreen")
lines(c(5,5), c(0, 0.7), col="darkorange")
lines(c(5,6), c(0.7, 0.7), col="darkorange")
lines(c(6,6), c(0.7, 0), col="darkorange")
lines(c(6,7), c(0.2, 0.2), col="skyblue")
lines(c(7,7), c(0.2, 0), col="skyblue")
# third dist
lines(c(8,8), c(0, 0.2), col="darkgreen")
lines(c(8,9), c(0.2, 0.2), col="darkgreen")
lines(c(9,9), c(0, 0.5), col="darkorange")
lines(c(9,10), c(0.5, 0.5), col="darkorange")
lines(c(10,10), c(0.5, 0), col="darkorange")
lines(c(10,11), c(0.3, 0.3), col="skyblue")
lines(c(11,11), c(0.3, 0), col="skyblue")
# place text
text(-0.17, 0.95, "P")
text(11.6, -0.03, "C")
text(0.5, 0.65, "0.6")
text(1.5, 0.35, "0.3")
text(2.5, 0.15, "0.1")
text(4.5, 0.15, "0.1")
text(5.5, 0.75, "0.7")
text(6.5, 0.25, "0.2")
text(8.5, 0.25, "0.2")
text(9.5, 0.55, "0.5")
text(10.5, 0.35, "0.3")
#classes
text(0.5, -0.03, "1")
text(1.5, -0.03, "2")
text(2.5, -0.03, "3")
text(4.5, -0.03, "1")
text(5.5, -0.03, "2")
text(6.5, -0.03, "3")
text(8.5, -0.03, "1")
text(9.5, -0.03, "2")
text(10.5, -0.03, "3")
# math text
text(1.5, 0.85, TeX("$P(C|\\underline{x})$"))
text(5.5, 0.85, TeX("$P^1_{\\Omega_{TR}}$"))
text(9.5, 0.85, TeX("$P^2_{\\Omega_{TR}}$"))
# ------------------------------------------------------------------------------
################################################################################
# Figure 5.6: Class distributions for a three class classification task with
# both estimated distributions having equal variance. The true distribution
# is given on the left.
################################################################################
# ------------------------------------------------------------------------------
# create empty plot
plot(1:12, 1:12, type="n", xlab="", ylab="",
     xlim=c(0, 12), ylim=c(-0.04, 1),
     main="", axes=FALSE)

# add arrows
arrows(0, 0, 0, 1, xpd = TRUE, length = 0.1)
arrows(0, 0, 12, 0, xpd = TRUE, length = 0.1)
# first distribution
lines(c(0,1), c(0.6, 0.6), col="darkgreen")
lines(c(1,1), c(0.6, 0), col="darkgreen")
lines(c(1,2), c(0.3, 0.3), col="darkorange")
```

```
lines(c(2,2), c(0.3, 0), col="darkorange")
lines(c(2,3), c(0.1, 0.1), col="skyblue")
lines(c(3,3), c(0.1, 0), col="skyblue")
# second dist
lines(c(4,4), c(0, 0.3), col="darkgreen")
lines(c(4,5), c(0.3, 0.3), col="darkgreen")
lines(c(5,5), c(0, 0.5), col="darkorange")
lines(c(5,6), c(0.5, 0.5), col="darkorange")
lines(c(6,6), c(0.5, 0), col="darkorange")
lines(c(6,7), c(0.2, 0.2), col="skyblue")
lines(c(7,7), c(0.2, 0), col="skyblue")
# third dist
lines(c(8,8), c(0, 0.2), col="darkgreen")
lines(c(8,9), c(0.2, 0.2), col="darkgreen")
lines(c(9,9), c(0, 0.5), col="darkorange")
lines(c(9,10), c(0.5, 0.5), col="darkorange")
lines(c(10,10), c(0.5, 0), col="darkorange")
lines(c(10,11), c(0.3, 0.3), col="skyblue")
lines(c(11,11), c(0.3, 0), col="skyblue")
# place text
text(-0.17, 0.95, "P")
text(11.6, -0.03, "C")
text(0.5, 0.65, "0.6")
text(1.5, 0.35, "0.3")
text(2.5, 0.15, "0.1")
text(4.5, 0.35, "0.3")
text(5.5, 0.55, "0.5")
text(6.5, 0.25, "0.2")
text(8.5, 0.25, "0.2")
text(9.5, 0.55, "0.5")
text(10.5, 0.35, "0.3")
#classes
text(0.5, -0.03, "1")
text(1.5, -0.03, "2")
text(2.5, -0.03, "3")
text(4.5, -0.03, "1")
text(5.5, -0.03, "2")
text(6.5, -0.03, "3")
text(8.5, -0.03, "1")
text(9.5, -0.03, "2")
text(10.5, -0.03, "3")
# math text
text(1.5, 0.75, TeX("$P(C|\\underline{x})$"))
text(5.5, 0.75, TeX("$P^1_{\\Omega_{TR}}$"))
text(9.5, 0.75, TeX("$P^2_{\\Omega_{TR}}$"))
par(mar=dmar)
# ------------------------------------------------------------------------------
################################################################################
# Figure 5.8: A two-dimensional representation of the simulated data from
# the machine learning benchmark problems found in the mlbench R package.
################################################################################
# ------------------------------------------------------------------------------
# 2dnormals: 2d example of data distribution
example <- mlbench.2dnormals(400, cl=6)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
       theme_bw() + xlab("X1") + ylab("X2") + ggtitle("2dnormals") +
       theme(legend.position="none")

# Twonorm: 2d example of data distribution
example <- mlbench.twonorm(400, d=2)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
       theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Twonorm") +
       theme(legend.position="none")

```

```
# Threenorm: 2d example of data distribution
example <-  mlbench.threenorm(400, d=2)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Threenorm") +
      theme(legend.position="none")

# Ringnorm: 2d example of data distribution
example <-  mlbench.ringnorm(400, d=2)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Ringnorm") +
      theme(legend.position="none")

# Circle: 2d example of data distribution
example <-  mlbench.circle(400, d=2)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Circle") +
      theme(legend.position="none")

# Cassini: 2d example of data distribution
example <-  mlbench.cassini(400)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Cassini") +
      theme(legend.position="none")

# Cuboids: 2d example of data distribution
example <-  mlbench.cuboids(400)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("Cuboids") +
      theme(legend.position="none")

# XOR: 2d example of data distribution
example <- mlbench.xor(400, d=2)
example <- as.data.frame(example)
ggplot(example, aes(x=x.1, y=x.2, col=classes)) + geom_point(size=2) +
      theme_bw() + xlab("X1") + ylab("X2") + ggtitle("XOR") +
      theme(legend.position="none")
# ------------------------------------------------------------------------------
################################################################################
# Table 5.1: Estimated bias, variance, systematic effect and variance effect
# on simulated data.
################################################################################
# ------------------------------------------------------------------------------
majVote <- function(x){names(which.max(table(x)))}
nTrain <- 400
nTest <- 1000
Models <- factor(rep(c("Tree", "Bagging", "Forest-RI", "Boosting"), each=6),
    level=c("Tree", "Bagging", "Forest-RI", "Boosting"))

# performs computations in parallel
cl <- makeCluster(3, type="SOCK")
registerDoSNOW(cl)

# MAIN EXPERIMENT FUNCTIONS
runBiasVarSimulation <- function(trainingSets, simTest, BayesPreds){

    loss <- ifelse(length(levels(simTest$classes)) > 2, "multinomial", "
        adaboost")

    # parameter tuning settings
    fitControl <- trainControl(method = "cv", number = 10)
    treeparaGrid <- expand.grid(cp=seq(0.1, 1, by=0.1))
```

```
502     rfparaGrid <- expand.grid(mtry=seq(1, ncol(simTest)-2, by=2))
503     gbmparaGrid <- expand.grid(n.trees=200, interaction.depth=c(1, 6),
            shrinkage=c(0.01, 0.05, 0.1),
504                                n.minobsinnode=10)
505     # boosting model
506     sim.Boost <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
            simTest,
507                                        method="gbm", paraGrid = gbmparaGrid,
                                               tControl=fitControl,
508                                        BayesPreds=BayesPreds, distribution=
                                               loss ,verbose=FALSE)
509     # single tree model
510     sim.Tree <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
            simTest,
511                                       method="rpart", paraGrid = treeparaGrid,
                                              tControl=fitControl, BayesPreds=
                                             BayesPreds)
512     # bagging model
513     sim.Bag <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
            simTest,
514                                      method="treebag", paraGrid=NULL, tControl
                                          =trainControl(method="none"),
515                                      BayesPreds=BayesPreds, nbagg=200)
516     # random forest model
517     sim.RF <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=simTest
            ,
518                                     method="rf", paraGrid = rfparaGrid,
                                            tControl=fitControl,
519                                     BayesPreds=BayesPreds, ntree=200)
520
521     list(results=rbind(sim.Tree$results, sim.Bag$results, sim.RF$results, sim.
            Boost$results),
522          tuneValues=list(sim.Tree$tuneValues, sim.Bag$tuneValues, sim.RF$
                 tuneValues, sim.Boost$tuneValues))
523 }
524
525
526 simulateBiasVarDecomp <- function(trainingSets, simTest, method, paraGrid,
        tControl, BayesPreds, ...){
527     tuneVals <- paraGrid[1,]
528     numOfExp <- 100
529
530     # train models and make predictions
531     BVpreds <- matrix(0, nrow=numOfExp, ncol=nTest)
532     var.T <- NULL
533     var <- NULL
534     bias <- NULL
535     VE <- NULL
536     SE <- NULL
537     misclassError <- NULL
538     C <- as.numeric(simTest$classes)
539
540     # train models
541     for(j in 1:numOfExp){
542         Model <- train(classes~., data=trainingSets[[j]], method=method,
543                        tuneGrid=paraGrid, trControl=tControl, ...)
544         tuneVals <- rbind(tuneVals, Model$bestTune)
545         BVpreds[j,] <- as.numeric(predict(Model, simTest))
546         print(paste("Method: ", method, ", Iter: ", j, " out of ", numOfExp))
547     }
548
549     # James (2003) decomposition estimates
550     BayesClassifier <- BayesPreds
551     majVoteClassifier <- apply(BVpreds, 2, function(x)majVote(x))
552     var.T <- mean(BayesClassifier != C)
553     var <- mean(apply(BVpreds, 1, function(x) mean(x != majVoteClassifier)))
```

```
    bias <- mean(majVoteClassifier != BayesClassifier)
    VE <- mean(apply(BVpreds, 1, function(x) mean(x != C)) - mean(
        majVoteClassifier != C))
    SE <- mean(majVoteClassifier != C) - mean(BayesClassifier != C)
    meanError <- mean(apply(BVpreds, 1, function(x){ mean(x != C) }))

    # plot bias and variance and systematic effect and variance effect
    vb <- c(meanError, var.T, SE, VE, bias, var)
    bar <- factor(c(1,2,3,4,5,6))
    type <- c("Error", "Bayes Error", "Systematic Effect", "Variance Effect",
        "Bias", "Variance")
    model <- rep(method, 6)
    biasVarPlotData <- data.frame(vb=vb, Decomposition=type, bar=bar, model=
        model)
    list(results=biasVarPlotData, tuneValues=tuneVals[-1,])
}

##########################
# Designed scenarios  #
##########################
# load data generation library
library(pensim)
# simulate data function from "pensim" package
simData <- function (nvars = c(100, 100, 100, 100, 600), cors = c(0.8, 0, 0.8,
    0, 0),
                     associations = c(0.5, 0.5, 0.3, 0.3, 0), firstonly = c(
                         TRUE, FALSE, TRUE, FALSE, FALSE),
                     nsamples = 100, censoring = "none",
                     labelswapprob = 0, response = "timetoevent", basehaz =
                         0.2,
                     logisticintercept = 0)
{
      library(MASS)
      if (labelswapprob < 0)
            stop("labelswapprob cannot be negative")
      if (labelswapprob > 0 & response == "timetoevent")
            stop("labelswapprob is only implemented for binary response")
      if (!class(nvars) %in% c("numeric", "integer"))
            stop("nvars must be a numeric vector")
      if (!class(cors) %in% c("numeric", "integer"))
            stop("cors must be a numeric vector")
      if (class(firstonly) != "logical")
            stop("firstonly must be a logical vector")
      if (!class(associations) %in% c("numeric", "integer"))
            stop("associations must be a numeric vector")
      if (length(nvars) != length(cors) | length(nvars) != length(firstonly) |
          length(nvars) != length(associations))
            stop("nvars, cors, firstonly, and associations must all have the
                same length.")
      x.out <- matrix(0, ncol = sum(nvars), nrow = nsamples)
      definecors <- data.frame(start = c(1, cumsum(nvars[-length(nvars)]) + 1)
          , end = cumsum(nvars), cors = cors, associations = associations,
                               num = nvars, firstonly = firstonly, row.names =
                                   letters[1:length(nvars)])
      Sigma <- matrix(0, ncol = sum(nvars), nrow = sum(nvars))
      wts <- rep(0, sum(nvars))
      for (i in 1:nrow(definecors)) {
            thisrange <- definecors[i, "start"]:definecors[i, "end"]
            Sigma[thisrange, thisrange] <- definecors[i, "cors"]
            diag(Sigma) <- 1
            x.out[, thisrange] <- mvrnorm(n = nsamples, mu = rep(0, nvars[i]),
                 Sigma = Sigma[thisrange, thisrange])
            if (definecors[i, "firstonly"]) {
                  wts[definecors[i, "start"]] <- definecors[i, "associations"]
            }
            else {
```

```
                    wts[definecors[i, "start"]:definecors[i, "end"]] <-
                        definecors[i,"associations"]
            }
            varnames <- paste(letters[i], 1:nvars[i], sep = ".")
            names(wts)[definecors[i, "start"]:definecors[i, "end"]] <-
                varnames
      }
      names(wts) <- make.unique(names(wts))
      dimnames(Sigma) <- list(colnames = names(wts), rownames = names(wts))
      colnames(x.out) <- names(wts)
      betaX <- x.out %*% wts
      x.out <- data.frame(x.out)
      if (identical(response, "timetoevent")) {
            h = basehaz * exp(betaX[, 1])
            x.out$time <- rexp(length(h), h)
            x.out$cens <- 1
            if (class(censoring) == "numeric" | class(censoring) ==
                "integer") {
                  if (length(censoring) == 2) {
                        censtimes <- runif(length(h), min = censoring[1],
                                           max = censoring[2])
                  }
                  else if (length(censoring) == 1) {
                        censtimes <- rep(censoring, length(h))
                  }
                  x.out$cens[x.out$time > censtimes] <- 0
                  x.out$time[x.out$time > censtimes] <- censtimes[x.out$time >
                      censtimes]
            }
      }
      else if (identical(response, "binary")) {
            p <- 1/(1 + exp(-(betaX + logisticintercept)))
            x.out$outcome <- ifelse(p > runif(length(p)), 1, 0)
            if (labelswapprob > 0) {
                  do.swap <- runif(length(p)) < labelswapprob
                  new.outcome <- x.out$outcome
                  new.outcome[x.out$outcome == 1 & do.swap] <- 0
                  new.outcome[x.out$outcome == 0 & do.swap] <- 1
                  x.out$outcome <- new.outcome
            }
            x.out$outcome <- factor(x.out$outcome)
      }
      else stop("response must be either timetoevent or binary")
      return(list(summary = definecors, associations = wts, covariance = Sigma
          ,
                  data = x.out, probs=p))
}
#####################
# SETUP 1: corr=0.9
#####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
```

```
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup1Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup1Results$results$model <- Models
saveRDS(setup1Results, "setup1Results.rda")
ggplot(data=setup1Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
      theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
      theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
          x=element_blank()) +
      guides(fill = guide_legend(title = "Decomposition:"))+
      geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
          "Scenario 1")
#####################
# SETUP 2: corr=0.5
#####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup2Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup2Results$results$model <- Models
saveRDS(setup2Results, "setup2Results.rda")
ggplot(data=setup2Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
      theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
      theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
          x=element_blank()) +
      guides(fill = guide_legend(title = "Decomposition:"))+
      geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
          "Scenario 2")
#####################
# SETUP 3: corr=0.1
#####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
```

```
set.seed(1)
test <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup3Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup3Results$results$model <- Models
saveRDS(setup3Results, "setup3Results.rda")
ggplot(data=setup3Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
      theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
      theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
          x=element_blank()) +
      guides(fill = guide_legend(title = "Decomposition:"))+
      geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
          "Scenario 3")
####################
# SETUP 4: corr=0
####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup4Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup4Results$results$model <- Models
saveRDS(setup4Results, "setup4Results.rda")
ggplot(data=setup4Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
      theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
      theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
          x=element_blank()) +
      guides(fill = guide_legend(title = "Decomposition:"))+
      geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
          "Scenario 4")

# Mease et al. data scenarios
# simulate data function
generateMeasedata <- function(nTrain=400, nTest=1000, Ndata=100, J=2,
    seedStart=1, q = 0.15){

      trainingSetsHD <- list()
      # simulate data
      for(iter in 1:Ndata){
            set.seed(iter+1)
            p <- 30
```

```
791             Xtrain<-matrix(0,nTrain,p)
792             for (i in 1:p){
793                   Xtrain[,i]<-runif(nTrain)
794             }
795             ytrain<-rep(0,nTrain)
796             for (i in 1:nTrain){
797                   ytrain[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtrain[i,1:J]))>(J
                          /2))))
798             }
799             # training data
800             trainingSetsHD[[iter]] <- data.frame(classes=factor(ytrain),
                      Xtrain)
801       }
802       set.seed(1)
803       Xtest<-matrix(0,nTest,p)
804       for (i in 1:p){
805             Xtest[,i]<-runif(nTest)
806       }
807       ytest<-rep(0,nTest)
808       for (i in 1:nTest){
809             ytest[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtest[i,1:J]))>(J/2))))
810       }
811       # training sets and test set data
812       testingSetsHD <- data.frame(classes=factor(ytest), Xtest)
813       list(trainingSetsHD=trainingSetsHD, testingSetsHD=testingSetsHD)
814 }
815
816 ##################
817 # Setup 5: J = 2
818 ##################
819 # simluating training data sets
820 q <- 0.15
821 J <- 2
822 simData1 <- generateMeasedata(J=2)
823 trainingSets <- simData1[[1]]
824 simTest <- simData1[[2]]
825 # run simulation and plot data
826 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
        +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
827 setup5Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
828 setup5Results$results$model <- Models
829 saveRDS(setup5Results, "setup5Results.rda")
830 ggplot(data=setup5Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
        geom_bar(stat="identity", position="identity") +
831       theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
832       theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
              x=element_blank()) +
833       guides(fill = guide_legend(title = "Decomposition:"))+
834       geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
              "Scenario 5")
835
836 ##################
837 # Setup 6: J = 5
838 ##################
839 J <- 5
840 # simluating training data sets
841 simData1 <- generateMeasedata(J=5)
842 trainingSets <- simData1[[1]]
843 simTest <- simData1[[2]]
844 # run simulation and plot data
845 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
        +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
846 setup6Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
847 setup6Results$results$model <- Models
848 saveRDS(setup6Results, "setup6Results.rda")
849 ggplot(data=setup6Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
```

```
    geom_bar(stat="identity", position="identity") +
850       theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
851       theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
              x=element_blank()) +
852       guides(fill = guide_legend(title = "Decomposition:"))+
853       geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
              "Scenario 6")
854
855 ####################
856 # Setup 7: J = 15
857 ####################
858 J <- 15
859 # simluating training data sets
860 simData1 <- generateMeasedata(J=15)
861 trainingSets <- simData1[[1]]
862 simTest <- simData1[[2]]
863 # run simulation and plot data
864 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
    +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
865 setup7Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
866 setup7Results$results$model <- Models
867 saveRDS(setup7Results, "setup7Results.rda")
868 ggplot(data=setup7Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
869       theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
870       theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
              x=element_blank()) +
871       guides(fill = guide_legend(title = "Decomposition:"))+
872       geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
              "Scenario 7")
873
874 ####################
875 # Setup 8: J = 20
876 ####################
877 J <- 20
878 # simluating training data sets
879 simData1 <- generateMeasedata(J=20)
880 trainingSets <- simData1[[1]]
881 simTest <- simData1[[2]]
882 # run simulation and plot data
883 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
    +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
884 setup8Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
885 setup8Results$results$model <- Models
886 saveRDS(setup8Results, "setup8Results.rda")
887 ggplot(data=setup8Results$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
888       theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
889       theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
              x=element_blank()) +
890       guides(fill = guide_legend(title = "Decomposition:"))+
891       geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
              "Scenario 8")
892
893 # MLBENCH DATA
894 ##############################
895 # 2dnormals simulation data
896 ##############################
897 # simluating training data sets
898 trainingSets <- list()
899 for(i in 1:100){
900       set.seed(i+1)
901       train <- mlbench.2dnormals(400, cl=6)
902       train <- as.data.frame(train)
903       trainingSets[[i]] <- train
904 }
```

```
905
906 # simulate test data set
907 set.seed(1)
908 test <- mlbench.2dnormals(1000, cl=6)
909 testFrame <- as.data.frame(test)
910 simTest <- testFrame
911
912 # run simulation and plot data
913 dnormalsResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test
      ))
914 dnormalsResults$results$model <- Models
915 saveRDS(dnormalsResults, "2dnormalsResultsTune.rda")
916 ggplot(data=dnormalsResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
      geom_bar(stat="identity") +
917       theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
918       theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.
            x=element_blank()) +
919       guides(fill = guide_legend(title = "Decomposition:"))+
920       geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
            "2dnormals (2, 6)")
921
922 ############################
923 # twonorm simulation data
924 ############################
925 # simluating training data sets
926 trainingSets <- list()
927 for(i in 1:100){
928     set.seed(i+1)
929     train <- mlbench.twonorm(400, d=20)
930     train <- as.data.frame(train)
931     trainingSets[[i]] <- train
932 }
933
934 # simulate test data set
935 set.seed(1)
936 test <- mlbench.twonorm(1000, d=20)
937 testFrame <- as.data.frame(test)
938 simTest <- testFrame
939
940 # run simulation and plot data
941 twonormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test)
      )
942 twonormResults$results$model <- Models
943 saveRDS(twonormResults, "twonormResultsTune.rda")
944 ggplot(data=twonormResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
      geom_bar(stat="identity") +
945     theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
946     theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.x=
          element_blank()) +
947     guides(fill = guide_legend(title = "Decomposition:"))+
948     geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("
          Twonorm (20, 2)")
949
950 ##############################
951 # threenorm simulation data
952 ##############################
953 # simluating training data sets
954 trainingSets <- list()
955 for(i in 1:100){
956     set.seed(i+1)
957     train <- mlbench.threenorm(400, d=20)
958     train <- as.data.frame(train)
959     trainingSets[[i]] <- train
960 }
961
962 # simulate test data set
```

```
963 set.seed(1)
964 test <- mlbench.threenorm(1000, d=20)
965 testFrame <- as.data.frame(test)
966 simTest <- testFrame
967
968 # run simulation and plot data
969 threenormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(
        test))
970 threenormResults$results$model <- Models
971 saveRDS(threenormResults, "threenormResultsTune.rda")
972 ggplot(data=threenormResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
        geom_bar(stat="identity") +
973     theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
974     theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.x=
            element_blank()) +
975     guides(fill = guide_legend(title = "Decomposition:"))+
976     geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("
            Threenorm (20, 2)")
977
978 ##############################
979 # ringnorm simulation data
980 ##############################
981 # simluating training data sets
982 trainingSets <- list()
983 for(i in 1:100){
984     set.seed(i+1)
985     train <- mlbench.ringnorm(400, d=20)
986     train <- as.data.frame(train)
987     trainingSets[[i]] <- train
988 }
989
990 # simulate test data set
991 set.seed(1)
992 test <- mlbench.ringnorm(1000, d=20)
993 testFrame <- as.data.frame(test)
994 simTest <- testFrame
995
996 # run simulation and plot data
997 ringnormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test
        ))
998 ringnormResults$results$model <- Models
999 saveRDS(ringnormResults, "ringnormResultsTune.rda")
1000 ggplot(data=ringnormResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
        geom_bar(stat="identity", position="identity") +
1001    theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
1002    theme(legend.position = "none", axis.ticks.x=element_blank(), axis.text.x=
            element_blank()) +
1003    guides(fill = guide_legend(title = "Decomposition:"))+
1004    geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("
            Ringnorm (20, 2)")
1005
1006 ############################
1007 # circle simulation data
1008 ############################
1009 # simluating training data sets
1010 trainingSets <- list()
1011 for(i in 1:100){
1012     set.seed(i+1)
1013     train <- mlbench.circle(400, d=20)
1014     train <- as.data.frame(train)
1015     trainingSets[[i]] <- train
1016 }
1017
1018 # simulate test data set
1019 set.seed(1)
1020 test <- mlbench.circle(1000, d=20)
```

```
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
circleResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test))
circleResults$results$model <- Models
saveRDS(circleResults, "circleResultsTune.rda")
ggplot(data=circleResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position = "identity") +
    theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
    theme(legend.position = "bottom", axis.ticks.x=element_blank(), axis.text.
        x=element_blank()) +
    guides(fill = guide_legend(title = "Decomposition:"))+
    geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("
        Circle (20, 2)")

############################
# cassini simulation data
############################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- mlbench.cassini(400)
      train <- as.data.frame(train)
      trainingSets[[i]] <- train
}

# simulate test data set
set.seed(1)
test <- mlbench.cassini(1000)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
cassiniResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test)
    )
cassiniResults$results$model <- Models
saveRDS(cassiniResults, "cassiniResultsTune.rda")
ggplot(data=cassiniResults$results, aes(x=bar, y=vb, fill=Decomposition)) +
    geom_bar(stat="identity", position="identity") +
      theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
      theme(legend.position = "bottom", axis.ticks.x=element_blank(), axis.
          text.x=element_blank()) +
      guides(fill = guide_legend(title = "Decomposition:"))+
      geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle(
          "Cassini (2, 3)")

############################
# cuboids simulation data
############################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
    set.seed(i+1)
    train <- mlbench.cuboids(400)
    train <- as.data.frame(train)
    trainingSets[[i]] <- train[1:400,]
}

# simulate test data set
set.seed(1)
test <- mlbench.cuboids(1000)
testFrame <- as.data.frame(test)
simTest <- testFrame[1:1000,]

```

```
# run simulation and plot data
cuboidsResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test)[1:1000])
cuboidsResults$results$model <- Models
saveRDS(cuboidsResults, "cuboidsResultsTune.rda")
ggplot(data=cuboidsResults$results, aes(x=bar, y=vb, fill=Decomposition)) + geom_bar(stat="identity", position="identity") +
    theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
    theme(legend.position = "bottom", axis.ticks.x=element_blank(), axis.text.x=element_blank()) +
    guides(fill = guide_legend(title = "Decomposition:"))+
    geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("Cuboids (3, 4)")

########################
# xor simulation data
########################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
    set.seed(i+1)
    train <- mlbench.xor(400, d=2)
    train <- as.data.frame(train)
    trainingSets[[i]] <- train
}

# simulate test data set
set.seed(1)
test <- mlbench.xor(1000, d=2)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
xorResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test))
xorResults$results$model <- Models
saveRDS(xorResults, "xorResultsTune.rda")
ggplot(data=xorResults$results, aes(x=bar, y=vb, fill=Decomposition)) + geom_bar(stat="identity", position = "identity") +
    theme_bw() + ylab("Error/Bias+Variance") + xlab("") +
    theme(legend.position = "bottom", axis.ticks.x=element_blank(), axis.text.x=element_blank()) +
    guides(fill = guide_legend(title = "Decomposition:"))+
    geom_hline(yintercept = 0, col="red") + facet_grid(. ~ model) + ggtitle("XOR (2, 2)")

#####################################
# create simulation results table
#####################################
res1 <- readRDS("setup1Results.rda")
res2 <- readRDS("setup2Results.rda")
res3 <- readRDS("setup3Results.rda")
res4 <- readRDS("setup4Results.rda")
res5 <- readRDS("setup5Results.rda")
res6 <- readRDS("setup6Results.rda")
res7 <- readRDS("setup7Results.rda")
res8 <- readRDS("setup8Results.rda")
res9 <- readRDS("2dnormalsResultsTune.rda")
res10 <- readRDS("twonormResultsTune.rda")
res11 <- readRDS("threenormResultsTune.rda")
res12 <- readRDS("ringnormResultsTune.rda")
res13 <- readRDS("circleResultsTune.rda")
res14 <- readRDS("cassiniResultsTune.rda")
res15 <- readRDS("cuboidsResultsTune.rda")
res16 <- readRDS("xorResultsTune.rda")
resList <- list(res1, res2, res3, res4, res5, res6, res7, res8, res9, res10, res11,
```

```
                res12, res13, res14, res15, res16)
tableFinal <- NULL
for(k in 1:length(resList)){
      res <- resList[[k]]
      splitDat <- split(res$results, res$results$model)
      cname <- unique(res$results$model)
      rname <- unique(res$results$Decomposition)
      tableFrame <- matrix(0, nrow=length(rname), ncol=length(cname))
      for(i in 1:length(splitDat)){
            tableFrame[,i] <- splitDat[[i]]$vb
      }
      rownames(tableFrame) <- paste(k, rname)
      colnames(tableFrame) <- cname
      tableFinal <- rbind(tableFinal, tableFrame)
}
tableFinal <- as.data.frame(tableFinal)

# perform statistical tests
n <- nrow(tableFinal)
errorTable <- tableFinal[seq(1, n, by=6),]
SEtable <- tableFinal[seq(3, n, by=6),]
VEtable <- tableFinal[seq(4, n, by=6),]
biasTable <- tableFinal[seq(5, n, by=6),]
varTable <- tableFinal[seq(6, n, by=6),]
compTableList <- list(errorTable, SEtable, VEtable, biasTable, varTable)
compPVals <- list()

# compute omnibus p-vals
library(scmamp)
for(i in 1:length(compTableList)){
      compPVals[[i]] <- friedmanAlignedRanksTest(compTableList[[i]][,-1])
}

# compute post-hoc p-vals
postPVals <- list()
for(i in 1:length(compTableList)){
      postPVals[[i]] <- postHocTest(compTableList[[i]][,-1], test="aligned
          ranks",
                                    correct="shaffer")
}

# create latex table
stargazer(tableFinal, summary = FALSE)
# ------------------------------------------------------------------------
##########################################################################
# In text: Correlation between bias, systematic effect, variance and
# variance effect
##########################################################################
# ------------------------------------------------------------------------
# compute bias, variance, systematic effect and variance effect correlations
SEIndex <- seq(3, 96, by=6)
VEIndex <- seq(4, 96, by=6)
biasIndex <- seq(5, 96, by=6)
varIndex <- seq(6, 96, by=6)
treeCors <- c(cor(tableFinal[biasIndex, 1], tableFinal[SEIndex, 1]),
                  cor(tableFinal[varIndex, 1], tableFinal[VEIndex, 1]))
baggingCors <- c(cor(tableFinal[biasIndex, 2], tableFinal[SEIndex, 2]),
                 cor(tableFinal[varIndex, 2], tableFinal[VEIndex, 2]))
RFCors <- c(cor(tableFinal[biasIndex, 3], tableFinal[SEIndex, 3]),
                 cor(tableFinal[varIndex, 3], tableFinal[VEIndex, 3]))
boostingCors <- c(cor(tableFinal[biasIndex, 4], tableFinal[SEIndex, 4]),
                 cor(tableFinal[varIndex, 4], tableFinal[VEIndex, 4]))

# compute median correlation over the different algorithms
biasSECor <- median(c(treeCors[1], baggingCors[1], RFCors[1], boostingCors[1])
    )
```

```
varVECor <- median(c(treeCors[2], baggingCors[2], RFCors[2], boostingCors[2]))
# ------------------------------------------------------------------------------
################################################################################
# Figure 5.9: Variation in the selection of the optimal subset size of
# randomly selected input variables at each node for Forest-RI over 100
# training sets dis- played for the first eight simulation configurations.
################################################################################
# ------------------------------------------------------------------------------
# plot parameter histogram for each data set and compute the standard
    deviation
# Sim 1
# plot line and bar plot
tuneVals1 <- res1$tuneValues[[3]]
sd1 <- round(sd(tuneVals1), 2)
barData1 <- summary(factor(tuneVals1, levels=sort(unique(tuneVals1))))
g1 <- ggplot(data.frame(mtry=tuneVals1), aes(x=1:100, y=mtry)) + geom_line(col
    ="darkorange") + geom_point() +
      theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData1))), y=barData1),
    aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
      theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 1: mvnorm, p=15,
     corr=0.9; [ SD = ", sd1, " ]")))
# Sim 2
# plot line and bar plot
tuneVals2 <- res2$tuneValues[[3]]
sd2 <- round(sd(tuneVals2), 2)
barData2 <- summary(factor(tuneVals2, levels=sort(unique(tuneVals2))))
g1 <- ggplot(data.frame(mtry=tuneVals2), aes(x=1:100, y=mtry)) + geom_line(col
    ="darkorange") + geom_point() +
      theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData2))), y=barData2),
    aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
      theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 2: mvnorm, p=15,
     corr=0.5; [ SD = ", sd2, " ]")))
# Sim 3
# plot line and bar plot
tuneVals3 <- res3$tuneValues[[3]]
sd3 <- round(sd(tuneVals3), 2)
barData3 <- summary(factor(tuneVals3, levels=sort(unique(tuneVals3))))
g1 <- ggplot(data.frame(mtry=tuneVals3), aes(x=1:100, y=mtry)) + geom_line(col
    ="darkorange") + geom_point() +
      theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData3))), y=barData3),
    aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
      theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 3: mvnorm, p=15,
     corr=0.1; [ SD = ", sd3, " ]")))
# Sim 4
# plot line and bar plot
tuneVals4 <- res4$tuneValues[[3]]
sd4 <- round(sd(tuneVals4), 2)
barData4 <- summary(factor(tuneVals4, levels=sort(unique(tuneVals4))))
g1 <- ggplot(data.frame(mtry=tuneVals4), aes(x=1:100, y=mtry)) + geom_line(col
    ="darkorange") + geom_point() +
      theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData4))), y=barData4),
    aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
      theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 4: mvnorm, p=15,
     corr=0; [ SD = ", sd4, " ]")))
# Sim 5
# plot line and bar plot
tuneVals5 <- res5$tuneValues[[3]]
sd5 <- round(sd(tuneVals5), 2)
```

```
barData5 <- summary(factor(tuneVals5, levels=sort(unique(tuneVals5))))
g1 <- ggplot(data.frame(mtry=tuneVals5), aes(x=1:100, y=mtry)) + geom_line(col="darkorange") + geom_point() +
  theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData5))), y=barData5), aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
  theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 5: Mease (2008), p=30, J=2; [ SD = ", sd5, " ]")))
# Sim 6
# plot line and bar plot
tuneVals6 <- res6$tuneValues[[3]]
sd6 <- round(sd(tuneVals6), 2)
barData6 <- summary(factor(tuneVals6, levels=sort(unique(tuneVals6))))
g1 <- ggplot(data.frame(mtry=tuneVals6), aes(x=1:100, y=mtry)) + geom_line(col="darkorange") + geom_point() +
  theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData6))), y=barData6), aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
  theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 6: Mease (2008), p=30, J=5; [ SD = ", sd6, " ]")))
# Sim 7
# plot line and bar plot
tuneVals7 <- res7$tuneValues[[3]]
sd7 <- round(sd(tuneVals7), 2)
barData7 <- summary(factor(tuneVals7, levels=sort(unique(tuneVals7))))
g1 <- ggplot(data.frame(mtry=tuneVals7), aes(x=1:100, y=mtry)) + geom_line(col="darkorange") + geom_point() +
  theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData7))), y=barData7), aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
  theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 7: Mease (2008), p=30, J=15; [ SD = ", sd7, " ]")))
# Sim 8
# plot line and bar plot
tuneVals8 <- res8$tuneValues[[3]]
sd8 <- round(sd(tuneVals8), 2)
barData8 <- summary(factor(tuneVals8, levels=sort(unique(tuneVals8))))
g1 <- ggplot(data.frame(mtry=tuneVals8), aes(x=1:100, y=mtry)) + geom_line(col="darkorange") + geom_point() +
  theme_bw() + ylab("Variable subsample size") + xlab("Training set")
g2 <- ggplot(data.frame(x=factor(as.numeric(names(barData8))), y=barData8), aes(x=x, y=y)) + geom_bar(stat="identity", fill="skyblue") +
  theme_bw() + xlab("Variable subsample size") + ylab("Frequency")
grid.arrange(g1,g2, ncol=2, top = textGrob(label = paste("Sim 8: Mease (2008), p=30, J=20; [ SD = ", sd8, " ]")))
# ------------------------------------------------------------------------------
```

D.6 Chapter 6 Code: Random Forest Algorithms

R Code D.6: Source Code: Random Forest Algorithms

```
#########################################
# CHAPTER 6: Random Forest Algorithms
#########################################

# Check for missing packages and install if missing
list.of.packages <- c("latex2exp", "mlbench", "ggplot2", "caret", "doSNOW", "lattice",
                      "obliqueRF", "MASS", "pensim", "stargazer", "e1071", "mda",
                      "class", "pls", "ROCR", "snow", "gplots", "extraTrees", "RRF",
                      "wsrf", "rotationForest", "randomForest")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

# download and load random rotation forests package
if("RRotF" %in% installed.packages()[,"Package"] == FALSE){
      library(devtools)
      install_github("arnupretorius/RRotF")
}
library(RRotF)

#########################################################################################
# Figure 6.2: Performance of Forest-RI as a function of noise /
# Figure 6.7: Comparing the performance of Forest-RI with WSRF as a function of noise.
#########################################################################################
# ---------------------------------------------------------------------------------------
# perform simulation
# iterated 50 times
# Some of the code is taken from http://www.davemease.com/contraryevidence/code1.txt
SimErrorsRF <- list(set1=NULL, set2=NULL, set3=NULL, set4=NULL, set5=NULL)
SimErrorsWSRF <- list(set1=NULL, set2=NULL, set3=NULL, set4=NULL, set5=NULL)
vars <- c(12, 52, 102, 202, 1002)
J <- 2
q <- 0.15
nTrain <- 400
nTest <- 1000
for(v in 1:length(vars)){
      for(iter in 1:50){
            set.seed(iter)
            p <- vars[v]
            Xtrain<-matrix(0,nTrain,p)
            Xtest<-matrix(0,nTest,p)
            for (i in 1:p){
                  Xtrain[,i]<-runif(nTrain)
                  Xtest[,i]<-runif(nTest)
            }
            ytrain<-rep(0,nTrain)
            for (i in 1:nTrain){
                  ytrain[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtrain[i,1:J]))>(J/2))))
            }
            ytest<-rep(0,nTest)
```

```
54              for (i in 1:nTest){
55                    ytest[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtest[i,1:J]))>(J/
                          2))))
56              }
57              # training data
58              train <- data.frame(y=factor(ytrain), Xtrain)
59              # test data
60              test <- data.frame(y=factor(ytest), Xtest)
61              # Compute errors of rf models
62              fitControl <- trainControl(method="none")
63              tuneControl <- data.frame(mtry=floor(sqrt(vars[v])))
64              rf.fit <- train(y~., data=train, method="rf", trControl=fitControl
                    ,
65                              tuneGrid=tuneControl, ntree=100)
66              wsrf.fit <- train(y~., data=train, method="wsrf", trControl=
                    fitControl,
67                              tuneGrid=tuneControl, ntree=100)
68              SimErrorsRF[[v]][iter] <- mean(test$y != predict(rf.fit, test
                    [,-1]))
69              SimErrorsWSRF[[v]][iter] <- mean(test$y != predict(wsrf.fit, test
                    [,-1]))
70        }
71  }
72  # plot results
73  resultsRF <- data.frame(set=rep(c("set1", "set2", "set3", "set4", "set5"),
        each=50),
74                          error=c(SimErrorsRF[[1]], SimErrorsRF[[2]],SimErrorsRF
                                [[3]],SimErrorsRF[[4]],SimErrorsRF[[5]]))
75  resultsWSRF <- data.frame(set=rep(c("set1", "set2", "set3", "set4", "set5"),
        each=50),
76                            error=c(SimErrorsWSRF[[1]], SimErrorsWSRF[[2]],
                                  SimErrorsWSRF[[3]],
77                                      SimErrorsWSRF[[4]],SimErrorsWSRF[[5]]))
78  # comupte relevant variables sampling probabilities
79  subSampleProbs <- sapply(vars, function(x){
80        mtry <- floor(sqrt(x))
81        round((2*choose(x-2, mtry-1) + choose(x-2, mtry-2))/choose(x,mtry), 2)
82  } )
83  # plot boxplots for random forest (forest-RI)
84  ggplot(resultsRF, aes(y=error, x=set)) + stat_boxplot(geom ='errorbar', width
        =0.5) +
85        geom_boxplot(notch = TRUE, fill="darkgreen", outlier.color = "red") +
86        geom_hline(yintercept = q, linetype="dashed", col="purple") +
87        scale_x_discrete(labels=c("(2, 10)", "(2, 50)", "(2, 100)", "(2, 200)",
              "(2, 1000)"))+
88        theme_bw() + xlab("Number of (relevant, noise) variables") + ylim(0,
              0.75) +
89        ylab("Test Misclassification Error") + annotate("text",x=1, y=0.15+q,
              label=subSampleProbs[1])+
90        annotate("text",x=2, y=0.19+q, label=subSampleProbs[2])+
91        annotate("text",x=3, y=0.23+q, label=subSampleProbs[3])+
92        annotate("text",x=4, y=0.28+q, label=subSampleProbs[4])+
93        annotate("text",x=5, y=0.37+q, label=subSampleProbs[5])+
94        annotate("text",x=3, y=0.1, label="Bayes Error")
95  # plot boxplots for WSRF
96  resultsWSRF <- data.frame(set=rep(c("set1", "set2", "set3", "set4", "set5","
        set6", "set7", "set8", "set9", "set99"), each=50),
97                            error=c(SimErrorsRF[[1]], SimErrorsWSRF[[1]],
                                  SimErrorsRF[[2]],
98                                      SimErrorsWSRF[[2]],SimErrorsRF[[3]],
                                            SimErrorsWSRF[[3]],
99                                      SimErrorsRF[[4]],SimErrorsWSRF[[4]],
100                                     SimErrorsRF[[5]],SimErrorsWSRF[[5]]),
101                           grp=rep(rep(c("Forest-RI", "WSRF"), each=50),5))
102 ggplot(resultsWSRF, aes(y=error, x=set, fill=grp)) + stat_boxplot(geom ='
        errorbar', width=0.5)+
```

```
103         geom_boxplot(notch = TRUE, outlier.color = "red") +
104         geom_hline(yintercept = q, linetype="dashed", col="purple") +
105         xlab("Number of (relevant, noise) variables") + ylim(0, 0.75) +
106         ylab("Test Misclassification Error") +
107         scale_x_discrete(labels=rep(c("(2, 10)", "(2, 50)", "(2, 100)", "(2,
                200)", "(2, 1000)"), each=2))+
108         scale_fill_manual(name="Model", labels=c("Forest-RI", "WSRF"), values=c(
                "darkgreen", "blue"))+
109         theme_bw()+
110         theme(legend.position=c(0.1, 0.6))+
111         annotate("text",x=5.5, y=0.1, label="Bayes Error")
112 # ------------------------------------------------------------------------------
113 ##############################################
114 # Figure 6.3: Binary tree representation
115 ##############################################
116 # ------------------------------------------------------------------------------
117 # create empty plot
118 dmar <- par()$mar
119 par(mar=c(0,0,0,0))
120 plot(11:22, 11:22, type="n", xlab="", ylab="",
121       xlim=c(11, 22), ylim=c(1, 22),
122       main="", axes=FALSE)
123 # create tree
124 text(16.5, 20, TeX("Node 1"), col="darkgreen")
125 text(15.5, 17.5, TeX("<< if $\\phi_1(\\underline{x}, \\Theta) = 0,$"))
126 text(17.5, 17.5, TeX("if $\\phi_1(\\underline{x}, \\Theta) = 1$ >>"))
127 lines(c(16.5, 16.5), c(18.5, 19.5))
128 lines(c(14, 19),c(19, 19))
129 # splits
130 lines(c(14, 14), c(19, 16))
131 lines(c(19, 19), c(19, 16))
132 # internal nodes
133 text(14, 15, TeX("Node 2"), col="darkgreen")
134 text(19, 15, TeX("Node 3"), col="darkgreen")
135 lines(c(14, 14), c(13.5, 14.5))
136 lines(c(13, 15), c(14, 14))
137 lines(c(19, 19), c(13.5, 14.5))
138 lines(c(18, 20), c(14, 14))
139 # split internal node 1
140 lines(c(13, 13), c(14, 9))
141 lines(c(15, 15), c(14, 9))
142 # split internal node 2
143 lines(c(18, 18), c(14, 9))
144 lines(c(20, 20), c(14, 9))
145 # root nodes 1, 2, 3
146 text(13, 8, TeX("$-$"), col="blue", cex=1.5)
147 text(15, 8, TeX("$+_{(1)}$"), col="darkorange", cex=1.5)
148 text(20, 8, TeX("$-$"), col="blue", cex=1.5)
149 # internal node 3
150 text(18, 8, TeX("Node 4"), col="darkgreen")
151 lines(c(18, 18), c(6.5, 7.5))
152 lines(c(17, 19), c(7, 7))
153 lines(c(17, 17), c(7, 3))
154 lines(c(19, 19), c(7, 3))
155 # root node 4 and 5
156 text(17, 2, TeX("$+_{(2)}$"), col="darkorange", cex=1.5)
157 text(19, 2, TeX("$-$"), col="blue", cex=1.5)
158 par(mar=dmar)
159 # ------------------------------------------------------------------------------
160 ##########################################################################################
161 # Figure 6.4: Logistic sigmoid function used to approximate a tree node
162 # splitting rule.
163 ##########################################################################################
164 # ------------------------------------------------------------------------------
165 x <- seq(from=-10, to=10, by=0.01)
166 y <- 1/(1+exp(-x))
```

```
sig <- data.frame(x=x, y=y)
ggplot(sig, aes(x=x,y=y)) + geom_line() + theme_bw()+
      geom_vline(xintercept = 0, col="orange")+
      geom_hline(yintercept = 0.5, col="purple", linetype="dashed")+
      scale_x_discrete(limits=c(0), labels=c("-b"))+
      ylab(TeX("$\\hat{\\phi}(\\underline{x}, \\Theta)$"))+
      annotate("text", x=-4.7, y=0.25, label="1/(1+exp(-b-X))")
# ------------------------------------------------------------------------
##########################################################################
# Table 6.1: The variables describing each random forest algorithm
##########################################################################
# ------------------------------------------------------------------------
# make mds plot of variants
# read in data
data <- read.csv("RFvariantsData.csv")
tableframe <- data.frame("variable"=colnames(data)[-1], "type"=c("categorical"
    , rep("numeric", 17)),
           "range"=c("NA", "1988 - 2015", rep("{0, 1}", 16)))
stargazer(tableframe, summary = FALSE)
# ------------------------------------------------------------------------
##########################################################################################
# Figure 6.5: Trait based comparison of random forest proposals by way of a
# best two-dimensional MDS approximation of the full trait space.
##########################################################################################
# ------------------------------------------------------------------------
# Compute group colors
# sources of randomness
gcols <- NULL
for(i in 1:nrow(data)){
      if(data[i,]$r_data == 0
         && data[i,]$r_subsample_var == 0 && data[i,]$r_split_points == 0
         && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "navy"
      } else if(data[i,]$r_data == 1
                && data[i,]$r_subsample_var == 0 && data[i,]$r_split_points ==
                   0
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "blue"
      } else if(data[i,]$r_data == 1
                && data[i,]$r_subsample_var == 1 && data[i,]$r_split_points ==
                   0
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "orange"
      } else if(data[i,]$r_data == 1
                && data[i,]$r_subsample_var == 1 && data[i,]$r_split_points ==
                   1
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "tan"
      } else if(data[i,]$r_data == 0
                && data[i,]$r_subsample_var == 1 && data[i,]$r_split_points ==
                   0
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "darkgreen"
      } else if(data[i,]$r_data == 0
                && data[i,]$r_subsample_var == 1 && data[i,]$r_split_points ==
                   1
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "tomato4"
      } else if(data[i,]$r_data == 0
                && data[i,]$r_subsample_var == 0 && data[i,]$r_split_points ==
                   1
                && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 0){
            gcols[i] <- "red"
      } else if(data[i,]$r_data == 0
                && data[i,]$r_subsample_var == 0 && data[i,]$r_split_points ==
                   0
```

```
225                 && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 1){
226             gcols[i] <- "purple"
227         } else if(data[i,]$r_data == 1
228                 && data[i,]$r_subsample_var == 1 && data[i,]$r_split_points ==
                        0
229                 && data[i,]$r_voting == 0 && data[i,]$r_ensemble == 1){
230             gcols[i] <- "green"
231         }
232 }
233
234 cols <- c("forestgreen", "darkred", "gold4", "skyblue",
235           "orange3", "magenta", "royalblue", "seagreen",
236           "red3", "peru", "violet", "yellow4",
237           "springgreen3", "tomato2", "skyblue3", "sienna2",
238           "plum4")
239 ccols <- NULL
240 for(i in 1:nrow(data)){
241         if(sum(data[i, 9:10]) > 0 && sum(data[i,11:19]) == 0){
242             ccols[i] <- cols[1]
243         } else if(sum(data[i,9:10]) == 0 && sum(data[i,11:14]) > 0 && sum(data[i
                ,15:19]) == 0){
244             ccols[i] <- cols[2]
245         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) == 0 && sum(data
                [i,15:16]) > 0 && sum(data[i,17:19]) == 0){
246             ccols[i] <- cols[3]
247         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) == 0 && sum(data
                [i,15:16]) == 0 && sum(data[i,17:19]) > 0){
248             ccols[i] <- cols[4]
249         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) > 0 && sum(data[i
                ,15:16]) == 0 && sum(data[i,17:19]) == 0){
250             ccols[i] <- cols[5]
251         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) == 0 && sum(data[
                i,15:16]) > 0 && sum(data[i,17:19]) == 0){
252             ccols[i] <- cols[6]
253         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) == 0 && sum(data[
                i,15:16]) == 0 && sum(data[i,17:19]) > 0){
254             ccols[i] <- cols[7]
255         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) > 0 && sum(data[
                i,15:16]) > 0 && sum(data[i,17:19]) == 0){
256             ccols[i] <- cols[8]
257         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) == 0 && sum(data
                [i,15:16]) > 0 && sum(data[i,17:19]) > 0){
258             ccols[i] <- cols[9]
259         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) > 0 && sum(data[
                i,15:16]) == 0 && sum(data[i,17:19]) > 0){
260             ccols[i] <- cols[10]
261         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) > 0 && sum(data[i
                ,15:16]) > 0 && sum(data[i,17:19]) == 0){
262             ccols[i] <- cols[11]
263         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) > 0 && sum(data[i
                ,15:16]) == 0 && sum(data[i,17:19]) > 0){
264             ccols[i] <- cols[12]
265         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) == 0 && sum(data[
                i,15:16]) > 0 && sum(data[i,17:19]) > 0){
266             ccols[i] <- cols[13]
267         } else if(sum(data[i, 9:10]) == 0 && sum(data[i,11:14]) > 0 && sum(data[
                i,15:16]) > 0 && sum(data[i,17:19]) > 0){
268             ccols[i] <- cols[14]
269         } else if(sum(data[i, 9:10]) > 0 && sum(data[i,11:14]) > 0 && sum(data[i
                ,15:16]) > 0 && sum(data[i,17:19]) > 0){
270             ccols[i] <- cols[15]
271         }
272 }
273
274 # use MDS to obtain optimal 2d approx space
275 num_data <- data[,-c(1,2,3)]
```

```
276 p <- cmdscale(dist(num_data))
277
278 # plot display
279 par(mar=c(1,1,1,1))
280 plot(x=p[,1], y=p[,2], xlim=c(-1.5, 3), ylim=c(-1,1.2), xaxt="n", yaxt="n",
281      xlab="", ylab="", col=ccols, pch=18)
282 text(p[1,1], p[1,2], paste(data$author[1], data$year[1]), pos=1, cex=0.6, col
    = gcols[1])
283 text(p[2,1], p[2,2], paste(data$author[2], data$year[2]), pos=1, cex=0.6, col
    = gcols[2])
284 text(p[3,1], p[3,2], paste(data$author[3], data$year[3]), pos=3, cex=0.6, col
    = gcols[3], offset=0.8)
285 text(p[4,1], p[4,2], paste(data$author[4], data$year[4]), pos=1, cex=0.6, col
    = gcols[4])
286 text(p[5,1], p[5,2], paste(data$author[5], data$year[5]), pos=4, cex=0.6, col
    = gcols[5])
287 text(p[6,1], p[6,2], paste(data$author[6], data$year[6]), pos=3, cex=0.6, col
    = gcols[6])
288 text(p[7,1], p[7,2], paste(data$author[7], data$year[7]), pos=3, cex=0.6, col
    = gcols[7])
289 text(p[8,1], p[8,2], paste(data$author[8], data$year[8]), pos=4, cex=0.6, col
    = gcols[8])
290 text(p[9,1], p[9,2], paste(data$author[9], data$year[9]), pos=4, cex=0.6, col
    = gcols[9])
291 text(p[10,1],p[10,2], paste(data$author[10], data$year[10]), pos=2, cex=0.6,
    col = gcols[10])
292 text(p[11,1],p[11,2], paste(data$author[11], data$year[11]), pos=2, cex=0.6,
    col = gcols[11])
293 text(p[12,1],p[12,2], paste(data$author[12], data$year[12]), pos=3, cex=0.6,
    col = gcols[12])
294 text(p[13,1],p[13,2], paste(data$author[13], data$year[13]), pos=4, cex=0.6,
    col = gcols[13])
295 text(p[14,1],p[14,2], paste(data$author[14], data$year[14]), pos=4, cex=0.6,
    col = gcols[14])
296 text(p[15,1], p[15,2], paste(data$author[15], data$year[15]), pos=1, cex=0.6,
    col = gcols[15])
297 text(p[16,1], p[16,2], paste(data$author[16], data$year[16]), pos=1, cex=0.6,
    col = gcols[16])
298 text(p[17,1], p[17,2], paste(data$author[17], data$year[17]), pos=1, cex=0.6,
    col = gcols[17])
299 text(p[18,1], p[18,2], paste(data$author[18], data$year[18]), pos=2, cex=0.6,
    col = gcols[18])
300 text(p[19,1], p[19,2], paste(data$author[19], data$year[19]), pos=2, cex=0.6,
    col = gcols[19])
301 text(p[20,1], p[20,2], paste(data$author[20], data$year[20]), pos=4, cex=0.6,
    col = gcols[20])
302 text(p[21,1], p[21,2], paste(data$author[21], data$year[21]), pos=2, cex=0.6,
    col = gcols[21])
303 text(p[22,1], p[22,2], paste(data$author[22], data$year[22]), pos=4, cex=0.6,
    col = gcols[22])
304 text(p[23,1], p[23,2], paste(data$author[23], data$year[23]), pos=3, cex=0.6,
    col = gcols[23])
305 text(p[24,1], p[24,2], paste(data$author[24], data$year[24]), pos=2, cex=0.6,
    col = gcols[24])
306 text(p[25,1], p[25,2], paste(data$author[25], data$year[25]), pos=2, cex=0.6,
    col = gcols[25])
307 text(p[26,1], p[26,2], paste(data$author[26], data$year[26]), pos=3, cex=0.6,
    col = gcols[26])
308 text(p[27,1], p[27,2], paste(data$author[27], data$year[27]), pos=2, cex=0.6,
    col = gcols[27])
309 text(p[28,1], p[28,2], paste(data$author[28], data$year[28]), pos=2, cex=0.6,
    col = gcols[28])
310 text(p[29,1], p[29,2], paste(data$author[29], data$year[29]), pos=2, cex=0.6,
    col = gcols[29])
311 text(p[30,1], p[30,2], paste(data$author[30], data$year[30]), pos=4, cex=0.6,
    col = gcols[30])
```

```
text(p[31,1], p[31,2], paste(data$author[31], data$year[31]), pos=3, cex=0.6,
    col = gcols[31], offset = 1)
text(p[32,1], p[32,2], paste(data$author[32], data$year[32]), pos=1, cex=0.6,
    col = gcols[32])
text(p[33,1], p[33,2], paste(data$author[33], data$year[33]), pos=1, cex=0.6,
    col = gcols[33], offset = 1)
text(p[34,1], p[34,2], paste(data$author[34], data$year[34]), pos=4, cex=0.6,
    col = gcols[34])
text(p[35,1], p[35,2], paste(data$author[35], data$year[35]), pos=1, cex=0.6,
    col = gcols[35])
text(p[36,1], p[36,2], paste(data$author[36], data$year[36]), pos=2, cex=0.6,
    col = gcols[36])
text(p[37,1], p[37,2], paste(data$author[37], data$year[37]), pos=1, cex=0.6,
    col = gcols[37])
# add custom legend
text(2.5, 1+0.2, "Randomisation Sources", cex=0.6)
text(2.3, 0.94+0.2, "auth+year:", col="blue", cex=0.6)
text(2.3, 0.89+0.2, "auth+year:", col="orange", cex=0.6)
text(2.3, 0.84+0.2, "auth+year:", col="tan", cex=0.6)
text(2.3, 0.79+0.2, "auth+year:", col="green", cex=0.6)
text(2.3, 0.74+0.2, "auth+year:", col="darkgreen", cex=0.6)
text(2.3, 0.69+0.2, "auth+year:", col="tomato4", cex=0.6)
text(2.3, 0.64+0.2, "auth+year:", col="red", cex=0.6)
text(2.3, 0.59+0.2, "auth+year:", col="purple", cex=0.6)
# add categories
text(2.78, 0.94+0.2, "R.1", cex=0.6)
text(2.78, 0.89+0.2, "R.1, R.2", cex=0.6)
text(2.78, 0.84+0.2, "R.1, R.2, R.3", cex=0.6)
text(2.78, 0.79+0.2, "R.1, R.2, R.4", cex=0.6)
text(2.78, 0.74+0.2, "R.2", cex=0.6)
text(2.78, 0.69+0.2, "R.2, R.3", cex=0.6)
text(2.78, 0.64+0.2, "R.3", cex=0.6)
text(2.78, 0.59+0.2, "R.4", cex=0.6)
# add deterministic modifications
text(2.53, 0.5, "Deterministic Modifications", cex=0.6)
pcols <- c("orange3", "violet", "skyblue3", "darkred",
           "seagreen", "peru", "tomato2")
xpoints <- rep(2.2, 7)
ypoints <- c(0.44, 0.39, 0.34, 0.29, 0.24, 0.19, 0.14)
points(xpoints,ypoints,col=pcols,pch=18)
# add categories DM
text(2.5, 0.44, "A, B", cex=0.6)
text(2.5, 0.39, "A, B, C", cex=0.6)
text(2.5, 0.34, "A, B, C, D", cex=0.6)
text(2.5, 0.29, "B", cex=0.6)
text(2.5, 0.24, "B, C", cex=0.6)
text(2.5, 0.19, "B, D", cex=0.6)
text(2.5, 0.14, "B, C, D", cex=0.6)
# ------------------------------------------------------------------------------
################################################################################
# Figure 6.6: Random forest decision boundaries: top left: extremely
# randomised forest; top right: rotation random forest; middle left: oblique
# random forest with logistic regression splits; middle right: weighted
# subspace random forest; bottom left: regularised random forest (Î» = 0.1);
# bottom right: regularised random forest (Î» = 0.6).
################################################################################
# ------------------------------------------------------------------------------
# Generate training data
set.seed(1)
mBlue <- mvrnorm(n=10, mu = c(1,0),Sigma = diag(1,2,2))
mOrange <- mvrnorm(n=10, mu = c(0,1),Sigma = diag(1,2,2))
B <- matrix(0,nrow=100,ncol=2)
O <- matrix(0,nrow=100,ncol=2)

for(i in 1:100){
      sample1 = sample(1:10, 1)
```

```
371         sample2 = sample(1:10, 1)
372         meanB = mBlue[sample1,]
373         meanO = mOrange[sample2,]
374         B[i,] = mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
375         O[i,] = mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
376 }
377
378 Btrain <- cbind(B[1:100,],matrix(0,100,1))
379 Otrain <- cbind(O[1:100,],matrix(1,100,1))
380 datatrain <- rbind(Btrain,Otrain)
381 Xtrain <- datatrain[,1:2]
382 Ytrain <- datatrain[,3]
383 train <- data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2])
384
385 # create decision boundary plotting grid
386 x1min <- min(Xtrain[,1])
387 x1max <- max(Xtrain[,1])
388 x2min <- min(Xtrain[,2])
389 x2max <- max(Xtrain[,2])
390 x1seq <- seq(from=x1min,to=x1max,length=100)
391 x2seq <- seq(from=x2min,to=x2max,length=100)
392 plotGrid <- data.frame(as.matrix(expand.grid(x1seq,x2seq)))
393 colnames(plotGrid) <- colnames(train)[2:3]
394
395 # create test set
396 B <- matrix(0,nrow=5000,ncol=2)
397 O <- matrix(0,nrow=5000,ncol=2)
398 for(i in 1:5000){
399         sample1 <- sample(1:10, 1)
400         sample2 <- sample(1:10, 1)
401         meanB <- mBlue[sample1,]
402         meanO <- mOrange[sample2,]
403         B[i,] <- mvrnorm(1,mu=meanB,Sigma=diag(1/5,2,2))
404         O[i,] <- mvrnorm(1,mu=meanO,Sigma=diag(1/5,2,2))
405 }
406
407 Btest <- cbind(B[1:5000,],matrix(0,5000,1))
408 Otest <- cbind(O[1:5000,],matrix(1,5000,1))
409 datatest <- rbind(Btest,Otest)
410 Xtest <- datatest[,1:2]
411 Ytest <- datatest[,3]
412 test <- data.frame(y=factor(Ytest), X1=Xtest[,1], X2=Xtest[,2])
413
414 # plot data
415 color <- ifelse(train$y == 0, "blue", "darkorange")
416 # Bayes decision boundary
417 p <- function(x) {
418         s <- sqrt(1/5)
419         p0 <- mean(dnorm(x[1], mBlue[,1], s) * dnorm(x[2], mBlue[,2], s))
420         p1 <- mean(dnorm(x[1], mOrange[,1], s) * dnorm(x[2], mOrange[,2], s))
421         p1/(p0+p1)
422 }
423
424 bayesrule <- apply(plotGrid, 1, p)
425 bayesPr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(
        x1seq)),
426                     z=as.vector(bayesrule))
427 bayesProbs <- apply(test[,2:3], 1, p)
428 bayesError <- sum(as.numeric(test$y != factor(ifelse(bayesProbs>0.5, 1, 0))))/
        nrow(test)
429
430 # Extremely randomised trees
431 fitControl <- trainControl(method="none")
432 tuneControl <- data.frame(mtry=1, numRandomCuts=1)
433 set.seed(13)
434 erf.fit <- train(y~., data=train, method="extraTrees", trControl=fitControl,
```

```
435                 tuneGrid=tuneControl, ntree=100)
436
437 # compute training and test error
438 erfTrainPreds <- predict(erf.fit, newdata=train[,-1])
439 erfTrainingError <- sum(as.numeric(train$y != erfTrainPreds))/nrow(train)
440
441 # Compute test error
442 erfTestPreds <- predict(erf.fit, test)
443 erfTestError <- sum(as.numeric(test$y != erfTestPreds))/nrow(test)
444
445 # construct decision boundary plot
446 erfProbs <- predict(erf.fit, plotGrid, type="prob")[,2]
447 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
448                z=as.vector(erfProbs))
449 gd <- expand.grid(x=x1seq, y=x2seq)
450 gerf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]), aes
        (x=X1, y=X2)) +
451   geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
452              col=ifelse(erfProbs<0.5, "skyblue", "orange")) +
453   geom_point(size = 3, pch = train$y, col=color) +
454   geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
455   geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
          , breaks=c(0,.5)) +
456   theme_bw() +
457   theme(legend.position="none")+
458   scale_color_manual(name="ERF decision boundary:",values=c("purple", "
          brown"),
459                      labels = c('Bayes','ERF'))+
460   scale_linetype_manual(name = 'ERF decision boundary:', values = c("
          dashed", "solid"),
461                         labels = c('Bayes','ERF'))+
462   annotate("text", x = 2.2, y = -1.6, size=3,
463            label = paste("Training error:", round(erfTrainingError, 3),
464                          "\nTest error:", round(erfTestError,3),
465                          "\nBayes error:", round(bayesError,3)), hjust=0)+
466   ggtitle("Extremely Randomised Forest")
467 gerf
468 # Rotation forest
469 fitControl <- trainControl(method="none")
470 tuneControl <- data.frame(K=1, L=10)
471 set.seed(13)
472 rotrf.fit <- train(y~., data=train, method="rotationForest", trControl=
        fitControl,
473                    tuneGrid=tuneControl)
474
475 # compute training and test error
476 rotrfTrainPreds <- predict(rotrf.fit, newdata=train[,-1])
477 rotrfTrainingError <- sum(as.numeric(train$y != rotrfTrainPreds))/nrow(train)
478
479 # Compute test error
480 rotrfTestPreds <- predict(rotrf.fit, test)
481 rotrfTestError <- sum(as.numeric(test$y != rotrfTestPreds))/nrow(test)
482
483 # construct decision boundary plot
484 rotrfProbs <- predict(rotrf.fit, plotGrid, type="prob")[,2]
485 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
486                z=as.vector(rotrfProbs))
487 gd <- expand.grid(x=x1seq, y=x2seq)
488 grotrf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]),
        aes(x=X1, y=X2)) +
489   geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
490              col=ifelse(rotrfProbs<0.5, "skyblue", "orange")) +
491   geom_point(size = 3, pch = train$y, col=color) +
492   geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
```

```
493       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
              , breaks=c(0,.5)) +
494       theme_bw() +
495       theme(legend.position="none")+
496       scale_color_manual(name="RotRF decision boundary:",values=c("purple", "
              brown"),
497                          labels = c('Bayes','RotRF'))+
498       scale_linetype_manual(name = 'RotRF decision boundary:', values = c("
              dashed", "solid"),
499                             labels = c('Bayes','RotRF'))+
500       annotate("text", x = 2.2, y = -1.6, size=3,
501                label = paste("Training error:", round(rotrfTrainingError, 3),
502                              "\nTest error:", round(rotrfTestError,3),
503                              "\nBayes error:", round(bayesError,3)), hjust=0)+
504       ggtitle("Rotation Random Forest")
505 grotrf
506
507 # Oblique RF - logistice regression splits
508 set.seed(13)
509 orf.fit <- obliqueRF(y=as.numeric(train$y), x=as.matrix(train[,2:3]),
510                      mtry=2, training_method="log", ntree=100)
511
512 # compute training and test error
513 orfTrainPreds <- predict(orf.fit, newdata=train[,-1])
514 orfTrainingError <- sum(as.numeric(as.numeric(train$y) != as.numeric(
        orfTrainPreds)))/nrow(train)
515
516 # Compute test error
517 orfTestPreds <- predict(orf.fit, test[,-1])
518 orfTestError <- sum(as.numeric(as.numeric(test$y) != as.numeric(orfTestPreds))
        )/nrow(test)
519
520 # construct decision boundary plot
521 orfProbs <- predict(orf.fit, plotGrid, type="prob")[,2]
522 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
523                z=as.vector(orfProbs))
524 gd <- expand.grid(x=x1seq, y=x2seq)
525 gorf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]), aes
        (x=X1, y=X2)) +
526       geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
527                  col=ifelse(orfProbs<0.5, "skyblue", "orange")) +
528       geom_point(size = 3, pch = train$y, col=color) +
529       geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
              dashed"), breaks=c(0,.5))+
530       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
              , breaks=c(0,.5)) +
531       theme_bw() +
532       theme(legend.position="none")+
533       scale_color_manual(name="ORF-log decision boundary:",values=c("purple",
              "brown"),
534                          labels = c('Bayes','ORF-log'))+
535       scale_linetype_manual(name = 'ORF-log decision boundary:', values = c("
              dashed", "solid"),
536                             labels = c('Bayes','ORF-log'))+
537       annotate("text", x = 2.2, y = -1.6, size=3,
538                label = paste("Training error:", round(orfTrainingError, 3),
539                              "\nTest error:", round(orfTestError,3),
540                              "\nBayes error:", round(bayesError,3)), hjust=0)+
541       ggtitle("Oblique Random Forest - logistic regression splits")
542 gorf
543
544 # weighted subspace random forests
545 fitControl <- trainControl(method="none")
546 tuneControl <- data.frame(mtry=2)
547 set.seed(13)
548 wsrf.fit <- train(y~., data=train, method="wsrf", trControl=fitControl,
```

```
549                     tuneGrid=tuneControl)
550
551 # compute training and test error
552 wsrfTrainPreds <- predict(wsrf.fit, newdata=train[,-1])
553 wsrfTrainingError <- sum(as.numeric(train$y != wsrfTrainPreds))/nrow(train)
554
555 # Compute test error
556 wsrfTestPreds <- predict(wsrf.fit, test)
557 wsrfTestError <- sum(as.numeric(test$y != wsrfTestPreds))/nrow(test)
558
559 # construct decision boundary plot
560 wsrfProbs <- predict(wsrf.fit, plotGrid, type="prob")[,2]
561 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
562                z=as.vector(wsrfProbs))
563 gd <- expand.grid(x=x1seq, y=x2seq)
564 gwsrf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]),
      aes(x=X1, y=X2)) +
565       geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
566                  col=ifelse(wsrfProbs<0.5, "skyblue", "orange")) +
567       geom_point(size = 3, pch = train$y, col=color) +
568       geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
569       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
          , breaks=c(0,.5)) +
570       theme_bw() +
571       theme(legend.position="none")+
572       scale_color_manual(name="WSRF decision boundary:",values=c("purple", "
          brown"),
573                          labels = c('Bayes','WSRF'))+
574       scale_linetype_manual(name = 'WSRF decision boundary:', values = c("
          dashed", "solid"),
575                             labels = c('Bayes','WSRF'))+
576       annotate("text", x = 2.2, y = -1.6, size=3,
577                label = paste("Training error:", round(wsrfTrainingError, 3),
578                              "\nTest error:", round(wsrfTestError,3),
579                              "\nBayes error:", round(bayesError,3)), hjust=0)+
580       ggtitle("Weighted Subspace Random Forest")
581 gwsrf
582
583 # regularised random forests (0.1)
584 fitControl <- trainControl(method="none")
585 tuneControl <- data.frame(mtry=2, coefReg=0.1)
586 set.seed(13)
587 rrf.fit <- train(y~., data=train, method="RRFglobal", trControl=fitControl,
588                  tuneGrid=tuneControl)
589
590 # compute training and test error
591 rrfTrainPreds <- predict(rrf.fit, newdata=train[,-1])
592 rrfTrainingError <- sum(as.numeric(train$y != rrfTrainPreds))/nrow(train)
593
594 # Compute test error
595 rrfTestPreds <- predict(rrf.fit, test)
596 rrfTestError <- sum(as.numeric(test$y != rrfTestPreds))/nrow(test)
597
598 # construct decision boundary plot
599 rrfProbs <- predict(rrf.fit, plotGrid, type="prob")[,2]
600 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
601                z=as.vector(rrfProbs))
602 gd <- expand.grid(x=x1seq, y=x2seq)
603 grrf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]), aes
      (x=X1, y=X2)) +
604       geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
605                  col=ifelse(rrfProbs<0.5, "skyblue", "orange")) +
606       geom_point(size = 3, pch = train$y, col=color) +
607       geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
          dashed"), breaks=c(0,.5))+
```

```
608       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
              , breaks=c(0,.5)) +
609       theme_bw() +
610       theme(legend.position="none")+
611       scale_color_manual(name="RRF decision boundary:",values=c("purple", "
              brown"),
612                          labels = c('Bayes','RRF'))+
613       scale_linetype_manual(name = 'RRF decision boundary:', values = c("
              dashed", "solid"),
614                             labels = c('Bayes','RRF'))+
615       annotate("text", x = 2.2, y = -1.6, size=3,
616                label = paste("Training error:", round(rrfTrainingError, 3),
617                              "\nTest error:", round(rrfTestError,3),
618                              "\nBayes error:", round(bayesError,3)), hjust=0)+
619       ggtitle(TeX("Regularised Random Forest ($\\lambda = 0.1$)"))
620 grrf
621
622 # regularised random forests (0.6)
623 fitControl <- trainControl(method="none")
624 tuneControl <- data.frame(mtry=2, coefReg=0.6)
625 set.seed(13)
626 rrf.fit <- train(y~., data=train, method="RRFglobal", trControl=fitControl,
627                  tuneGrid=tuneControl)
628
629 # compute training and test error
630 rrfTrainPreds <- predict(rrf.fit, newdata=train[,-1])
631 rrfTrainingError <- sum(as.numeric(train$y != rrfTrainPreds))/nrow(train)
632
633 # Compute test error
634 rrfTestPreds <- predict(rrf.fit, test)
635 rrfTestError <- sum(as.numeric(test$y != rrfTestPreds))/nrow(test)
636
637 # construct decision boundary plot
638 rrfProbs <- predict(rrf.fit, plotGrid, type="prob")[,2]
639 pr<-data.frame(x=rep(x1seq, length(x2seq)), y=rep(x2seq, each=length(x1seq)),
640                z=as.vector(rrfProbs))
641 gd <- expand.grid(x=x1seq, y=x2seq)
642 grrf <- ggplot(data.frame(y=factor(Ytrain), X1=Xtrain[,1], X2=Xtrain[,2]), aes
        (x=X1, y=X2)) +
643       geom_point(data=data.frame(gd), aes(x=x, y=y),pch=".", cex=1.2,
644                  col=ifelse(rrfProbs<0.5, "skyblue", "orange")) +
645       geom_point(size = 3, pch = train$y, col=color) +
646       geom_contour(data=bayesPr, aes(x=x, y=y, z=z, col="brown", linetype="
              dashed"), breaks=c(0,.5))+
647       geom_contour(data=pr, aes(x=x, y=y, z=z, col="purple", linetype="solid")
              , breaks=c(0,.5)) +
648       theme_bw() +
649       theme(legend.position="none")+
650       scale_color_manual(name="RRF decision boundary:",values=c("purple", "
              brown"),
651                          labels = c('Bayes','RRF'))+
652       scale_linetype_manual(name = 'RRF decision boundary:', values = c("
              dashed", "solid"),
653                             labels = c('Bayes','RRF'))+
654       annotate("text", x = 2.2, y = -1.6, size=3,
655                label = paste("Training error:", round(rrfTrainingError, 3),
656                              "\nTest error:", round(rrfTestError,3),
657                              "\nBayes error:", round(bayesError,3)), hjust=0)+
658       ggtitle(TeX("Regularised Random Forest ($\\lambda = 0.6$)"))
659 grrf
660 # ------------------------------------------------------------------------------
661 ################################################################################
662 # Table 6.2: Estimated bias, variance, systematic and variance effects for
663 # random forest algorithms.
664 ################################################################################
665 # ------------------------------------------------------------------------------
```

```
majVote <- function(x){names(which.max(table(x)))}
nTrain <- 400
nTest <- 1000
Models <- factor(rep(c("Forest-RI","ERF", "RotationRF", "ORF-log"), each=6),
    level=c("Forest-RI","ERF", "RotationRF", "ORF-log"))

# performs computations in parallel
cl <- makeCluster(3, type="SOCK")
registerDoSNOW(cl)

# MAIN EXPERIMENT FUNCTIONS
runBiasVarSimulation <- function(trainingSets, simTest, BayesPreds){

    # parameter tuning settings
    fitControl <- trainControl(method = "cv", number = 10)
    rfparaGrid <- expand.grid(mtry=c(1, floor(sqrt(ncol(simTest)-1)), floor(
        ncol(simTest)/2)))
    orfparaGrid <- expand.grid(mtry=c(1, floor(sqrt(ncol(simTest)-1)), floor(
        ncol(simTest)/2)))
    rrfparaGrid <- expand.grid(L=200, K=floor((ncol(simTest)-1)/c(2, 3, 4)))
    erfparaGrid <- expand.grid(mtry=c(1, floor(sqrt(ncol(simTest)-1)), floor(
        ncol(simTest)/2)), numRandomCuts=c(1, 5, 10, nrow(simTest)/2))

    # extremely randomised trees model
    sim.ERF <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
        simTest,
                                     method="extraTrees", paraGrid =
                                         erfparaGrid,
                                 tControl = fitControl, BayesPreds =
                                     BayesPreds, ntree=200)
    # rotation random forest
    sim.RRF <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
        simTest,
                                     method="rotationForest", paraGrid =
                                         rrfparaGrid,
                                 tControl = fitControl, BayesPreds =
                                     BayesPreds)
    # oblique random forest (logistic) model
    sim.ORF <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=
        simTest,
                                     method="ORFlog", paraGrid =
                                         orfparaGrid,
                                 tControl = fitControl, BayesPreds =
                                     BayesPreds, ntree=200)
    # random forest model
    sim.RF <- simulateBiasVarDecomp(trainingSets=trainingSets, simTest=simTest
        ,
                                 method="rf", paraGrid = rfparaGrid,
                                 tControl = fitControl, BayesPreds =
                                     BayesPreds, ntree=200)


    list(results=rbind(sim.RF$results, sim.ERF$results, sim.RRF$results, sim.
        ORF$results),
         tuneValues=list(sim.RF$tuneValues, sim.ERF$tuneValues, sim.RRF$
             tuneValues, sim.ORF$tuneValues))
}


simulateBiasVarDecomp <- function(trainingSets, simTest, method, paraGrid,
    tControl, BayesPreds, ...){

    majVote <- function(x){names(which.max(table(x)))}
    tuneVals <- paraGrid[1,]
    numOfExp <- 100
    # train models and make predictions
```

```
714     BVpreds <- matrix(0, nrow=numOfExp, ncol=nTest)
715     var.T <- NULL
716     var <- NULL
717     bias <- NULL
718     VE <- NULL
719     SE <- NULL
720     misclassError <- NULL
721     C <- as.numeric(simTest$classes)
722
723     # train models
724     for(j in 1:numOfExp){
725         Model <- train(classes~., data=trainingSets[[j]], method=method,
726                        tuneGrid=paraGrid, trControl=tControl, ...)
727         tuneVals <- rbind(tuneVals, Model$bestTune)
728         BVpreds[j,] <- as.numeric(predict(Model, simTest))
729         print(paste("Method: ", method, ", Iter: ", j, " out of ", numOfExp))
730     }
731
732     # James (2003) decomposition estimates
733     BayesClassifier <- BayesPreds
734     majVoteClassifier <- apply(BVpreds, 2, function(x)majVote(x))
735     var.T <- mean(BayesClassifier != C)
736     var <- mean(apply(BVpreds, 1, function(x) mean(x != majVoteClassifier)))
737     bias <- mean(majVoteClassifier != BayesClassifier)
738     VE <- mean(apply(BVpreds, 1, function(x) mean(x != C)) - mean(
            majVoteClassifier != C))
739     SE <- mean(majVoteClassifier != C) - mean(BayesClassifier != C)
740     meanError <- mean(apply(BVpreds, 1, function(x){ mean(x != C) }))
741
742     # store bias and variance and systematic effect and variance effect
743     vb <- c(meanError, var.T, SE, VE, bias, var)
744     bar <- factor(c(1,2,3,4,5,6))
745     type <- c("Error", "Bayes Error", "Systematic Effect", "Variance Effect",
            "Bias", "Variance")
746     model <- rep(method, 6)
747     biasVarPlotData <- data.frame(vb=vb, Decomposition=type, bar=bar, model=
            model)
748     list(results=biasVarPlotData, tuneValues=tuneVals[-1,])
749 }
750 ########################
751 # Designed scenarios
752 ########################
753 # load data generation library
754 # simulate data function from "pensim" package
755 simData <- function (nvars = c(100, 100, 100, 100, 600), cors = c(0.8, 0, 0.8,
        0, 0),
756                      associations = c(0.5, 0.5, 0.3, 0.3, 0), firstonly = c(
                         TRUE, FALSE, TRUE, FALSE, FALSE),
757                      nsamples = 100, censoring = "none",
758                      labelswapprob = 0, response = "timetoevent", basehaz =
                         0.2,
759                      logisticintercept = 0)
760 {
761       if (labelswapprob < 0)
762           stop("labelswapprob cannot be negative")
763       if (labelswapprob > 0 & response == "timetoevent")
764           stop("labelswapprob is only implemented for binary response")
765       if (!class(nvars) %in% c("numeric", "integer"))
766           stop("nvars must be a numeric vector")
767       if (!class(cors) %in% c("numeric", "integer"))
768           stop("cors must be a numeric vector")
769       if (class(firstonly) != "logical")
770           stop("firstonly must be a logical vector")
771       if (!class(associations) %in% c("numeric", "integer"))
772           stop("associations must be a numeric vector")
773       if (length(nvars) != length(cors) | length(nvars) != length(firstonly) |
```

```
774             length(nvars) != length(associations))
775               stop("nvars, cors, firstonly, and associations must all have the
                      same length.")
776           x.out <- matrix(0, ncol = sum(nvars), nrow = nsamples)
777           definecors <- data.frame(start = c(1, cumsum(nvars[-length(nvars)]) +
778                                                1), end = cumsum(nvars), cors =
                                                     cors, associations =
                                                    associations,
779                                    num = nvars, firstonly = firstonly, row.names =
                                         letters[1:length(nvars)])
780           Sigma <- matrix(0, ncol = sum(nvars), nrow = sum(nvars))
781           wts <- rep(0, sum(nvars))
782           for (i in 1:nrow(definecors)) {
783                 thisrange <- definecors[i, "start"]:definecors[i, "end"]
784                 Sigma[thisrange, thisrange] <- definecors[i, "cors"]
785                 diag(Sigma) <- 1
786                 x.out[, thisrange] <- mvrnorm(n = nsamples, mu = rep(0,
787                                                                      nvars[i]),
                                                                          Sigma =
                                                                          Sigma[
                                                                          thisrange
                                                                          ,
                                                                          thisrange
                                                                          ])
788                 if (definecors[i, "firstonly"]) {
789                       wts[definecors[i, "start"]] <- definecors[i, "associations"]
790                 }
791                 else {
792                       wts[definecors[i, "start"]:definecors[i, "end"]] <-
                            definecors[i,"associations"]
793                 }
794                 varnames <- paste(letters[i], 1:nvars[i], sep = ".")
795                 names(wts)[definecors[i, "start"]:definecors[i, "end"]] <-
                      varnames
796           }
797           names(wts) <- make.unique(names(wts))
798           dimnames(Sigma) <- list(colnames = names(wts), rownames = names(wts))
799           colnames(x.out) <- names(wts)
800           betaX <- x.out %*% wts
801           x.out <- data.frame(x.out)
802           if (identical(response, "timetoevent")) {
803                 h = basehaz * exp(betaX[, 1])
804                 x.out$time <- rexp(length(h), h)
805                 x.out$cens <- 1
806                 if (class(censoring) == "numeric" | class(censoring) ==
807                     "integer") {
808                       if (length(censoring) == 2) {
809                             censtimes <- runif(length(h), min = censoring[1],
810                                                max = censoring[2])
811                       }
812                       else if (length(censoring) == 1) {
813                             censtimes <- rep(censoring, length(h))
814                       }
815                       x.out$cens[x.out$time > censtimes] <- 0
816                       x.out$time[x.out$time > censtimes] <- censtimes[x.out$time >
                            censtimes]
817                 }
818           }
819           else if (identical(response, "binary")) {
820                 p <- 1/(1 + exp(-(betaX + logisticintercept)))
821                 x.out$outcome <- ifelse(p > runif(length(p)), 1, 0)
822                 if (labelswapprob > 0) {
823                       do.swap <- runif(length(p)) < labelswapprob
824                       new.outcome <- x.out$outcome
825                       new.outcome[x.out$outcome == 1 & do.swap] <- 0
826                       new.outcome[x.out$outcome == 0 & do.swap] <- 1
```

```
827                     x.out$outcome <- new.outcome
828                 }
829                 x.out$outcome <- factor(x.out$outcome+1)
830         }
831         else stop("response must be either timetoevent or binary")
832         return(list(summary = definecors, associations = wts, covariance = Sigma
                ,
833                     data = x.out, probs=p))
834 }
835
836 ####################
837 # SETUP 1: corr=0.9
838 ####################
839 # simluating training data sets
840 trainingSets <- list()
841 for(i in 1:100){
842         set.seed(i+1)
843         train <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
844                          firstonly=c(FALSE), nsamples=400, response="binary")
845         train <- train$data
846         train$classes <- train$outcome
847         trainingSets[[i]] <- train[,-16]
848 }
849
850 # simulate test data set
851 set.seed(1)
852 test <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
853                 firstonly=c(FALSE), nsamples=1000, response="binary")
854 testData <- test$data
855 testData$classes <- testData$outcome
856 simTest <- testData[,-16]
857
858 # run simulation and plot data
859 BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
860 setup1Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
861 setup1Results$results$model <- Models
862 saveRDS(setup1Results, "setup1Results.rda")
863 ####################
864 # SETUP 2: corr=0.5
865 ####################
866 # simluating training data sets
867 trainingSets <- list()
868 for(i in 1:100){
869         set.seed(i+1)
870         train <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
871                          firstonly=c(FALSE), nsamples=400, response="binary")
872         train <- train$data
873         train$classes <- train$outcome
874         trainingSets[[i]] <- train[,-16]
875 }
876
877 # simulate test data set
878 set.seed(1)
879 test <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
880                 firstonly=c(FALSE), nsamples=1000, response="binary")
881 testData <- test$data
882 testData$classes <- testData$outcome
883 simTest <- testData[,-16]
884
885 # run simulation and plot data
886 BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
887 setup2Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
888 setup2Results$results$model <- Models
889 saveRDS(setup2Results, "setup2Results.rda")
890 ####################
891 # SETUP 3: corr=0.1
```

```
####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup3Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup3Results$results$model <- Models
saveRDS(setup3Results, "setup3Results.rda")
####################
# SETUP 4: corr=0
####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup4Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup4Results$results$model <- Models
saveRDS(setup4Results, "setup4Results.rda")

# Mease et al. data scenarios
# simulate data function
generateMeasedata <- function(nTrain=400, nTest=1000, Ndata=100, p=30, J=2,
    seedStart=1, q = 0.15){

      trainingSets <- list()
      # simulate data
      for(iter in 1:Ndata){
            set.seed(iter+1)
            Xtrain<-matrix(0,nTrain,p)
            for (i in 1:p){
                  Xtrain[,i]<-runif(nTrain)
            }
```

```
957             ytrain<-rep(0,nTrain)
958             for (i in 1:nTrain){
959                   ytrain[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtrain[i,1:J]))>(J
                          /2))))+1
960             }
961             # training data
962             trainingSets[[iter]] <- data.frame(classes=factor(ytrain), Xtrain)
963       }
964       set.seed(1)
965       Xtest<-matrix(0,nTest,p)
966       for (i in 1:p){
967             Xtest[,i]<-runif(nTest)
968       }
969       ytest<-rep(0,nTest)
970       for (i in 1:nTest){
971             ytest[i]<-1*(runif(1)<(q+(1-2*q)*1*(sum((Xtest[i,1:J]))>(J/2))))+1
972       }
973       # training sets and test set data
974       testingSets <- data.frame(classes=factor(ytest), Xtest)
975       list(trainingSets=trainingSets, testingSets=testingSets)
976 }
977
978 ##################
979 # Setup 5: J = 2
980 ##################
981 # simluating training data sets
982 q <- 0.15
983 simData1 <- generateMeasedata(J=2)
984 trainingSets <- simData1[[1]]
985 simTest <- simData1[[2]]
986 # run simulation and plot data
987 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
        +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
988 setup5Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
989 setup5Results$results$model <- Models
990 saveRDS(setup5Results, "setup5ResultsAR.rda")
991 ##################
992 # Setup 6: J = 5
993 ##################
994 # simluating training data sets
995 simData1 <- generateMeasedata(J=5)
996 trainingSets <- simData1[[1]]
997 simTest <- simData1[[2]]
998 # run simulation and plot data
999 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
        +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1000 setup6Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
1001 setup6Results$results$model <- Models
1002 saveRDS(setup6Results, "setup6ResultsAR.rda")
1003 ##################
1004 # Setup 7: J = 15
1005 ##################
1006 # simluating training data sets
1007 simData1 <- generateMeasedata(J=15)
1008 trainingSets <- simData1[[1]]
1009 simTest <- simData1[[2]]
1010 # run simulation and plot data
1011 BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
        +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1012 setup7Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
1013 setup7Results$results$model <- Models
1014 saveRDS(setup7Results, "setup7ResultsAR.rda")
1015 ##################
1016 # Setup 8: J = 20
1017 ##################
1018 # simluating training data sets
```

```
simData1 <- generateMeasedata(J=20)
trainingSets <- simData1[[1]]
simTest <- simData1[[2]]
# run simulation and plot data
BayesClasses <- as.numeric(factor(apply(simTest[,-1], 1, function(x) 1*(0.5<(q
    +(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
setup8Results <- runBiasVarSimulation(trainingSets, simTest, BayesClasses)
setup8Results$results$model <- Models
saveRDS(setup8Results, "setup8ResultsAR.rda")

# MLBENCH DATA
############################
# twonorm simulation data
############################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
    set.seed(i+1)
    train <- mlbench.twonorm(400, d=20)
    train <- as.data.frame(train)
    trainingSets[[i]] <- train
}

# simulate test data set
set.seed(1)
test <- mlbench.twonorm(1000, d=20)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
twonormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test)
    )
twonormResults$results$model <- Models
saveRDS(twonormResults, "twonormResultsAR.rda")
##############################
# threenorm simulation data
##############################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
    set.seed(i+1)
    train <- mlbench.threenorm(400, d=20)
    train <- as.data.frame(train)
    trainingSets[[i]] <- train
}

# simulate test data set
set.seed(1)
test <- mlbench.threenorm(1000, d=20)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
threenormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(
    test))
threenormResults$results$model <- Models
saveRDS(threenormResults, "threenormResultsAR.rda")
##############################
# ringnorm simulation data
##############################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
    set.seed(i+1)
    train <- mlbench.ringnorm(400, d=20)
    train <- as.data.frame(train)
```

```
    trainingSets[[i]] <- train
}

# simulate test data set
set.seed(1)
test <- mlbench.ringnorm(1000, d=20)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
ringnormResults <- runBiasVarSimulation(trainingSets, simTest, bayesclass(test
    ))
ringnormResults$results$model <- Models
saveRDS(ringnormResults, "ringnormResultsAR.rda")

#####################################################
# Make table for all roud bias variance results
#####################################################
res1 <- readRDS("twonormResultsAR.rda")
res2 <- list(results=readRDS("threenormResultsAR.rda"))
res3 <- readRDS("ringnormResultsAR.rda")
res4 <- readRDS("circleResultsAR.rda")
res5 <- readRDS("setup1ResultsAR.rda")
res6 <- readRDS("setup2ResultsAR.rda")
res7 <- readRDS("setup3ResultsAR.rda")
res8 <- readRDS("setup4ResultsAR.rda")
res9 <- readRDS("setup5ResultsAR.rda")
res10 <- readRDS("setup6ResultsAR.rda")
res11 <- readRDS("setup7ResultsAR.rda")
res12 <- readRDS("setup8ResultsAR.rda")
resList <- list(res1, res2, res3, res4, res5, res6, res7, res8,
                res9, res10, res11, res12)
tableFinal <- NULL
for(k in 1:length(resList)){
      res <- resList[[k]]
      splitDat <- split(res$results, res$results$model)
      cname <- unique(res$results$model)
      rname <- unique(res$results$Decomposition)
      tableFrame <- matrix(0, nrow=length(rname), ncol=length(cname))
      for(i in 1:length(splitDat)){
            tableFrame[,i] <- splitDat[[i]]$vb
      }
      rownames(tableFrame) <- paste(k, rname)
      colnames(tableFrame) <- cname
      tableFinal <- rbind(tableFinal, tableFrame)
}
tableFinal <- as.data.frame(tableFinal)

n <- nrow(tableFinal)
errorTable <- tableFinal[seq(1, n, by=6),]
SEtable <- tableFinal[seq(3, n, by=6),]
VEtable <- tableFinal[seq(4, n, by=6),]
biasTable <- tableFinal[seq(5, n, by=6),]
varTable <- tableFinal[seq(6, n, by=6),]
compTableList <- list(errorTable, SEtable, VEtable, biasTable, varTable)
compPVals <- list()

# compute omnibus p-vals
library(scmamp)
for(i in 1:length(compTableList)){
      compPVals[[i]] <- friedmanAlignedRanksTest(compTableList[[i]])
}

# compute post-hoc p-vals
postPVals <- list()
for(i in 1:length(compTableList)){
```

```
1147        postPVals[[i]] <- postHocTest(compTableList[[i]], test="friedman",
1148                                      correct="shaffer")
1149 }
1150
1151 # create latex table
1152 stargazer(tableFinal, summary = FALSE)
1153 # ------------------------------------------------------------------------
1154 ##########################################################################
1155 # Table 6.5 (RESULTS): Win/Tie analysis of bias, variance, systematic and
1156 # variance effects for random forests, including random rotation forests.
1157 ##########################################################################
1158 # ------------------------------------------------------------------------
1159 majVote <- function(x){names(which.max(table(x)))}
1160 nTrain <- 400
1161 nTest <- 1000
1162
1163 # MAIN EXPERIMENT FUNCTIONS
1164 runBiasVarSimulation <- function(trainingSets, simTest, paraGrid, BayesPreds){
1165
1166        # linear combination oblique trees
1167        sim.obliqueRRFrf <- simulateBiasVarDecomp(trainingSets=trainingSets,
                   simTest=simTest,
1168                                                  model="rf", paraGrid =
                                                       paraGrid, BayesPreds =
                                                       BayesPreds)
1169
1170        # randomised oblique trees using logistic splits
1171        sim.obliqueRRFlog <- simulateBiasVarDecomp(trainingSets=trainingSets,
                   simTest=simTest,
1172                                                   model="log", paraGrid =
                                                        paraGrid, BayesPreds =
                                                        BayesPreds)
1173        list(sim.obliqueRRFlog, sim.obliqueRRFrf)
1174 }
1175
1176
1177 simulateBiasVarDecomp <- function(trainingSets, simTest, model, paraGrid,
       BayesPreds, ...){
1178
1179        numOfExp <- 100
1180        # train models and make predictions
1181        BVpreds <- matrix(0, nrow=numOfExp, ncol=nTest)
1182        var.T <- NULL
1183        var <- NULL
1184        bias <- NULL
1185        VE <- NULL
1186        SE <- NULL
1187        misclassError <- NULL
1188        p <- ncol(simTest)
1189        C <- as.numeric(simTest$classes)
1190
1191        # train models
1192        for(j in 1:numOfExp){
1193              x <- trainingSets[[j]][,-p]
1194              y <- trainingSets[[j]][,p]
1195              mod <- RRotF(x=x, y=y, K=paraGrid[1], L=200, mtry=paraGrid[2],
                        model=model)
1196              BVpreds[j,] <- predict(mod, simTest[,-p])+1
1197              print(paste("Method: ", model, ", Iter: ", j, " out of ", numOfExp
                        ))
1198        }
1199
1200        # James (2003) decomposition estimates
1201        BayesClassifier <- BayesPreds
1202        majVoteClassifier <- apply(BVpreds, 2, function(x)majVote(x))
1203        var.T <- mean(BayesClassifier != C)
```

```
1204        var <- mean(apply(BVpreds, 1, function(x) mean(x != majVoteClassifier)))
1205        bias <- mean(majVoteClassifier != BayesClassifier)
1206        VE <- mean(apply(BVpreds, 1, function(x) mean(x != C)) - mean(
                majVoteClassifier != C))
1207        SE <- mean(majVoteClassifier != C) - mean(BayesClassifier != C)
1208        meanError <- mean(apply(BVpreds, 1, function(x){ mean(x != C) }))
1209
1210        # plot bias and variance and systematic effect and variance effect
1211        vb <- c(meanError, var.T, SE, VE, bias, var)
1212        bar <- factor(c(1,2,3,4,5,6))
1213        type <- c("Error", "Bayes Error", "Systematic Effect", "Variance Effect"
                , "Bias", "Variance")
1214        modelName <- rep(model, 6)
1215        biasVarPlotData <- data.frame(vb=vb, Decomposition=type, bar=bar, model=
                modelName)
1216        biasVarPlotData
1217 }
1218
1219 ######################
1220 # SETUP 1: corr=0.9
1221 ######################
1222 # simluating training data sets
1223 trainingSets <- list()
1224 for(i in 1:100){
1225        set.seed(i+1)
1226        train <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
1227                         firstonly=c(FALSE), nsamples=400, response="binary")
1228        train <- train$data
1229        train$classes <- train$outcome
1230        trainingSets[[i]] <- train[,-16]
1231 }
1232
1233 # simulate test data set
1234 set.seed(1)
1235 test <- simData(nvars=c(15), cors=c(0.9), associations=c(1),
1236                 firstonly=c(FALSE), nsamples=1000, response="binary")
1237 testData <- test$data
1238 testData$classes <- testData$outcome
1239 simTest <- testData[,-16]
1240
1241 # run simulation and plot data
1242 setup1TuneVals <- readRDS("setup1ResultsAR.rda")[[2]]
1243 setup1ParaGrid <- c(median(setup1TuneVals[[3]][,2]), median(setup1TuneVals
        [[4]]))
1244 BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
1245 setup1Results <- runBiasVarSimulation(trainingSets, simTest, setup1ParaGrid,
        BayesClasses)
1246 saveRDS(setup1Results, "obliqueRRFsetup1Results.rda")
1247 ######################
1248 # SETUP 2: corr=0.5
1249 ######################
1250 # simluating training data sets
1251 trainingSets <- list()
1252 for(i in 1:100){
1253        set.seed(i+1)
1254        train <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
1255                         firstonly=c(FALSE), nsamples=400, response="binary")
1256        train <- train$data
1257        train$classes <- train$outcome
1258        trainingSets[[i]] <- train[,-16]
1259 }
1260
1261 # simulate test data set
1262 set.seed(1)
1263 test <- simData(nvars=c(15), cors=c(0.5), associations=c(1),
1264                 firstonly=c(FALSE), nsamples=1000, response="binary")
```

```
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
setup2TuneVals <- readRDS("setup2ResultsAR.rda")[[2]]
setup2ParaGrid <- c(median(setup2TuneVals[[3]][,2]), median(setup2TuneVals
    [[4]]))
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup2Results <- runBiasVarSimulation(trainingSets, simTest, setup2ParaGrid,
    BayesClasses)
saveRDS(setup2Results, "obliqueRRFsetup2Results.rda")
#####################
# SETUP 3: corr=0.1
#####################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0.1), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
setup3TuneVals <- readRDS("setup3ResultsAR.rda")[[2]]
setup3ParaGrid <- c(median(setup3TuneVals[[3]][,2]), median(setup3TuneVals
    [[4]]))
BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
setup3Results <- runBiasVarSimulation(trainingSets, simTest, setup3ParaGrid,
    BayesClasses)
saveRDS(setup3Results, "obliqueRRFsetup3Results.rda")
###################
# SETUP 4: corr=0
###################
# simluating training data sets
trainingSets <- list()
for(i in 1:100){
      set.seed(i+1)
      train <- simData(nvars=c(15), cors=c(0), associations=c(1),
                       firstonly=c(FALSE), nsamples=400, response="binary")
      train <- train$data
      train$classes <- train$outcome
      trainingSets[[i]] <- train[,-16]
}

# simulate test data set
set.seed(1)
test <- simData(nvars=c(15), cors=c(0), associations=c(1),
                firstonly=c(FALSE), nsamples=1000, response="binary")
testData <- test$data
testData$classes <- testData$outcome
simTest <- testData[,-16]

# run simulation and plot data
setup4TuneVals <- readRDS("setup4ResultsAR.rda")[[2]]
```

```
1327 setup4ParaGrid <- c(median(setup4TuneVals[[3]][,2]), median(setup4TuneVals
         [[4]]))
1328 BayesClasses <- as.numeric(factor(ifelse(test$probs > 0.5, 1, 0)))
1329 setup4Results <- runBiasVarSimulation(trainingSets, simTest, setup4ParaGrid,
         BayesClasses)
1330 saveRDS(setup4Results, "obliqueRRFsetup4Results.rda")
1331 ####################
1332 # Setup 5: J = 2
1333 ####################
1334 # simluating training data sets
1335 q <- 0.15
1336 simData1 <- generateMeasedata(J=2)
1337 trainingSets <- simData1[[1]]
1338 simTest <- simData1[[2]]
1339 # run simulation and plot data
1340 setup5TuneVals <- readRDS("setup5ResultsAR.rda")[[2]]
1341 setup5ParaGrid <- c(median(setup5TuneVals[[3]][,2]), median(setup5TuneVals
         [[4]]))
1342 BayesClasses <- as.numeric(factor(apply(simTest[,-31], 1, function(x) 1*(0.5<(
         q+(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1343 setup5Results <- runBiasVarSimulation(trainingSets, simTest, setup5ParaGrid,
         BayesClasses)
1344 saveRDS(setup5Results, "obliqueRRFsetup5ResultsAR.rda")
1345 ####################
1346 # Setup 6: J = 5
1347 ####################
1348 # simluating training data sets
1349 simData1 <- generateMeasedata(J=5)
1350 trainingSets <- simData1[[1]]
1351 simTest <- simData1[[2]]
1352 # run simulation and plot data
1353 setup6TuneVals <- readRDS("setup6ResultsAR.rda")[[2]]
1354 setup6ParaGrid <- c(median(setup6TuneVals[[3]][,2]), median(setup6TuneVals
         [[4]]))
1355 BayesClasses <- as.numeric(factor(apply(simTest[,-31], 1, function(x) 1*(0.5<(
         q+(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1356 setup6Results <- runBiasVarSimulation(trainingSets, simTest, setup6ParaGrid,
         BayesClasses)
1357 saveRDS(setup6Results, "obliqueRRFsetup6ResultsAR.rda")
1358 ####################
1359 # Setup 7: J = 15
1360 ####################
1361 # simluating training data sets
1362 simData1 <- generateMeasedata(J=15)
1363 trainingSets <- simData1[[1]]
1364 simTest <- simData1[[2]]
1365 # run simulation and plot data
1366 setup7TuneVals <- readRDS("setup7ResultsAR.rda")[[2]]
1367 setup7ParaGrid <- c(median(setup7TuneVals[[3]][,2]), median(setup7TuneVals
         [[4]]))
1368 BayesClasses <- as.numeric(factor(apply(simTest[,-31], 1, function(x) 1*(0.5<(
         q+(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1369 setup7Results <- runBiasVarSimulation(trainingSets, simTest, setup7ParaGrid,
         BayesClasses)
1370 saveRDS(setup7Results, "obliqueRRFsetup7ResultsAR.rda")
1371 ####################
1372 # Setup 8: J = 20
1373 ####################
1374 # simluating training data sets
1375 simData1 <- generateMeasedata(J=20)
1376 trainingSets <- simData1[[1]]
1377 simTest <- simData1[[2]]
1378 # run simulation and plot data
1379 setup8TuneVals <- readRDS("setup8ResultsAR.rda")[[2]]
1380 setup8ParaGrid <- c(median(setup8TuneVals[[3]][,2]), median(setup8TuneVals
         [[4]]))
```

```
1381 BayesClasses <- as.numeric(factor(apply(simTest[,-31], 1, function(x) 1*(0.5<(
         q+(1-2*q)*1*(sum((x[1:J]))>(J/2)))))))
1382 setup8Results <- runBiasVarSimulation(trainingSets, simTest, setup8ParaGrid,
         BayesClasses)
1383 saveRDS(setup8Results, "obliqueRRFsetup8ResultsAR.rda")
1384 ############################
1385 # twonorm simulation data
1386 ############################
1387 # simluating training data sets
1388 trainingSets <- list()
1389 for(i in 1:100){
1390       set.seed(i+1)
1391       train <- mlbench.twonorm(400, d=20)
1392       train <- as.data.frame(train)
1393       trainingSets[[i]] <- train
1394 }
1395
1396 # simulate test data set
1397 set.seed(1)
1398 test <- mlbench.twonorm(1000, d=20)
1399 testFrame <- as.data.frame(test)
1400 simTest <- testFrame
1401
1402 # run simulation and plot data
1403 twonormTuneVals <- readRDS("twonormResultsAR.rda")[[2]]
1404 twonormParaGrid <- c(median(twonormTuneVals[[3]][,2]), median(twonormTuneVals
         [[4]]))
1405 twonormResults <- runBiasVarSimulation(trainingSets, simTest, twonormParaGrid,
          bayesclass(test))
1406 saveRDS(twonormResults, "obliqueRRFtwonormResults.rda")
1407 ##############################
1408 # threenorm simulation data
1409 ##############################
1410 # simluating training data sets
1411 trainingSets <- list()
1412 for(i in 1:100){
1413       set.seed(i+1)
1414       train <- mlbench.threenorm(400, d=20)
1415       train <- as.data.frame(train)
1416       trainingSets[[i]] <- train
1417 }
1418
1419 # simulate test data set
1420 set.seed(1)
1421 test <- mlbench.threenorm(1000, d=20)
1422 testFrame <- as.data.frame(test)
1423 simTest <- testFrame
1424
1425 # run simulation and plot data
1426 threenormTuneVals <- readRDS("threenormResultsAR.rda")[[2]]
1427 threenormParaGrid <- c(5, 4)
1428 threenormResults <- runBiasVarSimulation(trainingSets, simTest,
         threenormParaGrid, bayesclass(test))
1429 saveRDS(threenormResults, "obliqueRRFthreenormResultsAR.rda")
1430 ##############################
1431 # ringnorm simulation data
1432 ##############################
1433 # simluating training data sets
1434 trainingSets <- list()
1435 for(i in 1:100){
1436       set.seed(i+1)
1437       train <- mlbench.ringnorm(400, d=20)
1438       train <- as.data.frame(train)
1439       trainingSets[[i]] <- train
1440 }
1441
```

```
# simulate test data set
set.seed(1)
test <- mlbench.ringnorm(1000, d=20)
testFrame <- as.data.frame(test)
simTest <- testFrame

# run simulation and plot data
ringnormTuneVals <- readRDS("ringnormResultsAR.rda")[[2]]
ringnormParaGrid <- c(median(ringnormTuneVals[[3]][,2]), median(
    ringnormTuneVals[[4]]))
ringnormResults <- runBiasVarSimulation(trainingSets, simTest,
    ringnormParaGrid, bayesclass(test))
saveRDS(ringnormResults, "obliqueRRFringnormResultsAR.rda")

#########################################################################
# Make table for rotation random forest bias variance results
#########################################################################
res1 <- readRDS("obliqueRRFtwonormResults.rda")
res2 <- readRDS("obliqueRRFthreenormResultsAR.rda")
res3 <- readRDS("obliqueRRFringnormResultsAR.rda")
res5 <- readRDS("obliqueRRFsetup1Results.rda")
res6 <- readRDS("obliqueRRFsetup2Results.rda")
res7 <- readRDS("obliqueRRFsetup3Results.rda")
res8 <- readRDS("obliqueRRFsetup4Results.rda")
res9 <- readRDS("obliqueRRFsetup5ResultsAR.rda")
res10 <- readRDS("obliqueRRFsetup6ResultsAR.rda")
res11 <- readRDS("obliqueRRFsetup7ResultsAR.rda")
res12 <- readRDS("obliqueRRFsetup8ResultsAR.rda")
ORRFresList <- list(res1, res2, res3, res5, res6, res7, res8,
                    res9, res10, res11, res12)

# rbind results
for(i in 1:length(ORRFresList)){
      temp <- ORRFresList[[i]]
      if(i < 4 || i > 7){
            ORRFresList[[i]] <- rbind(temp[[3]], temp[[1]])
      } else {
            ORRFresList[[i]] <- rbind(temp[[2]], temp[[1]])
      }
}

# import old results
res1 <- readRDS("twonormResultsAR.rda")$results
res2 <- readRDS("threenormResultsAR.rda")
res3 <- readRDS("ringnormResultsAR.rda")$results
res5 <- readRDS("setup1ResultsAR.rda")$results
res6 <- readRDS("setup2ResultsAR.rda")$results
res7 <- readRDS("setup3ResultsAR.rda")$results
res8 <- readRDS("setup4ResultsAR.rda")$results
res9 <- readRDS("setup5ResultsAR.rda")$results
res10 <- readRDS("setup6ResultsAR.rda")$results
res11 <- readRDS("setup7ResultsAR.rda")$results
res12 <- readRDS("setup8ResultsAR.rda")$results
resList <- list(res1, res2, res3, res5, res6, res7, res8,
                res9, res10, res11, res12)

# combine old with new results
for(i in 1:length(ORRFresList)){
      temp <- resList[[i]]
      ORRFresList[[i]] <- rbind(temp, ORRFresList[[i]])
}

# make tables for thesis
tableFinal <- NULL
for(k in 1:length(resList)){
      res <- ORRFresList[[k]]
```

```
1506        splitDat <- split(res, res$model)
1507        cname <- unique(res$model)
1508        rname <- unique(res$Decomposition)
1509        tableFrame <- matrix(0, nrow=length(rname), ncol=length(cname))
1510        for(i in 1:length(splitDat)){
1511              tableFrame[,i] <- splitDat[[i]]$vb
1512        }
1513        rownames(tableFrame) <- paste(k, rname)
1514        colnames(tableFrame) <- cname
1515        tableFinal <- rbind(tableFinal, tableFrame)
1516 }
1517 tableFinal <- as.data.frame(tableFinal)
1518
1519 # makes tables
1520 n <- nrow(tableFinal)
1521 errorTable <- tableFinal[seq(1, n, by=6),]
1522 SEtable <- tableFinal[seq(3, n, by=6),]
1523 VEtable <- tableFinal[seq(4, n, by=6),]
1524 biasTable <- tableFinal[seq(5, n, by=6),]
1525 varTable <- tableFinal[seq(6, n, by=6),]
1526 compTableList <- list(errorTable, SEtable, VEtable, biasTable, varTable)
1527 compPVals <- list()
1528
1529 # compute omnibus p-vals
1530 library(scmamp)
1531 for(i in 1:length(compTableList)){
1532        compPVals[[i]] <- friedmanAlignedRanksTest(compTableList[[i]][,-5])
1533 }
1534
1535 # compute post-hoc p-vals
1536 postPVals <- list()
1537 for(i in 1:length(compTableList)){
1538        postPVals[[i]] <- postHocTest(compTableList[[i]][,-5], test="aligned
           ranks",
1539                                      correct="finner", control=5)
1540 }
1541
1542 # create latex table
1543 stargazer(tableFinal, summary = FALSE)
1544 # ------------------------------------------------------------------------
```

D.7 Chapter 7 Code: Comparing Random Forests

R Code D.7: Source Code: Comparing Random Forests

```
#########################################
# CHAPTER 7: Comparing Random Forests
#########################################

# Check for missing packages and install if missing
list.of.packages <- c("MASS","dplyr","latex2exp", "mlbench", "ggplot2", "caret", "doSNOW", "lattice",
                      "obliqueRF", "stargazer", "rotationForest", "randomForest",
                      "scmamp", "surv2sampleComp", "ElemStatLearn", "hmeasure")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load required packages
load <- lapply(list.of.packages, require, character.only = TRUE)

# required packages from bioconductor for scmamp package
source("https://bioconductor.org/biocLite.R")
biocLite("graph")
n
biocLite("Rgraphviz")
n

# download and load random rotation forests package
if("RRotF" %in% installed.packages()[,"Package"] == FALSE){
      library(devtools)
      install_github("arnupretorius/RRotF")
}
library(RRotF)

#############################################################################
# Table 7.1: Available software for random forests in the R programming
# language
#############################################################################
# ---------------------------------------------------------------------------
data <- read.csv("RFvariantsData.csv")
data <- arrange(data, year)
psals <- paste(data[,2], " (", data[,3], ")", sep="")
pks <- c("unavailable", "unavailable", "ipred", "unavailable", "unavailable",
         "unavailable", "unavailable", "unavailable", "randomForest",
         "unavailable", "unavailable", "caret", "unavailable", "extraTrees",
         "unavailable", "unavailable", "unavailable", "unavailable", "unavailable",
         "party", "unavailable", "unavailable", "obliqueRF", "unavailable",
         "RRF", "wsrf", "unavailable", "unavailable", "RRF", "RRF", "unavailable",
         "unavailable", "unavailable", "unavailable", "unavailable", "unavailable",
         "unavailable")
softTable <- data.frame("Proposals"=psals, "R package"=pks)
# make latex table
stargazer(softTable, summary = FALSE, rownames=FALSE)
# ---------------------------------------------------------------------------
#############################################################################
# Table B.1: Papers considered in the meta-analysis. (Appendix B)
#############################################################################
# ---------------------------------------------------------------------------
```

```
data <- read.csv("RFComparisonsData.csv")
lop <- arrange(data, year) %>% select(paper_title, author, year, journal) %>%
      group_by(paper_title) %>% distinct(.keep_all=TRUE)
lop <- as.data.frame(lop)
colnames(lop) <- c("Paper title", "Author(s)", "Year", "Journal")
stargazer(lop, summary = FALSE, rownames = FALSE)
#------------------------------------------------------------------------
##########################################################################
# Table 7.3: Algorithm performance measures for binary classification
##########################################################################
#------------------------------------------------------------------------
measures <- c("Error", "Accuracy", "Sensitivity", "Specificity", "Precision",
              "Kappa", "AUC", "F-score", "H-measure")
formula <- 1:9
aps <- c("Balanced data", "Balanced data", "Skew data/minority class", "Skew
    data",
         "Skew data", "Skew data", "Not recommended", "Skew data/minority
             class",
         "Balanced data/skew data")
measFrame <- data.frame("Performance measure"=measures, "Calculation"=formula,
                        "Appropriate scenario"=aps)
stargazer(measFrame, summary = FALSE, rownames = FALSE)
#------------------------------------------------------------------------
##########################################################################
# Figure 7.2: Performance estimation method used in the papers considered in
# the meta-analysis.
##########################################################################
#------------------------------------------------------------------------
# load data
data <- read.csv("RFComparisonsData.csv")
evalMeth <- data %>% select(paper_title, evaluation)
evalMeth <- unique(evalMeth)
evalMeth <- as.data.frame(evalMeth %>% count(evaluation))
evalMeth <- evalMeth[order(evalMeth$n, decreasing = TRUE),]

# mark which estimation methods are not "reliable"
notIndex <- c(4, 9, 22, 27)
grp <- rep("Reliable", 27)
grp[notIndex] <- "Not reliable"
evalMeth$grp <- grp

# plot estimation methods used
ggplot(evalMeth, aes(x=evaluation, y=n, fill=grp)) + geom_bar(stat="identity")
    +
      scale_x_discrete(limits=evalMeth$evaluation) +
      scale_fill_manual(name="" ,values=c("darkgreen", "skyblue")) +
      theme_bw() + ylab("#Papers") + xlab("Estimation method") +
      theme(legend.position = c(0.9,0.7), axis.text.x = element_text(angle =
          30, vjust = 1, hjust = 1))
#------------------------------------------------------------------------
##########################################################################
# Figure 7.5: Reported error rates for Breimans Forest-RI on the top ten
# most popular data sets used in papers.
##########################################################################
#------------------------------------------------------------------------
# load data
data <- read.csv("RFComparisonsData.csv")

# compute top used data sets across papers
ds <- data %>% select(paper_title, dataset)
allDS <- unique(ds$dataset)
allDSCount <- rep(0, length(allDS))
dsSplit <- split(ds, ds$paper_title)
for(i in 1:length(dsSplit)){
      dsPerPaper <-  unique(dsSplit[[i]][,2])
      for(j in 1:length(dsPerPaper)){
```

```
115             index <- which(allDS == dsPerPaper[j])
116             allDSCount[index] <- allDSCount[index] + 1
117         }
118 }
119 dsFrame <- data.frame(dataset=allDS, freqUsed=allDSCount)
120 dsFrame <- dsFrame[order(dsFrame$freqUsed, decreasing = TRUE),]
121
122 # get dataset characteristics
123 dataChar <- data %>% select(dataset, dataset_size, num_inputs, classes) %>%
      distinct(.keep_all=TRUE)
124 dataChar <- merge(dsFrame, dataChar)
125 dataChar <- dataChar[order(dataChar$freqUsed, decreasing = TRUE),]
126
127 # plot variatability of RF on above data sets
128 rfDataSets <- factor(unique(dataChar$dataset)[1:10])
129 keepIndex <- NULL
130 count <- 1
131 for(i in 1:nrow(data)){
132         if(data[i,]$method == "rf" && data[i,]$dataset %in% rfDataSets){
133             keepIndex[count] <- i
134             count <- count + 1
135         }
136 }
137 rfCompData <- data[keepIndex,] %>% select(paper_title, dataset, method, error)
138 rfCompData <- rfCompData[order(rfCompData$dataset),]
139 # remove other lymphoma
140 lympRemove <- NULL
141 count <- 1
142 for(i in 1:nrow(rfCompData)){
143         if(rfCompData[i,]$dataset == "lymphoma" && rfCompData[i,]$error > 3){
144             lympRemove[count] <- i
145             count <- count + 1
146         }
147 }
148 rfCompData <- rfCompData[-lympRemove,]
149 rfCompData <- rfCompData[-101, ] # remove gross outlier in: On extreme pruning
      (possibly reported error instead of acc)
150 # plot errors
151 ggplot(rfCompData, aes(y=error, x=dataset)) + geom_boxplot(fill="skyblue",
      outlier.colour = "red")+
152         theme_bw() + ylab("Reported error rates") + xlab("Benchmark data set")+
153         scale_x_discrete(limits=unique(dataChar$dataset)[1:10])+
154         theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1))
155
156 # outlier breast cancer: On extreme pruning (Possibly not the same breast
      cancer dataset)
157 # outlier glass: Tripoli et al. paper
158 # outlier sonar: On extreme pruning (possibly reported error instead of acc)
159 # ------------------------------------------------------------------------
160 ##########################################################################
161 # In text: Omnibus p-val for Forest-RI over different papers
162 ##########################################################################
163 # ------------------------------------------------------------------------
164 # test between random forests from different papers
165 lop <- arrange(data, paper_title) %>% select(paper_title, dataset, method,
      error)
166 lop <- filter(lop, lop$method == "rf")
167 lop$acc <- round(100-lop$error,3)
168
169 # top 10 used data sets for Forest-RI
170 topDataset <- factor(dsFrame[1:15,1])
171 papers <- factor(unique(lop$paper_title))
172
173 # compute all combinations
174 allcomb <- combn(1:15, 10, simplify = FALSE)
175 candidateList <- list()
```

```
canL <- NULL
# find combinations of 15 choose 10 such that largest number of rfs can be
    compared
for (k in 1:length(allcomb)) {
      datasetTop10 <- topDataset[allcomb[[k]]]
      lop10 <- filter(lop, lop$dataset %in% datasetTop10)
      splitlop <- split(lop10, factor(lop10$dataset))
      candidatePapers <- NULL
      check <- 0
      count <- 1
      for(i in 1:length(papers)){
            for(j in 1:length(splitlop)){
                  papersPresent <- factor(splitlop[[j]][,1])
                  if(!(papers[i] %in% papersPresent)){
                        check <- 1
                  }
            }
            if(check == 0){
                  candidatePapers[count] <- as.character(papers[i])
                  count <- count + 1
            } else {
                  check <- 0
            }
      }
      candidateList[[k]] <- candidatePapers
      canL[[k]] <- length(candidatePapers)
}

# find combination sharing the maximum number of datasets
maxIndex <- which(canL == max(canL))[1]
# list of papers
candidateList[[maxIndex]]

# create compare matrix
dsets <- factor(topDataset[allcomb[[maxIndex]]])
filterIndex <- sapply(lop$dataset, function(x){ifelse(x %in% dsets, TRUE,
    FALSE)})
lop <- lop[filterIndex,]
lopSplit <- split(lop, factor(lop$dataset))
compareMat <- matrix(0, nrow=length(dsets), ncol=length(papers))
rownames(compareMat) <- names(lopSplit)
colnames(compareMat) <- papers
for(i in 1:length(lopSplit)){
      for(j in 1:nrow(lopSplit[[i]])){
            compareMat[i, which(papers == as.character(lopSplit[[i]]$paper_
                title[j]))] <- lopSplit[[i]]$acc[j]
      }
}
# prune to include papers containing all top ten data sets
keepIndex <- apply(compareMat, 2, function(x){
      ifelse(length(which(x == 0)) == 0, TRUE, FALSE)
})
rCompareMat <- compareMat[,keepIndex]

# compute omnibus tests (#algorithm < 5)
imanDavenportTest(rCompareMat)
friedmanAlignedRanksTest(rCompareMat)
quadeTest(rCompareMat)
# ------------------------------------------------------------------------------
##############################################################################
# Figure 7.6: Methods used to compare different algorithms over multiple
# data sets in the papers considered for the meta-analysis.
##############################################################################
# ------------------------------------------------------------------------------
# plot evaluation method used
evalsData <- data %>% select(paper_title, comparison) %>% distinct(.keep_all=
```

```
    TRUE)
lims <- unique(evalsData$comparison)[c(1,2,4,3,5,6,7,8)]
ggplot(evalsData, aes(x=comparison)) + geom_bar(fill="darkgreen") +
      xlab("Comparison method") + ylab("#Papers")+
      theme_bw()+
      scale_x_discrete(limits=lims)+
      scale_y_continuous(breaks = seq(0, 20, by = 2))+
      theme(axis.text.x = element_text(angle = 10, vjust = 1, hjust = 1))
# ------------------------------------------------------------------------
################################################################################
# Figure 7.6: Methods used to compare different algorithms over multiple
# data sets in the papers considered for the meta-analysis.
################################################################################
# ------------------------------------------------------------------------
# Redo analyses using omnibus and post-hoc tests
dataSplit <- split(data, factor(data$paper_title))
pvals <- NULL
checkPhTest <- NULL
phTests <- list()

# for each paper build a compare matrix and compute the Ivan-Davenport test p-
    value
for(k in 1:length(dataSplit)){
      lop <- arrange(dataSplit[[k]], dataset) %>% select(dataset, method,
          error)
      lop <- as.data.frame(summarise(group_by(lop, dataset, method), mean(
          error)))
      lop$acc <- round(100-lop$`mean(error)`,3)

      # create compare matrix
      lopSplit <- split(lop, factor(lop$dataset))
      dsets <- unique(lop$dataset)
      methods <- unique(lop$method)
      compareMat <- matrix(0, nrow=length(dsets), ncol=length(methods))
      rownames(compareMat) <- dsets
      colnames(compareMat) <- methods
      for(i in 1:length(lopSplit)){
            for(j in 1:nrow(lopSplit[[i]])){
                  compareMat[i, which(methods == as.character(lopSplit[[i]]$
                      method[j]))] <- lopSplit[[i]]$acc[j]
            }
      }
      pvals[k] <- round(imanDavenportTest(compareMat)[[3]], 3)
      if(!is.na(pvals[k]) && pvals[k] < 0.05 && "rf" %in% colnames(compareMat)
          ){
            phTests[[k]] <- postHocTest(compareMat, test = "friedman",
                                        control = "rf", correct = "finner",
                                            alpha = 0.05)
            checkPhTest[k] <- ifelse(length(which(phTests[[k]]$corrected.pval
                < 0.05)) == 0, TRUE, FALSE)
      }
}

# paper where no significant result was found
omnibusFailed <- which(pvals > 0.05)
PHvsRFFailed <- which(checkPhTest)
nonSigResult <- c(omnibusFailed, PHvsRFFailed)

# plot pvals
pvals[PHvsRFFailed] <- 0.05
grp <- rep("Omnibus: Iman-Davenport", length(pvals))
grp[PHvsRFFailed] <- "Post-hoc: Finner (Forest-RI as control)"
grp <- factor(grp)
pvalData <- data.frame(pv = pvals[-29], Test=grp[-29])
ggplot(pvalData, aes(x=1:nrow(pvalData), y=pv, fill=Test)) + geom_bar(stat="
    identity") +
```

```
296         theme_bw() + xlab("'Papers' (no names given)") +
297         ylab("p-value") + geom_hline(yintercept = 0.05, col="red", linetype="
                dashed") +
298         scale_fill_manual(values=c("darkgreen", "skyblue"))+
299         scale_x_continuous(breaks = seq(from=1,to=34, by=1))+
300         annotate("text", x=0.5, y=0.09, label = "alpha==0.05 ",parse = TRUE)+
301         theme(legend.position=c(0.2,0.8))
302 # ------------------------------------------------------------------------
303 ##################################################
304 # In text: Omnibus p-val for Breiman (2001a)
305 ##################################################
306 # ------------------------------------------------------------------------
307 # Breiman RF paper omnibus p-val = 0.014!
308 # pairwise corrected p-values
309 phTests[[26]]$corrected.pval
310 # ------------------------------------------------------------------------
311 ###################################################################
312 # Figure 7.8: The adjusted ranks for all-round algorithms
313 ###################################################################
314 # ------------------------------------------------------------------------
315 # load data
316 data <- read.csv("RFComparisonsData.csv")
317
318 # remove observations that are not "reliable"
319 data <- filter(data, data$evaluation != "OOB")
320 data <- filter(data, data$evaluation != "1 run")
321 data <- filter(data, data$evaluation != "3-fold cv")
322 data <- filter(data, !is.na(data$evaluation))
323
324 # split between allround situations and high-dimensional situations
325 allround <- filter(data, data$situation == "allround")
326 HD <- filter(data, data$situation == "HD" | data$situation == "select-HD")
327
328 # ALLROUND methods
329 lop <- arrange(allround, dataset) %>% select(dataset, method, error)
330 lop <- as.data.frame(summarise(group_by(lop, dataset, method), mean(error)))
331 lop$acc <- round(100-lop$`mean(error)`,3)
332
333 # create compare matrix
334 lopSplit <- split(lop, factor(lop$dataset))
335 dsets <- unique(lop$dataset)
336 methods <- unique(lop$method)
337 compareMat <- matrix(0, nrow=length(dsets), ncol=length(methods))
338 rownames(compareMat) <- dsets
339 colnames(compareMat) <- methods
340 for(i in 1:length(lopSplit)){
341         for(j in 1:nrow(lopSplit[[i]])){
342                 compareMat[i, which(methods == as.character(lopSplit[[i]]$method[j
                        ]))] <- lopSplit[[i]]$acc[j]
343         }
344 }
345 #remove rare data sets
346 removeRowIndex <- apply(compareMat, 1, function(x){
347         ifelse(length(which(x != 0)) < 3, 1, 0)
348 })
349 removeColIndex <- apply(compareMat, 2, function(x){
350         ifelse(length(which(x != 0)) < 10, 1, 0)
351 })
352 removeRowIndex <- which(removeRowIndex == 1)
353 removeColIndex <- which(removeColIndex == 1)
354 compareMat <- compareMat[-removeRowIndex,-removeColIndex]
355
356 # compute nomial ranks
357 rankMat <- apply(compareMat, 1, function(x){
358         index <- which(x != 0)
359         rankVec <- rank(-x[index], ties.method = "average")
```

```
        x[index] <- rankVec/length(rankVec)
        x
})
rankMat <- t(rankMat)

# adjust for number of datasets used
rankMat <- apply(rankMat, 2, function(x){
        prop <- length(which(x == 0))/length(x)
        x*prop
})

# compute average rank per method
avgRanks <- apply(rankMat, 2, function(x){
        index <- which(x != 0)
        mean(x[index])
})

# scale ranks
range2 = length(avgRanks) - 1
avgRanksStand = (avgRanks*range2) + 1

# sorted ranks
sortAvgRanks <- sort(avgRanksStand)

# plot sorted ranks
plotData <- data.frame(rank=sortAvgRanks, names=names(sortAvgRanks))
ggplot(plotData, aes(x=names, y=rank)) +
        geom_bar(stat="identity", fill="orange") + ylab("Adjusted Rank") +
        xlab("Algorithm") +
        scale_x_discrete(limits=names(sortAvgRanks))+ theme_bw()+
        theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))+
        ggtitle("All-round algorithms")
# ------------------------------------------------------------------------------
################################################################################
# Figure 7.9: Results from comparing the top five all-round algorithms: Top
# left: Kernel (Gaussian) density estimations based on accuracy. Top right:
# Adjusted p-value matrix using the Shaffer static approach. Bottom:
# Pairwise comparisons plot.
################################################################################
# ------------------------------------------------------------------------------
# choose top 5 algorithms
top5Algs <- names(sortAvgRanks)[1:5]

# find data sets associated with each rf
rfsAlgs <- top5Algs
algsDatasets <- list()
algsDatLen <- NULL
temp <- NULL
for(i in 1:length(rfsAlgs)){
        for(j in 1:length(lopSplit)){
                if(rfsAlgs[i] %in% lopSplit[[j]][,2]){
                        temp <- c(temp, as.character(lopSplit[[j]][1,1]))
                }
        }
        algsDatasets[[i]] <- temp
        algsDatLen[i] <- length(temp)
        temp <- NULL
}
# (all algs come from same article => same 24 datasets)
# define reduced comare matrix
datasets <- algsDatasets[[2]]
cAlgs <- top5Algs
rowIndex <- sapply(rownames(compareMat) , function(x) x %in% datasets)
colIndex <- sapply(colnames(compareMat), function(x) x %in% cAlgs)
rCompareMat <- compareMat[rowIndex, colIndex]

```

```
426 # plot densities
427 plotDensities(rCompareMat) + xlab("Accuracy") + theme_bw() + theme(legend.
        position=c(0.2, 0.7))
428
429 # perform Iman-Devenport test
430 imanDavenportTest(rCompareMat)
431 # (significant dfference found => perform post-hoc test)
432
433 # perform Shaffer's static test
434 pvalsShaffer <- postHocTest(rCompareMat, test = "friedman", use.rank=TRUE,
        correct="shaffer")
435
436 # plot p-values and hypothesis tests
437 # Shaffer
438 plotPvalues(pvalsShaffer$corrected.pval) + ggtitle("Shaffer's static")
439 drawAlgorithmGraph(pvalsShaffer$corrected.pval, pvalsShaffer$summary, font.
        size = 5)
440 # ------------------------------------------------------------------------
441 ##########################################################################
442 # Figure 7.10: Results from comparing the top five high-dimensional
443 # algorithms: Top left: Adjusted ranks. Top right: Kernel (Gaussian)
444 # density estimations based on accuracy. Bottom left: Adjusted p-value
445 # matrix using the Shaffer static approach. Bottom right: Pairwise
446 # comparisons plot.
447 ##########################################################################
448 # ------------------------------------------------------------------------
449 # HD methods
450 lopHD <- arrange(HD, dataset) %>% select(dataset, method, error)
451 lopHD <- as.data.frame(summarise(group_by(lopHD, dataset, method), mean(error)
        ))
452 lopHD$acc <- round(100-lopHD$`mean(error)`,3)
453
454 # create compare matrix
455 lopHDSplit <- split(lopHD, factor(lopHD$dataset))
456 dsets <- factor(unique(lopHD$dataset))
457 methods <- factor(unique(lopHD$method))
458 compareMat <- matrix(0, nrow=length(dsets), ncol=length(methods))
459 rownames(compareMat) <- dsets
460 colnames(compareMat) <- methods
461 for(i in 1:length(lopHDSplit)){
462         for(j in 1:nrow(lopHDSplit[[i]])){
463                 compareMat[i, which(methods == as.character(lopHDSplit[[i]]$method
                        [j]))] <- lopHDSplit[[i]]$acc[j]
464         }
465 }
466 #remove rare data sets
467 removeRowIndex <- apply(compareMat, 1, function(x){
468         ifelse(length(which(x != 0)) < 3, 1, 0)
469 })
470 #remove algorithms fitted to only a very small number of data sets
471 removeColIndex <- apply(compareMat, 2, function(x){
472         ifelse(length(which(x != 0)) < 5, 1, 0)
473 })
474 removeRowIndex <- which(removeRowIndex == 1) #(none found)
475 removeColIndex <- which(removeColIndex == 1)
476 compareMat <- compareMat[, -removeColIndex]
477
478 # compute nomial ranks
479 rankMat <- apply(compareMat, 1, function(x){
480         index <- which(x != 0)
481         rankVec <- rank(-x[index], ties.method = "average")
482         x[index] <- rankVec/length(rankVec)
483         x
484 })
485 rankMat <- t(rankMat)
486
```

```
# adjust for number of datasets used
rankMat <- apply(rankMat, 2, function(x){
        prop <- length(which(x == 0))/length(x)
        if(prop == 0){
                prop <- 1/(length(x)+1)
        }
        x*prop
})

# compute average rank per method
avgRanks <- apply(rankMat, 2, function(x){
        index <- which(x != 0)
        mean(x[index])
})

# scale ranks
range2 = length(avgRanks) - 1
avgRanksStand = (avgRanks*range2) + 1

# sorted ranks
sortAvgRanks <- sort(avgRanksStand)

# plot sorted ranks
plotData <- data.frame(rank=sortAvgRanks, names=names(sortAvgRanks))
ggplot(plotData, aes(x=names, y=rank)) +
        geom_bar(stat="identity", fill="orange") + ylab("Adjusted Rank") +
        xlab("Algorithm") +
        scale_x_discrete(limits=names(sortAvgRanks))+ theme_bw()+
        theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))+
        ggtitle("High-dimensional algorithms")

# choose top 5 algorithms
top5Algs <- names(sortAvgRanks)[1:5]

# find data sets associated with each rf
rfsAlgs <- top5Algs
algsDatasets <- list()
algsDatLen <- NULL
temp <- NULL
for(i in 1:length(rfsAlgs)){
        for(j in 1:length(lopHDSplit)){
                if(rfsAlgs[i] %in% lopHDSplit[[j]][,2]){
                        temp <- c(temp, as.character(lopHDSplit[[j]][1,1]))
                }
        }
        algsDatasets[[i]] <- temp
        algsDatLen[i] <- length(temp)
        temp <- NULL
}

# define reduced comare matrix
datasets <- intersect(intersect(algsDatasets[[1]], algsDatasets[[2]]),
    algsDatasets[[3]])
cAlgs <- top5Algs
rowIndex <- sapply(rownames(compareMat) , function(x) x %in% datasets)
colIndex <- sapply(colnames(compareMat), function(x) x %in% cAlgs)
rCompareMat <- compareMat[rowIndex, colIndex]

# plot densities
plotDensities(rCompareMat) + xlab("Accuracy") + theme_bw() + theme(legend.
    position=c(0.2, 0.7))

# perform Iman-Devenport test
imanDavenportTest(rCompareMat)
# (significant dfference found => perform post-hoc test)

```

```
# perform Shaffer's static test
pvalsShaffer <- postHocTest(rCompareMat, test = "friedman", use.rank=TRUE,
    correct="shaffer")

# plot p-values and hypothesis tests
# Shaffer
plotPvalues(pvalsShaffer$corrected.pval) + ggtitle("Shaffer's static")
drawAlgorithmGraph(pvalsShaffer$corrected.pval, pvalsShaffer$summary, font.
    size = 5)
# ------------------------------------------------------------------------
#####################
# Algorithm: rf-wv3
#####################
# ------------------------------------------------------------------------
exportList <- c("predTestHVDM", "distNew", "HVDM", "dp", "norm_diff", "norm_
    vdm", "npx", "npxc", "Ppxc")

# make predictions for all test cases
predict.HVDM <- function(model, Xtest, k){
      responseVarName <- as.character(model[[21]][[2]])
      Xtest <- Xtest[, !names(Xtest) %in% responseVarName]
      preds <- foreach(i=1:nrow(Xtest), .combine = c, .export = exportList) %
          dopar% {
            predTestHVDM(model, Xtest[i,],
                         model$trainingData[, !names(model$trainingData) %in%
                             ".outcome"],
                         model$trainingData[, ".outcome"], k)
      }
      preds
}

# make prediction for test instance
predTestHVDM <- function(model, new, Xtrain, C, k){
      nn <- distNew(new, Xtrain, C, k)
      classes <- levels(C)
      preds <- predict(model, Xtrain[nn$index,])
      voteCount <- sapply(1:length(classes), function(i) {
            c <- classes[i]
            indexc <- which(preds == c)
            sum(1/nn$dists[indexc])
      })
      classes[which(voteCount == max(voteCount))]
}

# find k nearest neighbours to new instance
distNew <- function(new, Xtrain, C, k){
      N <- nrow(Xtrain)
      distVec <- sapply(1:N, function(i) {
            HVDM(new, Xtrain[i,], Xtrain, C)
      })
      indexMat <- data.frame(dists=distVec, index=1:N)
      indexMat <- indexMat[order(indexMat$dists),]
      return(indexMat[1:k,])
}

# compute HVDM between obs x and y
HVDM <- function(x, y, Xtrain, C){
      P <- ncol(Xtrain)
      distVar <- sapply(1:P, function(i) {
            dp(x[i], y[i], Xtrain[,i], C)^2
      })
      dist <- sum(distVar)
      sqrt(dist)
}

# compute distance between x and y on var p using HVDM
```

```
dp <- function(x, y, p, C){
        if(class(p) != "numeric" && class(p) != "integer"){
                if(length(which(p == x)) == 0 || length(which(p == y)) == 0){
                        return(1)
                } else {
                        norm_vdm(x, y, p, C)
                }
        } else {
                norm_diff(x, y, p)
        }
}

# compute the normalized diff for x and y on var p [linear variable]
norm_diff <- function(x, y, p){
        abs(x - y)/4*sd(p)
}

# compute normalized vdm for x and y on var p [nominal variable]
norm_vdm <- function(x, y, p, C){
        vdmTerm <- sapply(1:length(levels(C)), function(i) {
                c <- levels(C)[i]
                abs(Ppxc(p, x, C, c) - Ppxc(p, y, C, c))^2
        })
        varDist <- sum(vdmTerm)
        sqrt(varDist)
}

# compute number of obs in training set with value x for var p
npx <- function(p, x){
        length(which(p == x))
}
# compute number of obs in training set with value x for var p and class c
npxc <- function(p, x, C, c){
        length(intersect(which(p == x), which(C == c)))
}
# compute conditional probability of class c given value of x for var p
Ppxc <- function(p, x, C, c){
        npxc(p, x, C, c)/npx(p, x)
}
# ------------------------------------------------------------------------------
##############################################################################
# Figure 7.11: Prediction time comparisons between Forest-RI and rf-wv3.
# Left: Prediction time as a function of the number of test observations.
# Right: Pre- diction time for twenty test observations for different sizes
# of the input space.
##############################################################################
# ------------------------------------------------------------------------------
# simulate twonorm data for different values of N and p
# experiment with growth in N
genExp1Data <- function(Ngrid){
        tnTrain <- data.frame(mlbench.twonorm(100, d=5))
        tnTest <- list()
        for(i in 1:length(Ngrid)){
                set.seed(i)
                tnTest[[i]] <- data.frame(mlbench.twonorm(Ngrid[i], d=5))
        }
        list(tnTrain, tnTest)
}
# Experiment with growth in p
genExp2Data <- function(pGrid){
        tnTrain <- list()
        tnTest <- list()
        for(i in 1:length(pGrid)){
                set.seed(i)
                tnTrain[[i]] <- data.frame(mlbench.twonorm(100, d=pGrid[i]))
                tnTest[[i]] <- data.frame(mlbench.twonorm(20, d=pGrid[i]))
```

```
678         }
679         list(tnTrain, tnTest)
680 }
681 exp1Data <- genExp1Data(seq(1, 100, by=10))
682 exp2Data <- genExp2Data(seq(2, 20, by=2))
683 # perform N experiment
684 library(caret)
685 rfTime <- NULL
686 rfwv3Time <- NULL
687
688 # Parallel computing
689 cl <- makeCluster(3, type="SOCK")
690 registerDoSNOW(cl)
691
692 # run experiment 1
693 for(i in 1:length(exp1Data[[2]])){
694         start <- Sys.time()
695         rf <- train(classes~., data=exp1Data[[1]], method="rf", trControl=
              tControl, tuneGrid=tgrid)
696         preds <- predict(rf, exp1Data[[2]][[i]])
697         rfTime[i] <- as.numeric(Sys.time() - start)
698         start <- Sys.time()
699         rf <- train(classes~., data=exp1Data[[1]], method="rf", trControl=
              tControl, tuneGrid=tgrid)
700         preds <- predict.HVDM(rf, exp1Data[[2]][[i]], k=nrow(exp1Data[[1]]))
701         rfwv3Time[i] <- as.numeric(Sys.time() - start)
702 }
703
704 rfTime2 <- NULL
705 rfwv3Time2 <- NULL
706 tControl <- trainControl(method="none")
707 # run experiment 2
708 for(i in 1:length(exp2Data[[2]])){
709         start <- Sys.time()
710         rf <- train(classes~., data=exp2Data[[1]][[i]], method="rf", trControl=
              tControl,
711                     tuneGrid=data.frame(mtry=floor(sqrt(ncol(exp2Data[[1]][[i]]))
                        )))
712         preds <- predict(rf, exp2Data[[2]][[i]])
713         rfTime2[i] <- as.numeric(Sys.time() - start)
714         start <- Sys.time()
715         rf <- train(classes~., data=exp2Data[[1]][[i]], method="rf", trControl=
              tControl,
716                     tuneGrid=data.frame(mtry=floor(sqrt(ncol(exp2Data[[1]][[i]]))
                        )))
717         preds <- predict.HVDM(rf, exp2Data[[2]][[i]], k=nrow(exp2Data[[1]][[1]])
              )
718         rfwv3Time2[i] <- as.numeric(Sys.time() - start)
719 }
720
721 # plot comparisons with increase in N
722 time1Data <- data.frame(N=seq(1, 100, by=10), time=c(rfwv3Time, rfTime),
      Algorithm=c(rep("rf-wv3", 10), rep("rf", 10)))
723 ggplot(time1Data, aes(x=N, y=time, col=Algorithm)) + geom_line() + geom_point
      () +
724       theme_bw() + xlab("Number of test observations (p fixed at 5)") + ylab("
              Prediction time (in secs)") +
725       scale_color_manual(values=c("skyblue", "red")) + theme(legend.position=c
              (0.2, 0.7))
726
727 # plot comparisons with increase in p
728 time2Data <- data.frame(p=seq(2, 20, by=2), time=c(rfwv3Time2, rfTime2),
      Algorithm=c(rep("rf-wv3", 10), rep("rf", 10)))
729 ggplot(time2Data, aes(x=p, y=time, col=Algorithm)) + geom_line() + geom_point
      () +
730       theme_bw() + xlab("Number of input variables (N fixed at 20)") + ylab("
```

```
          Prediction time (in secs)") +
731       scale_color_manual(values=c("skyblue", "red")) + theme(legend.position=c
              (0.2, 0.7))
732 # ------------------------------------------------------------------------------
733 ################################################################################
734 # Table 7.5 (RESULTS): Win/Tie analysis of benchmark performances for
735 # oblique random rotation forests.
736 ################################################################################
737 # ------------------------------------------------------------------------------
738 # prepare the data
739 # data from UCI
740
741 ##########################
742 # All-round data sets
743 ##########################
744 # SAheart
745 data("SAheart")
746 colnames(SAheart)[10] <- "response"
747 SAheart$response <- factor(SAheart$response)
748 levels(SAheart$response) <- c("A", "B")
749 # spam
750 data("spam")
751 colnames(spam)[58] <- "response"
752 # Adult
753 adult <- read.csv("adult.data", header = FALSE)
754 colnames(adult)[15] <- "response"
755 # bank
756 bank <- read.table("bank-full.csv", sep=";", header = TRUE)
757 colnames(bank)[17] <- "response"
758 # bank note
759 bankNote <- read.csv("data_banknote_authentication.txt", header = FALSE)
760 colnames(bankNote)[5] <- "response"
761 bankNote$response <- factor(bankNote$response)
762 levels(bankNote$response) <- c("A", "B")
763 # popFailure
764 popFailure <- read.table("pop_failures.dat", header = TRUE)
765 colnames(popFailure)[21] <- "response"
766 popFailure$response <- factor(popFailure$response)
767 levels(popFailure$response) <- c("A", "B")
768 # wisconsin breaset cancer data
769 wdbc <- read.csv("wdbc.data", header = FALSE)
770 colnames(wdbc)[2] <- "response"
771 wdbc$response <- factor(wdbc$response)
772 # Breast cancer
773 data("BreastCancer")
774 BreastCancer <- BreastCancer[,-1]
775 BreastCancer <- BreastCancer[complete.cases(BreastCancer),]
776 colnames(BreastCancer)[10] <- "response"
777 # German credit
778 data("GermanCredit")
779 colnames(GermanCredit)[10] <- "response"
780 # Votes
781 data("HouseVotes84")
782 HouseVotes84 <- HouseVotes84[complete.cases(HouseVotes84),]
783 colnames(HouseVotes84)[1] <- "response"
784 # pima
785 data("PimaIndiansDiabetes")
786 colnames(PimaIndiansDiabetes)[9] <- "response"
787 # Sonar
788 data("Sonar")
789 colnames(Sonar)[61] <- "response"
790
791 # create benchmark dataset list
792 mlbList <- list(adult=adult, bank=bank, bankNote=bankNote, breastCancer=
      BreastCancer, pima=PimaIndiansDiabetes,
793                   germandCredit=GermanCredit, popFailure=popFailure, saheart=
```

```
                    SAheart, sonar=Sonar, spam=spam,
794                 votes=HouseVotes84, wdbc=wdbc)
795
796 # split into training and test sets
797 mlTrainingSets <- list()
798 mlTestingSets <- list()
799 for(i in 1:length(mlbList)){
800       dat <- mlbList[[i]]
801       trainIndex <- createDataPartition(dat$response, p=0.7, list=FALSE)
802       mlTrainingSets[[i]] <- dat[trainIndex,]
803       mlTestingSets[[i]] <- dat[-trainIndex,]
804 }
805
806 # estimate performance
807 perfMeasuresRF <- matrix(0, nrow=length(mlbList), ncol=7)
808 rownames(perfMeasuresRF) <- names(mlbList)
809 colnames(perfMeasuresRF) <- c("Acc", "Sens", "Spec", "Prec", "Kappa", "F", "H"
      )
810 perfMeasuresORRF <- matrix(0, nrow=length(mlbList), ncol=7)
811 rownames(perfMeasuresORRF) <- names(mlbList)
812 colnames(perfMeasuresORRF) <- c("Acc", "Sens", "Spec", "Prec", "Kappa", "F", "
     H")
813 perfMeasuresORRFlog <- matrix(0, nrow=length(mlbList), ncol=7)
814 rownames(perfMeasuresORRFlog) <- names(mlbList)
815 colnames(perfMeasuresORRFlog) <- c("Acc", "Sens", "Spec", "Prec", "Kappa", "F"
     , "H")
816 perfMeasuresRotF <- matrix(0, nrow=length(mlbList), ncol=7)
817 rownames(perfMeasuresRotF) <- names(mlbList)
818 colnames(perfMeasuresRotF) <- c("Acc", "Sens", "Spec", "Prec", "Kappa", "F", "
     H")
819 perfMeasuresORFlog <- matrix(0, nrow=length(mlbList), ncol=7)
820 rownames(perfMeasuresORFlog) <- names(mlbList)
821 colnames(perfMeasuresORFlog) <- c("Acc", "Sens", "Spec", "Prec", "Kappa", "F",
      "H")
822
823 for(j in 1:length(mlbList)){
824       # training data
825       trainData <- mlTrainingSets[[j]]
826       x <- trainData[, !names(trainData) %in% "response"]
827       y <- trainData$response
828
829       # testing data
830       testData <- mlTestingSets[[j]]
831       xtest <- testData[, !names(testData) %in% "response"]
832       ytest <- testData$response
833
834       # model parameter grids
835       # parameter tuning settings
836       fitControl <- trainControl(method = "cv", number = 10)
837       orfparaGrid <- expand.grid(mtry=c(1, floor(sqrt(ncol(x))), floor(ncol(x)
              /2)))
838       rrfparaGrid <- expand.grid(L=200, K=floor((ncol(x))/c(2, 3, 4)))
839       orrfparaGrid <- expand.grid(K=floor((ncol(x))/c(3)), L=200, mtry=c(1,
              floor(sqrt(ncol(x))), floor(ncol(x)/2)))
840
841       # Forest-RI
842       print(paste("Method: Forest-RI; Data set:", names(mlbList)[j]))
843       Mod <- train(response~., data=trainData, method="rf", trControl=
              fitControl,
844                    tuneGrid=orfparaGrid)
845       preds <- predict(Mod, testData)
846       confMat <- confusionMatrix(preds, ytest)
847       probs <- predict(Mod, xtest, type="prob")[,2]
848
849       # Forest-RI: performance measures
850       results <- summary(HMeasure(ytest, probs), show.all = TRUE)
```

```
851         results$ACC <- confMat$overall[1]
852         results$Kappa <- confMat$overall[2]
853         perfMeasuresRF[j,] <- as.numeric(results[,c(23,11,12,13,24,17,1)])
854
855
856         # oblique rotation random forest: predictions
857         print(paste("Method: rotation random forest; Data set:", names(mlbList)[
                j]))
858         optPara <- findOptimalTuning(x=x, y=y, paraGrid = orrfparaGrid, model="
                rf")
859         optTune <- as.numeric(optPara$optTuneVals)
860         Mod <- RRotF(x=x, y=y, K=optTune[1], L=optTune[2], mtry=optTune[3],
                model="rf")
861         preds <- predict(Mod, xtest)
862         confMat <- confusionMatrix(preds, as.numeric(ytest)-1)
863         probs <- predict(Mod, xtest, type="prob")
864
865         # oblique rotation random forest: performance measures
866         results <- summary(HMeasure(ytest, probs), show.all = TRUE)
867         results$ACC <- confMat$overall[1]
868         results$Kappa <- confMat$overall[2]
869         perfMeasuresORRF[j,] <- as.numeric(results[,c(23,11,12,13,24,17,1)])
870
871         # oblique rotation random forest with logsitic splits: predictions
872         print(paste("Method: oblique rotation random forest with logistic splits
                ; Data set:", names(mlbList)[j]))
873         optPara <- findOptimalTuning(x=x, y=y, paraGrid = orrfparaGrid, model="
                log")
874         optTune <- as.numeric(optPara$optTuneVals)
875         Mod <- RRotF(x=x, y=y, K=optTune[1], L=optTune[2], mtry=optTune[3],
                model="log")
876         preds <- predict(Mod, xtest)
877         confMat <- confusionMatrix(preds, as.numeric(ytest)-1)
878         probs <- predict(Mod, xtest, type="prob")
879
880         # oblique rotation random forest with logsitic splits: performance
                measures
881         results <- summary(HMeasure(ytest, probs), show.all = TRUE)
882         results$ACC <- confMat$overall[1]
883         results$Kappa <- confMat$overall[2]
884         perfMeasuresORRFlog[j,] <- as.numeric(results[,c(23,11,12,13,24,17,1)])
885
886         # rotation forest: predictions
887         print(paste("Method: rotation forest; Data set:", names(mlbList)[j]))
888         Mod <- train(response~., data=trainData, method="rotationForest",
                trControl=fitControl,
889                      tuneGrid=rrfparaGrid)
890         preds <- predict(Mod, testData)
891         confMat <- confusionMatrix(preds, ytest)
892         probs <- predict(Mod, xtest, type="prob")[,2]
893
894         # rotation forest: performance measures
895         results <- summary(HMeasure(ytest, probs), show.all = TRUE)
896         results$ACC <- confMat$overall[1]
897         results$Kappa <- confMat$overall[2]
898         perfMeasuresRotF[j,] <- as.numeric(results[,c(23,11,12,13,24,17,1)])
899
900         # oblique random forest using logistic splits: predictions
901         print(paste("Method: oblique random forest with log splits; Data set:",
                names(mlbList)[j]))
902         Mod <- train(response~., data=trainData, method="ORFlog", trControl=
                fitControl,
903                      tuneGrid=orfparaGrid)
904         preds <- predict(Mod, testData)
905         confMat <- confusionMatrix(preds, ytest)
906         probs <- predict(Mod, xtest, type="prob")[,2]
```

```
907
908         # oblique random forest using logistic splits: performance measures
909         results <- summary(HMeasure(ytest, probs), show.all = TRUE)
910         results$ACC <- confMat$overall[1]
911         results$Kappa <- confMat$overall[2]
912         perfMeasuresORFlog[j,] <- as.numeric(results[,c(23,11,12,13,24,17,1)])
913
914 }
915
916 compareResultsList <- list("rotationForest"=perfMeasuresRotF, "obliqueRFlog"=
        perfMeasuresORFlog,
917                            "obliqueRRF"=perfMeasuresORRF, "obliqueORRFlog"=
                                perfMeasuresORRFlog, "Forest-RI"=perfMeasuresRF
                                )
918 saveRDS(compareResultsList, "benchMarkComparisonsRFs.rda")
919
920 # format benchmark results for thesis
921 B <- compareResultsList
922 finalResultList <- list()
923 algDat <- matrix(0, nrow=7, ncol=5)
924 rownames(algDat) <- colnames(B[[1]])
925 colnames(algDat) <- names(B)
926 for(i in 1:nrow(B[[1]])){
927         for(j in 1:length(B)){
928                 algDat[,j] <- B[[j]][i,]
929         }
930         finalResultList[[i]] <- algDat
931 }
932
933 # name list with data set names
934 names(finalResultList) <- rownames(B[[1]])
935 saveRDS(finalResultList, "finalBenchmarkResults.rda")
936
937 # make latex tables for the results from each data set
938 library(stargazer)
939 for(i in 1:length(finalResultList)){
940         stargazer(finalResultList[[i]], summary = FALSE)
941 }
942
943 # Compute omnibus tests for different performance metrics
944 finalResultList <- readRDS("finalBenchmarkResults.rda")
945 perfMat <- matrix(0, nrow=length(finalResultList), ncol=5)
946 perfMatList <- list()
947 for(i in 1:nrow(finalResultList[[1]])){
948         for(j in 1:length(finalResultList)){
949                 perfMat[j,] <- finalResultList[[j]][i,]
950                 rownames(perfMat) <- names(finalResultList)
951                 colnames(perfMat) <- colnames(finalResultList[[1]])
952         }
953         perfMatList[[i]] <- perfMat
954 }
955 names(perfMatList) <- rownames(finalResultList[[1]])
956
957 # compute Iman-Devenport omnibus p-value per performance metric
958 omniBusTest <- list()
959 for(i in 1:length(perfMatList)){
960         omniBusTest[[i]] <- imanDavenportTest(perfMatList[[i]][,-3])
961 }
962 names(omniBusTest) <- names(perfMatList)
963 # ------------------------------------------------------------------------
```

Bibliography

Allwein, E.L., Schapire, R.E. and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, vol. 1, no. 1, pp. 113–141.

Amaratunga, D., Cabrera, J. and Lee, Y.-S. (2008). Enriched random forests. *Bioinformatics*, vol. 24, no. 18, pp. 2010–2014.

Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, vol. 9, no. 7, pp. 1545–1588.

Bader-El-Den, M. and Gaber, M. (2012). Garf: towards self-optimised random forests. *Proceedings of the International Conference on Neural Information Processing*, pp. 506–515.

Bergmann, B. and Hommel, G. (1988). Improvements of general multiple test procedures for redundant systems of hypotheses. *Multiple Hypothesenprüfung/Multiple Hypotheses Testing*, pp. 100–115.

Bernard, S., Heutte, L. and Adam, S. (2009). On the selection of decision trees in random forests. *Proceedings of the International Joint Conference on Neural Networks*, pp. 302–307.

Blake, C. and Merz, C.J. (1998). UCI Repository of Machine Learning Databases.

Blaser, R. and Fryzlewicz, P. (2015). Random rotation ensembles. *Journal of Machine Learning Research*, vol. 17, no. 4, pp. 1–26.

Boinee, P., De Angelis, A. and Foresti, G.L. (2008). Meta random forests. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 2, no. 6, pp. 138–147.

Bostrom, H. (2007). Estimating class probabilities in random forests. *Proceedings of the Sixth International Conference on Machine Learning and Applications*, pp. 211–216.

Boulesteix, A.-L., Janitza, S., Kruppa, J. and König, I.R. (2012). Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 6, pp. 493–507.

Bradley, A.P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159.

Braga-Neto, U. and Dougherty, E. (2004). Bolstered error estimation. *Pattern Recognition*, vol. 37, no. 6, pp. 1267–1281.

Braida, F., Mello, C.E., Pasinato, M.B. and Zimbrão, G. (2015). Transforming collaborative filtering into supervised learning. *Expert Systems with Applications*, vol. 42, no. 10, pp. 4733–4742.

Breiman, L. (1996*a*). Bagging predictors. *Machine Learning*, vol. 24, no. 2, pp. 123–140.

Breiman, L. (1996*b*). Out-of-bag estimation. *Statistics Department, University of California Berkeley, Berkeley CA 94708, 1996b. 33, 34*, pp. 1–13.

Breiman, L. (2000). Randomizing outputs to increase prediction accuracy. *Machine Learning*, vol. 40, no. 3, pp. 229–242.

Breiman, L. (2001*a*). Random forests. *Machine learning*, vol. 45, no. 1, pp. 5–32.

Breiman, L. (2001*b*). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, vol. 16, no. 3, pp. 199–231.

Breiman, L., Friedman, J., Stone, C.J. and Olshen, R.A. (1984). *Classification and regression trees*. CRC press.

Bühlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, pp. 477–505.

Burman, P. (1989). A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, vol. 76, no. 3, pp. 503–514.

Calvo, B. and Santafé, G. (2015). scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems. *The R Journal*, vol. 8, no. 1, pp. 248–256.

Caruana, R., Karampatziakis, N. and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning*, pp. 96–103.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning*, pp. 161–168.

Cox, T.F. and Cox, M.A. (2000). *Multidimensional scaling*. CRC press.

Cutler, A. and Zhao, G. (2001). Pert-perfect random tree ensembles. *Computing Science and Statistics*, vol. 33, pp. 490–497.

Cutler, D.R., Edwards, T.C., Beard, K.H., Cutler, A., Hess, K.T., Gibson, J. and Lawler, J.J. (2007). Random forests for classification in ecology. *Ecology*, vol. 88, no. 11, pp. 2783–2792.

Das, A., Abdel-Aty, M. and Pande, A. (2009). Using conditional inference forests to identify the factors affecting crash severity on arterial corridors. *Journal of Safety Research*, vol. 40, no. 4, pp. 317–327.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, vol. 7, no. 1, pp. 1–30.

Deng, H. (2013). Guided random forest in the rrf package. *arXiv preprint arXiv:1306.0237*.

Deng, H. (2014). Interpreting tree ensembles with intrees. *arXiv preprint arXiv:1408.5456*.

Deng, H. and Runger, G. (2012). Feature selection via regularized trees. *Proceedings of the International Joint Conference on Neural Networks*, pp. 1–8.

Deng, H. and Runger, G. (2013). Gene selection with guided regularized random forest. *Pattern Recognition*, vol. 46, no. 12, pp. 3483–3489.

Díaz-Uriarte, R. and De Andres, S.A. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, vol. 7, no. 1, p. 1.

Dietterich, T.G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, vol. 10, no. 7, pp. 1895–1923.

Dietterich, T.G. and Kong, E.B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. *Technical Report, Department of Computer Science, Oregon State University*.

Domingos, P. (2000). A unified bias-variance decomposition. *Proceedings of 17th International Conference on Machine Learning*, pp. 231–238.

Dunn, O.J. (1959). Estimation of the medians for dependent variables. *The Annals of Mathematical Statistics*, vol. 30, no. 1, pp. 192–197.

Efron, B. (1983). Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331.

Efron, B. and Efron, B. (1982). *The jackknife, the bootstrap and other resampling plans*, vol. 38. Philadelphia: Society for Industrial and Applied Mathematics.

Efron, B. and Tibshirani, R. (1997). Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560.

Fawagreh, K., Gaber, M.M. and Elyan, E. (2015). On extreme pruning of random forest ensembles for real-time predictive applications. *arXiv preprint arXiv:1503.04996*.

Finner, H. (1993). On a monotonicity problem in step-down multiple test procedures. *Journal of the American Statistical Association*, vol. 88, no. 423, pp. 920–923.

Fisher, R. (1955). Statistical methods and scientific induction. *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 17, no. 1, pp. 69–78.

Freund, Y. and Schapire, R.E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. *Proceedings of the European Conference on Computational Learning Theory*, pp. 23–37.

Friedman, J.H. (1997). On bias, variance, 0/1 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55–77.

Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232.

Friedman, J.H. and Popescu, B.E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, vol. 2, no. 3, pp. 916–954.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701.

Fuchs, T. (2014). Random Forests (slides). Available at: `https://github.com/gdwangh/coursera-JPL-bigdata/tree/master/slide/day8`

Fushiki, T. (2011). Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, vol. 21, no. 2, pp. 137–146.

Gao, D., Zhang, Y.-X. and Zhao, Y.-H. (2009). Random forest algorithm for classification of multiwavelength data. *Research in Astronomy and Astrophysics*, vol. 9, no. 2, p. 220.

García, S., Fernández, A., Luengo, J. and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, vol. 180, no. 10, pp. 2044–2064.

Garcia, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, vol. 9, no. 1, pp. 2677–2694.

Geman, S., Bienenstock, E. and Doursat, R. (1992). Neural networks and the bias/-variance dilemma. *Neural Computation*, vol. 4, no. 1, pp. 1–58.

Genuer, R., Poggi, J.-M. and Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236.

Geurts, P. (2002). *Contributions to decision tree induction: bias/variance tradeoff and time series classification*. Ph.D. thesis, University of Liège Belgium.

Geurts, P., Ernst, D. and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, vol. 63, no. 1, pp. 3–42.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.
Available at: `http://www.deeplearningbook.org`.

Gurney, K. (1997). *An introduction to neural networks*. CRC press.

Hand, D.J. (2009). Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine learning*, vol. 77, no. 1, pp. 103–123.

Hand, D.J. (2010). Evaluating diagnostic tests: the area under the roc curve and the balance of errors. *Statistics in medicine*, vol. 29, no. 14, pp. 1502–1510.

Hand, D.J. and Anagnostopoulos, C. (2013). When is the area under the receiver operating characteristic curve an appropriate measure of classifier performance? *Pattern Recognition Letters*, vol. 34, no. 5, pp. 492–495.

Hand, D.J. and Anagnostopoulos, C. (2014). A better beta for the h-measure of classification performance. *Pattern Recognition Letters*, vol. 40, no. 1, pp. 41–46.

Hastie, T., Tibshirani, R. and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer series in Statistics Springer, Berlin.

Heath, D., Kasif, S. and Salzberg, S. (1993). Induction of oblique decision trees. *In Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1002–1007.

Heskes, T. (1998). Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, vol. 10, no. 6, pp. 1425–1433.

Ho, T.K. (1995). Random decision forests. *Proceedings of the Third International Conference on Document Analysis and Recognition*, vol. 1, pp. 278–282.

Ho, T.K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844.

Hochberg, Y. (1988). A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, vol. 75, no. 4, pp. 800–802.

Hodges, J., Lehmann, E.L. *et al.* (1962). Rank methods for combination of independent experiments in analysis of variance. *The Annals of Mathematical Statistics*, vol. 33, no. 2, pp. 482–497.

Holland, B.S. and Copenhaver, M.D. (1987). An improved sequentially rejective bonferroni test procedure. *Biometrics*, vol. 43, no. 2, pp. 417–423.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70.

Hommel, G. (1988). A stagewise rejective multiple test procedure based on a modified bonferroni test. *Biometrika*, vol. 75, no. 2, pp. 383–386.

Hothorn, T., Hornik, K. and Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, vol. 15, no. 3, pp. 651–674.

Hu, H., Li, J., Wang, H., Daggard, G. and Shi, M. (2006). A maximally diversified multiple decision tree algorithm for microarray data classification. *Proceedings of the Workshop on Intelligent Systems for Bioinformatics*, vol. 73, pp. 35–38.

Iman, R.L. and Davenport, J.M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics-Theory and Methods*, vol. 9, no. 6, pp. 571–595.

Isaksson, A., Wallman, M., Göransson, H. and Gustafsson, M.G. (2008). Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, vol. 29, no. 14, pp. 1960–1965.

James, G. and Hastie, T. (1997). Generalizations of the bias/variance decomposition for prediction error. *Dept. Statistics, Stanford Univ., Stanford, CA, Tech. Rep.*

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R.* Springer.

James, G.M. (2003). Variance and bias for general loss functions. *Machine Learning*, vol. 51, no. 2, pp. 115–135.

Kass, G.V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, vol. 29, no. 2, pp. 119–127.

Kim, H., Kim, H., Moon, H. and Ahn, H. (2011). A weight-adjusted voting algorithm for ensembles of classifiers. *Journal of the Korean Statistical Society*, vol. 40, no. 4, pp. 437–449.

Klassen, M., Cummings, M. and Saldana, G. (2008). Investigation of random forest performance with cancer microarray data. *Computers and Their Applications*, pp. 64–69.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 14, no. 2, pp. 1137–1145.

Kohavi, R. and Wolpert, D.H. (1996). Bias plus variance decomposition for zero-one loss functions. *Proceedings of the 13th International Conference on Machine Learning*, vol. 96, pp. 275–83.

Kong, E.B. and Dietterich, T.G. (1995). Error-correcting output coding corrects bias and variance. *Proceedings of the 12th International Conference on Machine Learning*, pp. 313–321.

Kuhn, R. and De Mori, R. (1995). The application of semantic classification trees to natural language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 449–460.

Kullback, S. and Leibler, R.A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86.

Kwok, S.W. and Carter, C. (1990). Multiple decision trees. *Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 327–338.

Larivière, B. and van Den Poel, D. (2005). Predicting customer retention and profitability by using random forests and regression forests techniques. *Expert Systems with Applications*, vol. 29, no. 2, pp. 472–484.

Latinne, P., Debeir, O. and Decaestecker, C. (2001). Limiting the number of trees in random forests. *International Workshop on Multiple Classifier Systems*, pp. 178–187.

Lee, J.W., Lee, J.B., Park, M. and Song, S.H. (2005). An extensive comparison of recent classification tools applied to microarray data. *Computational Statistics & Data Analysis*, vol. 48, no. 4, pp. 869–885.

Leisch, F. and Dimitriadou, E. (2010). Machine Learning Benchmark Problems: R Package mlbench.
Available at: `https://cran.r-project.org/web/packages/mlbench/mlbench.pdf`.

Lemmond, T.D., Hatch, A.O., Chen, B.Y., Knapp, D., Hiller, L., Mugge, M. and Hanley, W.G. (2008). Discriminant random forests. *Proceedings of the International Conference on Data Mining*, pp. 55–61.

Li, J.D. (2008). A two-step rejection procedure for testing multiple hypotheses. *Journal of Statistical Planning and Inference*, vol. 138, no. 6, pp. 1521–1527.

Lin, Y. and Jeon, Y. (2006). Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, vol. 101, no. 474, pp. 578–590.

Mease, D. and Wyner, A. (2008). Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, vol. 9, no. 2, pp. 131–156.

Menze, B.H., Kelm, B.M., Splitthoff, D.N., Koethe, U. and Hamprecht, F.A. (2011). On oblique random forests. *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 453–469.

Morgan, J.N. and Sonquist, J.A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, vol. 58, no. 302, pp. 415–434.

Nair, A., Kuban, B.D., Tuzcu, E.M., Schoenhagen, P., Nissen, S.E. and Vince, D.G. (2002). Coronary plaque classification with intravascular ultrasound radiofrequency data analysis. *Circulation*, vol. 106, no. 17, pp. 2200–2206.

Nemenyi, P. (1962). Distribution-free multiple comparisons. *Biometrics*, vol. 18, no. 2, p. 263.

Nguyen, T.-T., Huang, J.Z. and Nguyen, T.T. (2015). Unbiased feature selection in learning random forests for high-dimensional data. *The Scientific World Journal*, vol. 2015.

Ostrander, R., Weinfurt, K.P., Yarnold, P.R. and August, G.J. (1998). Diagnosing attention deficit disorders with the behavioral assessment system for children and the child behavior checklist: test and construct validity analyses using optimal discriminant classification trees. *Journal of Consulting and Clinical Psychology*, vol. 66, no. 4, p. 660.

Payne, T.R. and Edwards, P. (1998). Implicit feature selection with the value difference metric. *Proceedings of the 13th European Conference on Artificial Intelligence*, pp. 450–454.

Peng, R. (2011). Reproducible research in computational science. *Science*, vol. 334, no. 6060, pp. 1226–1227.

Peng, R. (2015). Reproducible research. *Johns Hopkins University Data Science Specialization (Coursera).*
Available at: `https://www.coursera.org/learn/reproducible-research`.

Provost, F.J., Fawcett, T. and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. *Proceedings of the International Conference on Machine Learning*, vol. 98, pp. 445–453.

Quade, D. (1979). Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, vol. 74, no. 367, pp. 680–683.

Quinlan, J.R. (1986). Induction of decision trees. *Machine learning*, vol. 1, no. 1, pp. 81–106.

Quinlan, J.R. (1993). *C4.5: Program for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, vol. 5, no. 1, pp. 101–141.

Robnik-Šikonja, M. (2004). Improving random forests. *Proceedings of the European Conference on Machine Learning*, pp. 359–370.

Rodriguez, J.J., Kuncheva, L.I. and Alonso, C.J. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630.

Rom, D.M. (1990). A sequentially rejective test procedure based on a modified bonferroni inequality. *Biometrika*, vol. 77, no. 3, pp. 663–665.

Saffari, A., Leistner, C., Santner, J., Godec, M. and Bischof, H. (2009). On-line random forests. *Workshops of the IEEE 12th International Conference on Computer Vision*, pp. 1393–1400.

Santafe, G., Inza, I. and Lozano, J.A. (2015). Dealing with the evaluation of supervised classification algorithms. *Artificial Intelligence Review*, vol. 44, no. 4, pp. 467–508.

Segal, M.R. (2004). Machine learning benchmarks and random forest regression. *Division of Biostatistics, University of California, San Francisco, CA 94143-0560*.

Seyedhosseini, M. and Tasdizen, T. (2015). Disjunctive normal random forests. *Pattern Recognition*, vol. 48, no. 3, pp. 976–983.

Shaffer, J.P. (1986). Modified sequentially rejective multiple test procedures. *Journal of the American Statistical Association*, vol. 81, no. 395, pp. 826–831.

Siroky, D.S. *et al.* (2009). Navigating random forests and related advances in algorithmic modeling. *Statistics Surveys*, vol. 3, pp. 147–163.

Smith, S.J., Iverson, S.J. and Bowen, W. (1997). Fatty acid signatures and classification trees: new tools for investigating the foraging ecology of seals. *Canadian Journal of Fisheries and Aquatic Sciences*, vol. 54, no. 6, pp. 1377–1386.

Storey, J.D. and Tibshirani, R. (2003). Statistical significance for genomewide studies. *Proceedings of the National Academy of Sciences*, vol. 100, no. 16, pp. 9440–9445.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T. and Zeileis, A. (2008). Conditional variable importance for random forests. *BMC Bioinformatics*, vol. 9, no. 1, p. 1.

Tan, P.J. and Dowe, D.L. (2006). Decision forests with oblique decision trees. *Proceedings of the Mexican International Conference on Artificial Intelligence*, pp. 593–603.

The Radicati Group, I. (2015). Email Statistics Report, 2015-2019. *Email Statistics Report*, vol. 44, no. 0, p. 4.
Available at: `http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf`

Tibshirani, R. (1996). *Bias, variance and prediction error for classification rules*. University of Toronto, Department of Statistics.

Tibshirani, R., Hastie, T., Narasimhan, B. and Chu, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6567–6572.

Tripoliti, E.E., Fotiadis, D.I. and Manis, G. (2013). Modifications of the construction and voting mechanisms of the random forests algorithm. *Data and Knowledge Engineering*, vol. 87, pp. 41–65.

Tsymbal, A., Pechenizkiy, M. and Cunningham, P. (2006). Dynamic integration with random forests. *Proceedings of the European Conference on Machine Learning*, pp. 801–808.

Vayssières, M.P., Plant, R.E. and Allen-Diaz, B.H. (2000). Classification trees: An alternative non-parametric approach for predicting species distributions. *Journal of Vegetation Science*, vol. 11, no. 5, pp. 679–694.

Welbl, J. (2014). Casting random forests as artificial neural networks (and profiting from it). *Proceedings of the German Conference on Pattern Recognition*, pp. 765–771.

Wilson, D.R. and Martinez, T.R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34.

Wolpert, D.H. (1997). On bias plus variance. *Neural Computation*, vol. 9, no. 6, pp. 1211–1243.

Wolpert, D.H. and Macready, W.G. (1999). An efficient method to estimate bagging's generalization error. *Machine Learning*, vol. 35, no. 1, pp. 41–55.

Xu, B., Huang, J.Z., Williams, G., Wang, Q. and Ye, Y. (2012). Classifying very high-dimensional data with random forests built from small subspaces. *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 8, no. 2, pp. 44–63.

Ye, Y., Wu, Q., Huang, J.Z., Ng, M.K. and Li, X. (2013). Stratified sampling for feature subspace selection in random forests for high dimensional data. *Pattern Recognition*, vol. 46, no. 3, pp. 769–787.

Zaklouta, F., Stanciulescu, B. and Hamdoun, O. (2011). Traffic sign classification using K-d trees and random forests. *Proceedings of the International Joint Conference on Neural Networks*, pp. 2151–2155.

Zhang, C.-X. and Zhang, J.-S. (2008). Rotboost: A technique for combining rotation forest and adaboost. *Pattern Recognition Letters*, vol. 29, no. 10, pp. 1524–1536.

Zhang, L. and Suganthan, P.N. (2014). Random forests with ensemble of feature spaces. *Pattern Recognition*, vol. 47, no. 10, pp. 3429–3437.

Ziegler, A. and König, I.R. (2014). Mining data with random forests: current options for real-world applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 1, pp. 55–63.