

Strategies for Combining Tree-Based Learners

Nicholas George Meyer



Thesis presented in partial fulfilment
of the requirements for the degree of
MCom (Mathematical Statistics)
at the University of Stellenbosch

Supervisor: Prof. D. W. Uys

March 2020

PLAGIARISM DECLARATION

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.
2. I agree that plagiarism is a punishable offence because it constitutes theft.
3. I also understand that direct translations are plagiarism.
4. Accordingly, all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
5. I declare that the work contained in this assignment, except otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

NG Meyer	March 2020
Initials and surname	Date

Acknowledgements

Foremost, I would like to thank my parents. If it were not for your endless support and guidance, none of this would have been possible. Thank you so much.

I would also like to extend my sincere gratitude toward the following people and institutions for helping to make this thesis possible:

Prof. D. W. Uys for supervising this thesis.

Prof. S.J. Steel for sharing his invaluable insight into Statistical Learning Theory.

Dr. F. Kamper for providing me with feedback on the optimisation chapter of this thesis.

The Department of Statistics and Actuarial Science, including Schrodgers and Lombard, for providing me with financial assistance through the Schrodgers-Lombard grant.

Dewald Botha for assisting me with the Afrikaans translation of the Abstract.

My employer, Explore, for supporting my studies.

Abstract

In supervised statistical learning, an ensemble is a predictive model that is the conglomeration of several other predictive models. Ensembles are applicable to both classification and regression problems and have demonstrated theoretical and practical appeal. Furthermore, due to the recent advances in computing, the application of ensemble methods has become widespread.

Structurally, ensembles can be characterised according to two distinct aspects. The first is by the method employed to train the individual base learning models that constitute the conglomeration. The second is by the technique used to combine the predictions of the individual base learners for the purpose of obtaining a single prediction for an observation. This thesis considers the second issue. Insofar, the focus is on weighting strategies for combining tree models that are trained in parallel on bootstrap resampled versions of the training sample.

The contribution of this thesis is the development of a regularised weighted model. The purpose is two-fold. First, the technique provides flexibility in controlling the bias-variance trade-off when fitting the model. Second, the proposed strategy mitigates issues that plague similar weighting strategies through the application of ℓ_2 regularisation. The aforesaid includes an ill-condition optimisation problem for finding the weights and overfitting in low signal to noise scenarios.

In this thesis a derivation is provided, which outlines the mathematical details to solve for the weights of the individual models. Crucially, the solution relies on methods from convex optimisation which is discussed. In addition, the technique will be assessed against established ensemble techniques on both simulated and real-world data sets. The results show that the proposal performs well relative to the established averaging techniques such as bagging and random forest.

It is argued that the proposed approach offers a generalisation to the bagging regression ensemble. In this regard, the bagging regressor is a highly regularised weighted ensemble leveraging ℓ_2 regularisation; not merely an equally weighted ensemble. This deduction relies on the imposition of two constraints to the weights, namely: a positivity constraint and a normalisation constraint.

Key words:

Ensemble, Regression, Regularisation, Convex Optimisation, Bagging, Random Forest

Opsomming

In statistiese leer teorie is 'n ensemble 'n voorspellingsmodel wat uit verskeie ander voorspellingsmodelle bestaan. Ensembles kan vir beide regressie en klassifikasie probleme gebruik word, en toon teoretiese en praktiese toepasbaarheid. Ook word ensembles al hoe meer gebruik as gevolg van die onlangse ontwikkeling van rekenaars.

Die tipiese struktuur van ensembles behels twee aspekte. Die eerste is om individuele modelle, wat deel van die ensemble is, te leer. Die tweede is om die voorspellings van individuele modelle te kombineer. Die resultaat is 'n enkele voorspelling vir 'n spesifieke waarneming. Hierdie tesis beskou die tweede aspek. Sover moontlik is die fokus op geweegde strategieë vir gekombineerde regressie bome, waar die bome parallel op skoenlussteekproewe geleer word.

Die bydrae van die tesis is die ontwikkeling van 'n geregulariseerde geweegde model. Die doel is tweeledig. Eerstens bied die tegniek buigsaamheid om die sydigheid-variensie kompromis, wanneer die model geleer word, te beheer. Tweedens vermy die voorgestelde strategie probleme wat by soortgelyke tegnieke, waar ℓ_2 regulering gebruik word, ontstaan. Bogenoemde sluit optimeringsprobleme, waar dit moeilik is om gewigte te bepaal, in; en ook probleme waar oorpasing figureer as gevolg van 'n lae sein-tot-ruis verhouding.

In die tesis word 'n wiskundige raamwerk voorgestel om gewigte van individuele modelle te bepaal. Die oplossing maak gebruik van konvekse optimering, wat ook in die tesis bespreek word. Verder word die voorgestelde tegniek met bestaande ensemble tegnieke, in gesimuleerde en werklike datastelle, vergelyk. Die resultate toon dat die voorgestelde tegniek goed vaar, as dit met *bagging* en *random forest* vergelyk word.

Daar word geargumenteer dat die voorgestelde tegniek 'n veralgemening van die *bagging* regressie ensemble is. In hierdie verband kan *bagging* regressie as 'n hoogs l_2 geregulariseerde ensemble tegniek beskou word; en nie net bloot as 'n geweegde ensemble tegniek nie. Hierdie afleiding is gebaseer op twee beperkings wat op die gewigte geplaas word, naamlik: dat die gewigte positief is, en dat die gewigte genormaliseer word.

Slutelwoorde:

Ensemble, Regressie, Regulering, Konvekse optimering, *Bagging*, *Random Forest*

Table of contents

PLAGIARISM DECLARATION	ii
Acknowledgements	iii
Abstract	iv
Opsomming	v
List of tables	x
List of figures	xi
List of abbreviations and/or acronyms	xii
CHAPTER 1 INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 PROBLEM STATEMENT	1
1.3 LITERATURE REVIEW	2
1.4 CLARIFICATION OF KEY CONCEPTS	3
1.4.1 Supervised Learning	3
1.4.2 Regression	3
1.4.3 Ensemble	3
1.4.4 Convex Optimisation	3
1.5 IMPORTANCE / BENEFITS OF THE STUDY	3
1.6 CHAPTER OUTLINE	4
CHAPTER 2 REVIEW OF STATISTICAL LEARNING THEORY	6
2.1 INTRODUCTION	6
2.2 SUPERVISED LEARNING	6
2.2.1 Squared error loss	8
2.2.2 Mean absolute error	10
2.2.3 Coefficient of determination	10
2.2.4 Training error	10
2.2.5 Generalisation error	11
2.2.6 Model selection	11
2.2.7 Bias-variance trade-off	12
2.2.8 K-fold cross validation	13
2.2.9 Signal to noise	15
2.3 MULTIPLE LINEAR REGRESSION	15
2.3.1 Multiple linear regression construction	16
2.3.2 Non-linear effects	17
2.4 REGULARISATION	17
2.4.1 Ridge regression	17
2.4.2 Ridge regression construction	19

2.4.3	Ridge regression from a Bayesian perspective	20
2.4.4	Benefits of shrinkage	21
2.5	REGRESSION TREES	21
2.5.1	Tree construction	22
2.5.2	Tree size	24
2.5.3	Building a regression tree in Python	27
2.6	THE BOOTSTRAP PROCEDURE	27
2.6.1	Connection to Bayesian inference	29
	CHAPTER 3 COMBINATION STRATEGIES	31
3.1	INTRODUCTION	31
3.2	AVERAGING	33
3.2.1	Simple averaging	33
3.2.2	Weighted averaging	33
3.3	BAGGING	34
3.3.1	Properties of bagging	35
3.3.2	Building a bagged regressor in Python	38
3.4	RANDOM FOREST	39
3.4.1	Details about random forest	40
3.4.1.1	Out of bag samples	40
3.4.1.2	Variable importance	40
3.4.1.3	Proximity plot	40
3.4.1.4	Random forest and overfitting	41
3.4.2	Building a random forest in Python	41
3.5	STACKING	42
3.6	PROPOSED STRATEGY	43
3.6.1	Derivation	43
3.6.2	Motivation	46
3.6.3	Building a weighted ensemble in Python	47
	CHAPTER 4 REVIEW OF OPTIMISATION THEORY	50
4.1	INTRODUCTION	50
4.2	PRIMAL OPTIMISATION PROBLEM	50
4.3	DUAL OPTIMISATION PROBLEM	52
4.4	CERTIFICATES OF SUB OPTIMALITY AND STOPPING CRITERIA	54
4.5	INTERIOR POINT METHODS	55
4.5.1	Logarithmic barrier function and central path	56
4.5.2	Logarithmic Barrier	56
4.5.3	Central Path	57

4.5.4	Dual points from central path	58
4.5.5	Unconstrained minimisation	59
4.5.6	The barrier method	59
4.5.7	Accuracy of centering	60
4.5.8	Choice of μ	60
4.5.9	Choice of $t^{(0)}$	60
CHAPTER 5 DATA DESCRIPTION AND METHODOLOGY		61
5.1	INTRODUCTION	61
5.2	DATA DESCRIPTION	61
5.2.1	The Ozone data set	61
5.2.2	The Boston house-prices data set	62
5.2.3	Friedman 1	64
5.2.4	Friedman 2 and Friedman 3	65
5.2.5	Note on implementation	66
5.3	METHODOLOGY	66
5.3.1	Methodology 1	67
5.3.2	Methodology 2	69
5.3.3	Methodology 3	69
5.4	GRID SEARCH CV	70
5.5	CONQP FUNCTION	71
CHAPTER 6 RESULTS		72
6.1	INTRODUCTION	72
6.1.1	General remarks on the results	72
6.2	SIMULATION RESULTS	73
6.2.1	Friedman 1 results	73
6.2.2	Friedman 2 results	74
6.2.3	Friedman 3 results	75
6.2.4	Remarks	75
6.3	REAL WORLD DATA SET RESULTS	77
6.3.1	The Boston house prices data set	77
6.3.2	The Ozone data set	78
6.3.3	Remarks	78
6.4	BIAS VARIANCE DECOMPOSITION	80
CHAPTER 7 CONCLUSION		82
7.1	FINAL REMARKS	82
7.2	FURTHER RESEARCH	83
REFERENCES		85

APPENDIX A	89	
A.1	DEFINITIONS	89
A.1.1	Line	89
A.1.2	Affine set	89
A.1.3	Affine hull	89
A.1.4	Interior and relative interior	89
A.1.5	Convex set	90
A.1.6	Cone	90
A.1.7	Convex cone	90
A.1.8	Dual cone	90
A.1.9	Nonnegative orthant	90
A.1.10	Self-dual cone	90
A.1.11	Second-order cone	90
A.1.12	Positive semi-definite cone	90
A.1.13	Convex function	91
A.2	COMPLEMENTARY SLACKNESS	91
A.3	KARUSH-KUHN-TUCKER CONDITIONS	92
A.3.1	KKT conditions for non-convex problems	92
A.3.2	KKT conditions for convex problems	93
A.4	UNCONSTRAINED OPTIMISATION PROBLEMS	94
A.4.1	Initial point and sublevel set	94
A.5	DECENT METHOD	95
A.5.1	Line search	96
A.5.2	Newton step	97
A.5.2.1	Minimiser of second-order approximation	97
A.5.2.2	Steepest decent direction in Hessian norm	97
A.5.3	Newton decrement	97
APPENDIX B	99	
B.1	SOURCE CODE	99
B.1.1	Weighted bagging class	99
B.1.2	Experiment class	113
B.1.3	Main class	118
B.1.4	Configuration class	119
B.1.5	Data class	119

List of tables

Table 5.1	Description of the predictors included in the Ozone data set.
Table 5.2	Description of the predictors included in the Boston housing prices data set.
Table 6.1	Arithmetic mean MSE over 10 experiments on the Friedman 1 data set.
Table 6.2	Arithmetic mean MAE over 10 experiments on the Friedman 1 data set.
Table 6.3	Arithmetic mean R^2 over 10 experiments on the Friedman 1 data set.
Table 6.4	Arithmetic mean MSE over 10 experiments on the Friedman 2 data set.
Table 6.5	Arithmetic mean MAE over 10 experiments on the Friedman 2 data set.
Table 6.6	Arithmetic mean R^2 over 10 experiments on the Friedman 2 data set.
Table 6.7	Arithmetic mean MSE over 10 experiments on the Friedman 3 data set.
Table 6.8	Arithmetic mean MAE over 10 experiments on the Friedman 3 data set.
Table 6.9	Arithmetic mean R^2 over 10 experiments on the Friedman 3 data set.
Table 6.10	Arithmetic mean MSE over 10 experiments on the Boston house prices data set.
Table 6.11	Arithmetic mean MAE over 10 experiments on the Boston house prices data set.
Table 6.12	Arithmetic mean R^2 over 10 experiments on the Boston house prices data set.
Table 6.13	Arithmetic mean MSE over 10 experiments on the Ozone data set.
Table 6.14	Arithmetic mean MAE over 10 experiments on the Ozone data set.
Table 6.15	Arithmetic mean R^2 over 10 experiments on the Ozone data set.

List of figures

- Figure 2.1 The bias-variance trade-off as a function of model complexity.
- Figure 2.2 10-fold cross validation.
- Figure 2.3 The ridge coefficients as a function of the shrinkage parameter on synthetic data.
- Figure 2.4 Recursive binary partitioning regression tree of two-dimensional feature space.
- Figure 2.5 Collapsing internal nodes of a large tree.
- Figure 3.1 Three motivating reasons for why combination methods are an appropriate strategy in the context of ensemble learning.
- Figure 3.2 The bagging procedure.
- Figure 3.3 Source code that is used to solve for the optimal least squares weights in the proposed weighted ensemble.
- Figure 4.1 The dashed line shows the function L_{\cdot} . The solid lines show the approximation \hat{L}_{\cdot} for $t = 0.5, 1$ and 2 .
- Figure 5.1 Methodology 2 involves using the OOB observations from each bootstrap sample to obtain the optimal weight for the weighted ensemble.
- Figure 6.1 Bias-variance decomposition and weight profiles for 25 base learners on Friedman 1.

List of abbreviations and/or acronyms

API	Application Programming Interface.
CV	Cross Validation.
EDF	Empirical Density Function.
EPE	Expected Prediction Error.
iid	Independent and Identically distributed.
K-fold CV	K fold Cross Validation.
KKT	Karush–Kuhn–Tucker.
LASSO	Least Absolute Shrinkage and Selection.
MAE	Mean Absolute Error.
MARS	Multivariate Adaptive Regression Splines.
MDS	Multi-Dimensional Scaling.
MLE	Maximum Likelihood Estimator.
MLR	Multiple Linear Regression.
MSE	Mean Squared Error.
OLS	Ordinary Least Squares.
OOB	Out of Bag.
RHS	Right Hand Side.
RSS	Residual Sum of Squares.
SLT	Statistical Learning Theory.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In supervised statistical learning, it is common to fit or learn a single model for the purpose of prediction and/or inference in regression or classification problems. On the other hand, ensemble methods consider fitting multiple learners for the same regression or classification problem. Considerable attention has been given to the study of ensemble methods in the statistical learning literature. The techniques that have manifested from this research have proven to be both theoretically justifiable and empirically beneficial in certain applications.

The models that compose an ensemble are often referred to as base learners. If the base learners in the ensemble are members of the same model class, the resulting ensemble is called a homogeneous ensemble. Otherwise, if the opposite is true, i.e. if the base learners can be categorised as belonging to different model classes, the ensemble is called a heterogeneous ensemble.

The literature defines two broad strategies for constructing an ensemble. In parallel ensemble methods, multiple learners are trained independently of one another. Instead, in sequential ensemble methods, the dependence between base learners in the ensemble is exploited. In either regard, once the base learners have been fit it is necessary to combine their expertise into a single model – the ensemble model. To this end, there are several established strategies for both classification and regression problems. In this thesis some of these methods will be explored from both a theoretical and empirical perspective.

In addition, this thesis proposes and investigates an adaptation to the existing methodology of a weighted ensemble in the context of regression problems. The proposed adaptation involves introducing a regularisation parameter into the optimisation criterion for finding the weights. The solution relies on convex optimisation theory. Therefore, attention will be given to this subject matter.

To motivate the proposal, a mathematical derivation for the technique will be presented. The aim of which is to highlight how the optimal weights can be determined. In addition, an empirical study will be conducted in which the proposed technique will be compared to existing techniques on both simulated and real-world data sets.

1.2 PROBLEM STATEMENT

In this thesis consideration will be given to the problem of combining multiple tree-based learners in a homogenous regression tree ensemble. Specifically, the objective of this thesis is to study the impact of regularisation (as used in the optimisation criterion for finding the weights in a weighted

ensemble) on the generalisation ability of tree-based ensemble methods in the context of regression problems. The study will provide an empirical investigation in which the proposed method will be compared to existing, averaging, tree-base regression ensemble methods, namely: bagging and random forest. It is of interest to consider and answer the question of whether regularisation can facilitate added flexibility in the construction of the final ensemble model.

1.3 LITERATURE REVIEW

In this section we provide a brief overview of developments leading up to the discussion in this study. In review of the literature on ensemble methods it is common to begin with a reference to the work of Dasarathy and Sheela (1979) as this is the first known example of an ensemble model. In this work the partitioning of the feature space, using multiple classifiers, is investigated.

Subsequent to this development, Hansen and Salamon (1990) showed that an ensemble constituting closely configured neural networks could be used to improve classification accuracy.

The seminal work of Schapire (1990) affirmatively proved that weakly learnable problems are equivalent to strongly learnable problems. This property is called boosting, and an interest in this problem emerged as a result of the theoretical question posed by Kearns and Valiant (1989).

Jordan and Jacobs (1994) introduced the concept of mixtures of experts. In this method multiple expert learners are used to partition the input space into homogenous regions.

Wolpert (1992) introduced stacked generalisation (stacking) which uses a model to combine the predictions of other models.

The combination of multiple classifiers was studied by Xu *et al.*, (1992), Ho *et al.*, (1994), Rogova (1994), Lam and Suen (1997) and Woods *et al.*, (1997). Further, dynamic classifier selection is considered in Woods *et al.*, (1997).

Freund and Schapire (1997) demonstrated that a strong classifier, with an arbitrarily low error on a binary classification problem, can be constructed from an ensemble of weak classifiers. In respect of this, the qualifier “weak” meaning to say that the error of any one of the base learners being only slightly better than random guessing. The authors called the resulting method Adaptive Boosting or AdaBoost.

Breiman (1994) introduced the concept of the bagging ensemble, which is a parallel ensemble method that leverages bootstrap resampling proposed by Efron (1979). Bagging improves the diversity among the base learners.

In addition, Breiman (2001) refined the idea of bagging by introducing the random forest ensemble. The improvement in random forest manifests itself in the form of a randomised feature selection procedure.

Beyond that already mentioned, there have been many more developments in the field of ensemble methods. However, in general, the procedures in ensemble methods are differentiated based on two fundamental characteristics. That is, the manner in which the individual base learners are trained and the method used to combine the individual base learners.

1.4 CLARIFICATION OF KEY CONCEPTS

The following terms are used throughout this thesis and are therefore defined for reference.

1.4.1 Supervised Learning

Supervised learning is a statistical modelling technique concerned with fitting or learning a function for associating inputs to outputs. The inputs are often referred to as observations, which are measurements on predictor variables, and the outputs are referred to as the response. To fit a supervised learning model, it is necessary to consider input-output pairs.

1.4.2 Regression

Regression is a type of supervised learning problem where the response is real-valued.

1.4.3 Ensemble

An ensemble is a type of supervised learning model which is constructed by combining the predictions of multiple supervised learning models.

1.4.4 Convex Optimisation

Convex optimisation is a mathematical framework employed to find a minimising solution to an objective function, given a set of equality and/or inequality constraints. In particular, the objective function should be convex - convexity is a mathematical property defining characteristics of the objective function.

1.5 IMPORTANCE / BENEFITS OF THE STUDY

In general, ensemble methods offer the practitioner a powerful modelling technique for real world applications. Often, the problems encountered in these circumstances contain a significant amount of noise and hence the application of a single model may be inadequate. Through the combination of several learners, it is often possible to overcome the problems associated to a single model and in doing so significantly improves the outcome.

This thesis presents a method that gives the practitioner greater flexibility in the application of a particular form of combination strategy called weighting. The added flexibility is due to the introduction of a regularisation parameter into the optimisation criterion for finding the weights. Regularisation facilitates the ability to control the bias-variance trade-off and in doing so can potentially improve the generalisation ability of the ensemble. Furthermore, regularisation is used to

overcome the issue of poor conditioning which is associated to weighting strategies that rely on inverting a covariance matrix. Hence, the adaptation proposed in this thesis ensures that the problem is both feasible and potentially useful to the practitioner. In this thesis the base learners are chosen to be regression trees which are trained in parallel on separate bootstrap resampled versions of the training sample.

Notwithstanding the aforementioned, in reading this thesis the statistician or practitioner will be introduced to different ensemble modelling techniques and to ideas derived from the fields of engineering and computer science – including convex optimisation and open source machine learning API's. This will give the reader an appreciation for both theoretical and practical aspects of combination strategies in ensemble models.

1.6 CHAPTER OUTLINE

In Chapter 2 the discussion focuses on Statistical Learning Theory. In particular, attention will be given to terminology and definitions regarding the different statistical learning procedures and methods that will be used throughout this thesis. The chapter begins through consideration of basic, however fundamental, techniques in supervised learning. Thereafter, the discussion moves on to different statistical learning models. Of importance in this chapter is the bias-variance trade-off and regularisation, as it is these concepts that are critical to understanding statements regarding the proposed weighting strategy.

Chapter 3 presents several combination strategies in the context of regression problems. In addition to this, Chapter 3 introduces the regularised weighting strategy. In Chapter 3 various theoretical aspects of the combination strategies will be discussed. It is stated that combination is a good idea for three main reasons. Proofs will be given providing evidence to support these claims. The derivation for the proposed weighted regularisation strategy will be provided in addition to a motivation.

Convex optimisation theory is discussed in Chapter 4. The chapter defers important foundational concepts to Appendix A. The chapter culminates with a discussion on path following methods. Path following methods are important for solving optimisation problems that contain equality and inequality constraints. From Chapter 3 the reader will be aware that the regularised weighted ensemble can be formulated as a quadratic optimisation problem with equality and inequality constraints. The imposed constraints ensure that the weights sum to one and are positive. Hence, a path following algorithm can be used to solve for the optimal weights.

Chapter 5 discusses the data used for the purpose of experimentation and also provides a detailed account of the experimental methodology. In this thesis five data sets are considered. Three of these are simulated. The procedure used to generate these data sets will be discussed in Chapter 5. Lastly,

the chapter discusses the different parameter settings that were considered for optimising the complexity parameters of the base learners in the ensemble models tested.

Chapter 6 discusses the empirical results. In this chapter a comparison is given between five different modelling procedures on five different data sets. For this purpose, we use the mean squared error (MSE), R^2 score and the mean absolute error (MAE).

Chapter 7 provides a short conclusion and discusses areas for further research.

CHAPTER 2

REVIEW OF STATISTICAL LEARNING THEORY

2.1 INTRODUCTION

In this chapter important concepts from the Statistical Learning Theory (SLT) literature are discussed. SLT formalises the task of learning from data and it is of importance in statistics, data mining and artificial intelligence (Hastie *et al.*, 2009).

In Section 2.2 supervised learning is discussed. Subsequently, results of importance to the development of a good learning model are discussed, including error measures, the bias-variance trade-off, model selection and cross validation. In Section 2.2 to 2.5, attention is given to modelling techniques, including the theoretical aspects of these techniques, such as multiple linear regression (MLR) and regression trees. In Section 2.6 the bootstrap resampling technique is discussed.

Unless otherwise stated, the discussion in the chapter has been refashioned from literature of Hastie *et al.*, (2009) and James *et al.*, (2013).

2.2 SUPERVISED LEARNING

SLT is concerned with the problem of learning from data. The solution to this problem often involves fitting a function that is capable of capturing the expected or regular behaviour in data. Familiar to the field of SLT are the focal areas of supervised and unsupervised learning. This thesis focuses exclusively on the former. To this end there are two dominant problems, namely: regression and classification learning.

In supervised learning a function or model is fit to associate inputs with their corresponding outputs. The purpose is predictive inference, where an outcome is obtained given an input for which there is no output, and/or statistical inference, where the aim is to discern the way in which the output is affected as the input is changed. The input (equivalently called an observation) is comprised of one or more measurements that are referred to as features or predictors. The outcome is referred to as the response.

In supervised learning it is assumed that there exists a true but unknown function defining a systematic association between the predictors and their response. The aim is to approximate this function, and to do so it is necessary to have a set of predictor-response realisations. This set is called a training set and it is defined as

$$\mathcal{T} = \{(\underline{x}_i, y_i), i = 1, \dots, N\}, \quad (2.1)$$

where $N \in \mathbb{N}$. The notation, \underline{x}_i , in (2.1) denotes an observation which is a realisation of a multivariate random vector of variables $\underline{X}^T = (X_1, \dots, X_p)$ for which $X_j, j = 1, \dots, p$ is a random predictor variable

and $p \in \mathbb{N}$. The vector \underline{X} is often called the vector of independent variables. The outcome y_i in (2.1) is a scalar realisation of a random response variable Y . The variable Y is often referred to as the dependent variable. Note that in (2.1) Y is univariate. However, in general the response can be a vector of random variables $\underline{Y}^T = (Y_1, \dots, Y_k)$ with $k \in \mathbb{N}$. It is assumed that the collection of all possible realisations (referred to as the population) are distributed according to the unknown joint probability distribution $P(\underline{X}, Y)$. Hence, the training set in (2.1) is a sample from this distribution.

Note that in (2.1) nothing has been assumed about the characteristics of Y . Thus, the expression in (2.1) holds for both classification and regression problems. In the former, the response can assume one of a finite number of categories, $Y \in \{1, \dots, K\}$. In the latter, the response is numeric, i.e. $Y \in \mathbb{R}$. Notwithstanding this, the predictors can be quantitative and/or qualitative in either regression or classification.

Now, assume that we observe a numeric response, i.e. $Y \in \mathbb{R}$. It is instructive to define the association between the predictors and the response as

$$Y = f(\underline{X}) + \epsilon. \quad (2.2)$$

The expression in (2.2) is a general representation of a non-deterministic association between the predictors and the response. Specifically, f is the fixed, but unknown function representing the systematic relationship between the inputs and the outcomes. The stochastic error term ϵ is often assumed to be distributed as $\epsilon \sim N(0, \sigma_\epsilon^2)$ independent of \underline{X} , that is, normally distributed with mean 0 and variance σ_ϵ^2 . The error term captures random noise in the response due to measurement error and other unobserved influences.

In conjunction with the assumption of normality, if it is assumed that the error terms are independent and identically distributed (iid) and independent of \underline{X} , the function f in (2.2) is

$$f(\underline{x}) = E_{Y|\underline{X}}(Y|\underline{X} = \underline{x}). \quad (2.3)$$

In (2.3) the function that needs to be approximated is the conditional mean of the response. The conditional probability distribution $P(Y|\underline{X})$ depends solely on \underline{X} through the conditional expectation in (2.3).

In regression the training observations are used to approximate the unknown function given in (2.3). The result is a learned model or estimator \hat{f} for which a prediction, given an observation, is $\hat{Y} = \hat{f}(\underline{X})$. In this regard, \hat{f} is an approximation for f . It is crucial to note that the learner does not know anything about the distribution $P(\underline{X}, Y)$. Rather, the only information available to the learner is that which is present in the training data.

The equation given in (2.2) permits the use of both parametric and non-parametric methods for learning. In the case of parametric modelling, the problem of estimating the true function f is

conveniently reduced to the problem of estimating a set of model parameters. In parametric modelling it is perhaps more instructive to denote the true function as $f_{\underline{\theta}}$, where $\underline{\theta}$ explicitly represents the true but unknown parameters of interest. In parametric modelling the number of model parameters does not vary with the size of the training data. However, we are required to make an assumption about the explicit form of the true function. For example, we might be inclined to assume that the true function is linear, whence an appropriate parametric technique can be applied. This greatly simplifies the task of learning, but it incurs the drawback that a rigid assumption may be inappropriate. This could result in a poor approximation of f . To remedy this, a less rigid assumption can be made (which in turn implies that the model is more complex), but this requires estimating more parameters which could lead to overfitting. Overfitting is a term that describes the phenomenon in which a model learns errors or noise present in the training data. This is often the result of an overly complex model and insufficient data. In general, the amount of data required to get a good approximation to the underlying function increases as the number of model parameters increases.

Other than parametric modelling, non-parametric modelling can be considered. In non-parametric modelling no explicit assumption regarding the form of f is made. The result is a model that enjoys a lot of flexibility, but it requires significantly more observations (when compared to parametric methods) to obtain a good approximation for f . In non-parametric modelling the estimator is chosen to be as close to the data as possible while regulating the roughness of the fit.

In the case of either parametric or non-parametric modelling, the final estimator that has been learned should reflect the general patterns in the underlying population – this is referred to as the signal in the data. Ideally the learner should not learn patterns specific to the training data – which is referred to as the noise.

2.2.1 Squared error loss

To quantify the goodness of fit, a measure of closeness must be defined. However, it is of importance to first clarify the reasons for assessing a model. The first reason is model selection. Model selection involves using an estimate of the generalisation error to select the best possible model from among many candidates. The generalisation error is a measure of the ability of a learning method to accurately predict or estimate the response for observations that are not contained in the training set. Refer to Subsection 2.2.5 for a detailed explanation of the generalisation error. The second reason is model assessment which is a quantification of the predictive capability of a model.

The squared error loss is a popular metric used in regression problems to quantify goodness of fit and is defined as

$$\mathcal{L}(Y, f(\underline{X})) = (Y - f(\underline{X}))^2. \quad (2.4)$$

The squared error loss function yields a quadratic increase in error for a linear increase in difference between the true response Y and $f(\underline{X})$. Moreover, it is exclusively non-negative and values closer to zero are better. The squared error loss is also almost always greater than zero because of randomness that is independent of the predictors, or because the estimator does not account for information in the training data. The latter is called underfitting. Underfitting is the consequence of an invalid or overly rigid assumption regarding the functional form of f . It is further noteworthy to consider that the squared error loss is a second moment (about the origin) of the model error. This implies that it accounts for both the variance and bias of an estimator. Thus, for an unbiased estimator, the squared error loss is equal to the variance of the estimator.

The expected (squared) prediction error (EPE) conditional on \underline{X} is

$$EPE = E_{\underline{X}} E_{Y|\underline{X}} \left((Y - f(\underline{X}))^2 \mid \underline{X} \right). \quad (2.5)$$

The minimiser of (2.5) is

$$f(\underline{x}) = E_{Y|\underline{X}}(Y \mid \underline{X} = \underline{x}). \quad (2.6)$$

The solution in (2.5) is the conditional mean of the response given a realised observation. Notice how (2.6) compares with (2.3), however, in (2.3) the squared error loss is not assumed.

Related to the squared error loss is the residual sum of squares (RSS) and the mean squared error (MSE). The RSS measures the sum of the square of the deviations from the output of the true function to the actual response, i.e. $\epsilon_i = y_i - f(\underline{x}_i)$, $i = 1, \dots, N$, and is defined as

$$RSS = \sum_{i=1}^N \epsilon_i^2. \quad (2.7)$$

The deviations ϵ_i , $i = 1, \dots, N$ are called the residuals. The RSS plays a role in the ordinary least squares criterion for linear regression which is discussed in Section 2.3.

The MSE is defined as

$$MSE = \frac{1}{N - d} \sum_{i=1}^N (y_i - \hat{y}_i)^2. \quad (2.8)$$

The MSE is a measure of the square of the error made by a fitted function over the training observations where $\hat{y}_i = \hat{f}(\underline{x}_i)$. The MSE, as given in (2.8), is an unbiased estimate of the variance of the unobserved errors present in the data assuming no model intercept. If the model contains an intercept term, the denominator is $N - d - 1$. The value $d \in \mathbb{N}$ is the degrees of freedom of the model. Simply, the degrees of freedom of a model is a count on the number of parameters of a model that are free to vary. Instead, the biased approximate is scaled by N , the number of observations.

For example, if $\tilde{\theta}$ is an estimator for θ , the biased MSE is $MSE(\tilde{\theta}) = Var(\tilde{\theta}) + [E(\tilde{\theta}) - \theta]^2$. The second term is the squared bias in the estimate of the MSE for the variance of the unobserved errors.

2.2.2 Mean absolute error

As mentioned previously, the MSE is a measure of the square of the error made by a fitted function over the training observations. Thus, large errors contribute significantly more to the overall error than small errors do. This means that the MSE is sensitive to outliers in the sample. Outliers are observations that are possibly erroneous because they differ significantly from the other observations in the sample. The mean absolute error (MAE) is more robust to outliers than the MSE as it quantifies absolute deviations of the fitted function from the training observations instead of squared deviations as is the case when using the MSE. The MAE is defined as

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|. \quad (2.9)$$

2.2.3 Coefficient of determination

The coefficient of determination R^2 is also a measure of goodness of fit. R^2 measures the proportion of the variance in the dependent variable that is explained by the independent variables. Typically, R^2 ranges in value from zero and one, with values closer to one indicating a better fit. The coefficient of determination, R^2 , is calculated as

$$R^2 = 1 - \frac{SS}{TSS}, \quad (2.10)$$

where

$$SS = \sum_{i=1}^N (y_i - \hat{y}_i)^2 \text{ and } TSS = \sum_{i=1}^N (y_i - \bar{y})^2, \quad (2.11)$$

with

$$\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i. \quad (2.12)$$

In (2.10) and (2.11), SS is the sum of squares and TSS is the total sum of squares.

2.2.4 Training error

The training error is a measure of the loss obtained by a fitted model on the training observations. If a model achieves a small training error, then it means that the model is good at predicting the training observations. In Subsection 2.2.5 it is argued that this is not necessarily a desirable outcome. Indeed, a small training does not, in general, imply that the approximation \hat{f} is good. The training error is defined as the average loss over the training samples and is given in (2.13). In (2.13)

$(\underline{x}_i, y_i) \in \mathcal{T}, i = 1, \dots, N$ and $\hat{y}_i = \hat{f}(\underline{x}_i)$. Note in (2.13) that if the loss function used is the squared error loss, the training error is equivalent to the biased MSE criterion.

$$\overline{err} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, \hat{y}_i), \quad (2.13)$$

2.2.5 Generalisation error

The generalisation error or equivalently the test error is defined as the loss obtained by a learned model on observations that were not used as input into the learning procedure (or rather during the learning phase). These observations are called testing observations. Test observations are generated from the same joint distribution $P(\underline{X}, Y)$ as training observations. The test error is calculated as

$$Err_{\mathcal{T}} = E_{\underline{X}, Y} \left(\mathcal{L} \left(Y, \hat{f}(\underline{X}) \right) \middle| \mathcal{T} \right). \quad (2.14)$$

Observe that in (2.14) the training set is fixed, i.e. the expectation is conditioned on the training set. Furthermore, note that \underline{X}, Y are independent of \mathcal{T} . The expected test error, which is defined as

$$Err = E_{\mathcal{T}}(Err_{\mathcal{T}}), \quad (2.15)$$

averages the error measure in (2.14) over different training samples. It turns out that cross validation (CV) estimates the metric in (2.15) because the process used in CV involves averaging an error measure over many different training samples. CV is discussed in Subsection 2.2.8. In addition to this, the measure in (2.15) is more useful for the purpose of model selection than the training error. Model selection is discussed in Subsection 2.2.6.

2.2.6 Model selection

Model selection describes the act of choosing, from among a collection of candidate models, the best model given the training sample. For example, if our aim is to fit a regression tree to the data, model selection might involve choosing the optimal tree depth from among a number of alternatives. Of course, there are other aspects which we can consider. Regression trees are discussed in Section 2.5.

If predictive ability is the primary objective, the training error is not an amenable measure for model selection. In fact, it is more important for a model to be able to correctly predict observations which have not yet occurred, i.e. observations in the test sample. It is therefore imperative that we obtain an adequate estimate for the test error given in (2.14) and select the best model using this estimate.

The test error is also good for guiding the choice of learning method and model class. If the training error is small and the test error is large, it indicates that the model is overfit to the training data. In this scenario it is possible that the model is too complex for the problem at hand. Hence, by using

the test error as a guide we may be inclined to try a simpler class of models. Linear regression methods are an example of a simple model class. The multiple linear regression (MLR) method is discussed in Section 2.3.

The disparity between the training and testing error is a consequence of a model adapting to training data which contains random noise. To avoid this problem, the training sampled should be partitioned into three disjoint sets, namely: a train, a test and a validation set. The validation set serves the purpose of optimising the complexity parameters of a model. Complexity parameters or tuning parameters control the fit of a model to the training data. In our previous example of a regression tree, tree depth is a complexity parameter. The model that achieves the smallest validation error is selected as the final estimator for associating inputs with outputs. The error obtained on the validation set is a more desirable approximation for the generalisation error than the training error. However, because the validation set is being used to optimise the complexity parameters, the validation error is still an optimistic indication of the true generalisation ability of the selected model. Therefore, the test set provides an unbiased estimate of the generalisation error. Note that this means that the test set should not be used until the very end of the model fitting process.

The use of disjoint sets for the purpose of tuning complexity parameters is called CV. In Subsection 2.2.8 a generalisation of this procedure, called K-fold cross validation (K-fold CV) is discussed.

2.2.7 Bias-variance trade-off

The EPE conditional on \underline{x} using the squared error loss is

$$EPE(\underline{x}) = E_{Y,\mathcal{T}} \left((Y - \hat{f}(\underline{x}))^2 \mid \underline{X} = \underline{x} \right). \quad (2.16)$$

Furthermore, Err can be expressed as

$$Err = \sigma_\varepsilon^2 + \left(E_{\mathcal{T}}(\hat{f}(\underline{x})) - f(\underline{x}) \right)^2 + E_{\mathcal{T}} \left(\hat{f}(\underline{x}) - E_{\mathcal{T}}(\hat{f}(\underline{x})) \right)^2. \quad (2.17)$$

The first term on the right-hand side (RHS) of (2.17) denotes the variance of the response around its true mean. No matter how well f is approximated, this term cannot be reduced. The second term is the squared bias, which is the amount by which the average of the prediction differs from the true mean. The third term is the expected squared deviation of the estimator around its mean.

In general, a model with high complexity will have low bias but high variance, while a model that has low complexity will have low variance, but high bias. This concept is depicted in Figure 2.1. Therefore, to minimise the EPE there is a trade-off between bias and variance. It is necessary to adjust model complexity to minimise the combined bias and variance of the model.

In tree-based methods, which are discussed in Section 2.5, the bias-variance trade-off is controlled by selecting the optimum tree depth. As stated previously, parameters which regulate the bias-

variance trade-off (also known as complexity parameters or tuning parameter) cannot be chosen directly from the training sample since it will lead to overfitting and in turn poor generalisation performance. Procedures such as K-fold CV, which is discussed in Subsection 2.2.8, should rather be used to optimise these parameters.

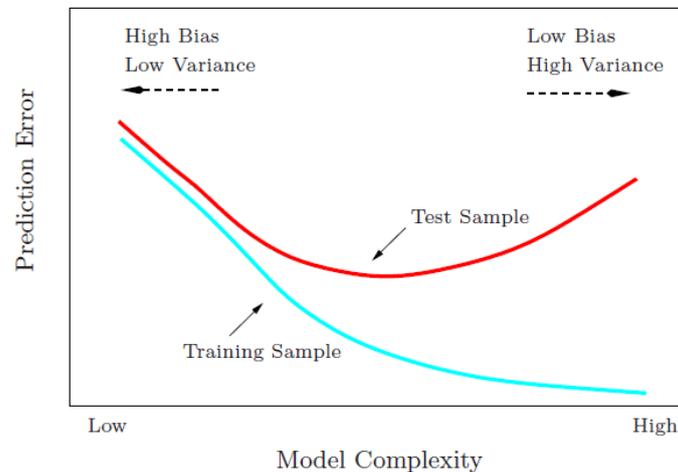


Figure 2.1: The bias-variance trade-off as a function of model complexity.

Source: (Hastie *et al.*, 2001)

2.2.8 K-fold cross validation

For certain parameters, it is desirable to optimise them in a way that ensures a close fit to the training data. For example, the variables and cut points in a regression tree (discussed in Section 2.5) are chosen greedily. However, when trying to select complexity parameters that regulate the bias-variance trade-off, this approach will result in overfitting. For example, the deepest regression tree will invariably be the one for which the training error is smallest. However, the deepest regression tree will not necessarily be optimal on an independent test sample. Indeed, it will probably have high variance and a poor generalisation error.

K-fold CV is a method that can be applied to obtain an unbiased estimate for the generalisation error in scenarios where the amount of available data is limited. In the K-fold CV procedure, the same data can be used for both fitting a model and for model selection. In the CV procedure, the available data is split once, and a disjoint training and validation set is used to fit and tune the model. If the amount of data is limited, this is an inappropriate strategy. In K-fold CV the strategy also involves using disjoint parts of the available data for fitting and testing the model. However, in contrast to CV, this process is repeated multiple times for different parts (or folds) of the available data. In the end, the best model parameters are determined using all folds, and the final model is fitted to all of the available data. It is important to mention that in both procedures the test set should be withheld until the final model is chosen, i.e. the data in the test set should be independent of any data used to fit or tune the model.

To outline the details of the K-fold CV procedure, let α represent a complexity parameter for a model. To apply this technique, it is necessary to define a set of candidate values over which α is to be optimised. Assume that such a set exists. The first step in the K-fold CV procedure is to randomly split the training observations into $K \in \mathbb{N}$ groups or folds such that $K \leq N$. Hence, each group contains approximately N/K observations. Let $\kappa : \{1, \dots, N\} \mapsto \{1, \dots, K\}$ be an indexing function that indicates the partition to which observation i is allocated by randomisation. For a candidate value of α the model is fit K times. Each model is fit using a different collection of $K - 1$ folds with the remaining fold being used to determine the test error. Let $\hat{f}_\alpha^{-k}(\underline{x})$ denote the fitted function, for a candidate value of α , which is computed with the k th fold removed. K-fold CV will yield K measures of the test error and these results will then be averaged to obtain an estimate of the generalisation error as

$$CV = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, \hat{f}^{-\kappa(i)}(\underline{x}_i)). \tag{2.18}$$

The value of α that achieves the smallest estimated test error in (2.18) is chosen to be the optimum value. The K-fold CV procedure is depicted in Figure 2.2 for the case where $K = 10$.

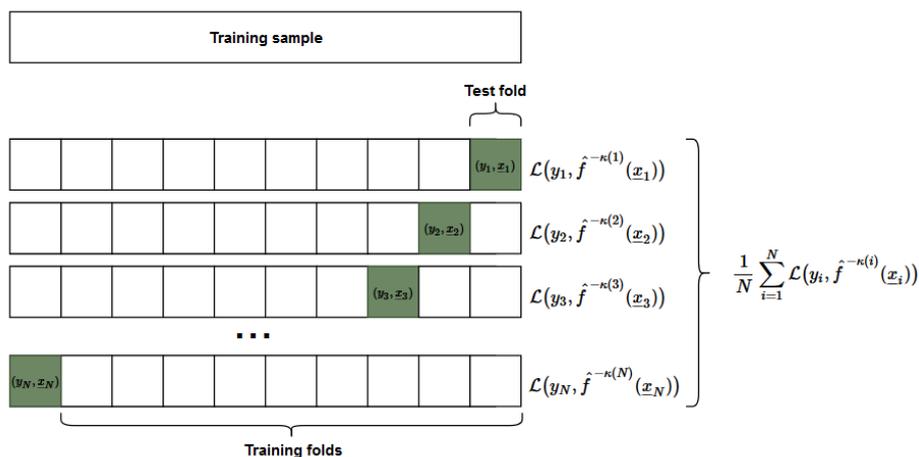


Figure 2.2: 10-fold cross validation.

The value of K is typically chosen to be 5 or 10 because this has been shown to provide a good balance between the bias and variance of the estimators. If K is large, then the majority of the observations in the training set will be used to fit the model; hence the resulting approximation will have low bias. However, since the training sets are nearly equivalent, the resulting estimator will have high variance. The variance being with respect to the model parameters over different training samples. In addition to this, the computational burden associated with large training samples may prove to be a significant hinderance. If $K = 5$ or if $K = 10$ then the training sets are sufficiently dissimilar. Hence, the approximation that results will have high bias but lower variance. The extent

to which the model is biased is dependent on how the performance of the learning procedure varies with the size of the training data set.

2.2.9 Signal to noise

The ratio of signal to the random noise is called the signal-to-noise ratio. In the context of simulated data, it is possible to generate observations with a specific signal-to-noise ratio. To achieve this outcome, it is first necessary to generate N observations. One possible strategy is to generate observations $\{\underline{x}_j, j = 1, \dots, N\}$ by assuming that the features are distributed according to a uniform distribution over the interval 0 to 1, i.e. $X_i \sim Uni(0,1)$, $i = 1, \dots, p$. The resulting observations are iid.

The next step is to generate the responses using a function f according to $y_i = f(\underline{x}_i)$ for $i = 1, \dots, N$. The responses generated in this manner can be used in conjunction with a desired signal-to-noise ratio to determine an appropriate variance for the error term in (2.2). This can be determined as

$$\sigma_\epsilon^2 = \frac{\sqrt{\sum_{i=1}^N (y_i - \bar{y})^2}}{s}, \quad (2.19)$$

where $s \in \mathbb{R}$ is a chosen signal-to-noise ratio and $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$. Using the result in (2.19) we can generate a new set of responses $y'_i, i = 1, \dots, N$ that have the desired signal-to-noise using the relationship

$$y'_i = y_i + \epsilon, \quad (2.20)$$

with $\epsilon \sim N(0, \sigma_\epsilon^2)$.

2.3 MULTIPLE LINEAR REGRESSION

MLR is a simple but highly important and useful technique in supervised regression learning. MLR assumes that the regression function $E_{Y|X}[Y | X = \underline{x}]$ is linear in the inputs (or a linear approximation is adequate). It is important to note that the assumption of linearity is with respect to the model coefficients; not necessarily the predictors.

Given an observation $\underline{x}^T = (x_1, \dots, x_p)$ from a random vector of inputs $\underline{X}^T = (X_1, \dots, X_p)$ and a response y , the MLR model is given as

$$y = \beta_0 + \sum_{j=1}^p x_j \beta_j + \epsilon. \quad (2.21)$$

In (2.21) the coefficients $\beta_j, j = 1, \dots, p$ are unknown and need to be estimated. The coefficient β_0 in (2.21) represents the intercept term (or bias) which corresponds to an input vector $\underline{1}^T = (1, \dots, 1)$ of size $1 \times N$. Further, in (2.21) the error terms are assumed to be iid

normal distributed, i.e. $\epsilon \sim N(0, \sigma_\epsilon^2)$. Note this implies that the residuals are assumed to have 0 mean and constant variance (homoscedastic).

MLR is desirable because it is highly interpretable. In this regard, if we consider a predictor X_j and hold all other predictors X_i for $i \neq j$ constant, the coefficient β_j is interpreted as follows: a one unit increase in X_j causes a β_j unit change in the mean response.

In addition to the representation given in (2.21), the MLR model can be specified in matrix notation as

$$\underline{y} = X\underline{\beta} + \underline{\epsilon}. \quad (2.22)$$

In (2.22) the vectors $\underline{y}^T = (y_1, \dots, y_N)$ and $\underline{\epsilon}^T = (\epsilon_1, \dots, \epsilon_N)$ are both of size $1 \times N$ and the vector $\underline{\beta}^T = (\beta_0, \beta_1, \dots, \beta_p)$ is of size $1 \times (p + 1)$. Further, the matrix $X = [\underline{1} \ \underline{x}_1 \ \dots \ \underline{x}_p]$ is of size $N \times (p + 1)$. To retrieve a particular observation from the matrix X we use the notation x_{ij} , where $i = 1, \dots, N$ and $j = 1, \dots, p$.

2.3.1 Multiple linear regression construction

To estimate the model coefficients in (2.21) the ordinary least squares (OLS) criterion is commonly applied. In OLS the coefficients are found to minimise the RSS over all observations. Hence, the criterion that we seek to minimise is

$$\underline{\hat{\beta}}^{OLS} = \arg \min_{\underline{\beta}} \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2. \quad (2.23)$$

Note that the criterion in (2.23) uses the squared error loss as discussed in Subsection 2.2.1. The optimal coefficients $\underline{\hat{\beta}}^{OLS}$ are called the ordinary least squares coefficients. The OLS coefficients can be obtained by taking the derivative of the objective in (2.23) with respect to $\underline{\beta}$ which gives

$$\underline{\hat{\beta}}^{OLS} = (X^T X)^{-1} X^T \underline{y}, \quad (2.24)$$

assuming that $X^T X$ is non-singular. The fitted model is then

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^p x_j \hat{\beta}_j. \quad (2.25)$$

Due to the Gauss-Markov theorem it is known that the solution given in (2.24) has the smallest variance among all linear unbiased estimators. If the predictors are real valued, the model in (2.25) gives a p dimensional hyperplane in the $(p + 1)$ dimensional input-output space. The height at any

point in this space is given by $\underline{\hat{\beta}}^T \underline{x}$.

2.3.2 Non-linear effects

The simple representation of the MLR model in (2.23) accounts for only additive relationships among the input variables. The underlying assumption is that the influence of each predictor on the response is independent of the influence of all other predictors in the model. It is possible to make this model more flexible by incorporating transformations of the predictors into the formulation. Examples of transformations include terms such as x_j^2 and interaction terms such as $x_j x_i$ for $i \neq j$. The result is a model that can be represented as

$$y = \beta_0 + \sum_{m=1}^M h_m(\underline{x}) \beta_j + \epsilon. \quad (2.26)$$

In (2.26) the functions $h_m(\underline{x})$, $m = 1, \dots, M$ define some transformations of the predictor. The representation in (2.26) is not linear in the predictors but it is still linear with respect to the coefficients $\beta_j, j = 1, \dots, M$. Thus, the response surface is non-linear and of fewer than p dimensions in the $(p + 1)$ dimensional input-output space.

2.4 REGULARISATION

This section considers a simple adaptation to MLR through the introduction of a penalisation term. The resulting technique is called ridge regression. Recall that in Subsection 2.2.7, it is stated that the EPE can be expanded into the sum of a squared bias term, a variance term and a noise term. By controlling the size of the penalty term, it is possible to reduce the variance component of the EPE. However, this is not in isolation. Indeed, it comes at the cost of a small increase in bias.

2.4.1 Ridge regression

Ridge regression (Hoerl and Kennard, 1970) is a supervised linear modelling technique that is an adaptation to MLR through the introduction of a regularisation parameter. In general, regularisation facilitates controlling the bias-variance trade-off, and it is useful for dealing with problems that are ill-conditioned, and it helps to prevent overfitting. To achieve this outcome an additional parameter (called the shrinkage parameter) and a measure of model complexity (called the penalty term) is introduced into the loss criterion of the MLR model.

The shrinkage parameter regulates model complexity by constraining the size of the MLR coefficients. The constraint causes the size of the MLR coefficients to shrink toward 0 and also toward each other. The results of ridge regression are not equivalent under scaling, and so usually the inputs are first standardised before solving for the coefficients.

In general, a regularisation problem can be formulated as in (2.27). In (2.27) \mathcal{L} is an appropriate loss function (such as the squared error loss) and $\xi \geq 0$ is the shrinkage parameter which controls the extent to which the model parameters are constrained. Note that the shrinkage parameter is a complexity parameter. The function J is the penalty term and it measures the complexity of f .

$$\hat{f} = \arg \min_f \left(\sum_{i=1}^N \mathcal{L}(y_i, \hat{y}_i) + \xi J(f) \right). \quad (2.27)$$

In ridge regression model complexity is measured using the ℓ_2 penalty term. In ℓ_2 regularisation the sum of the square of the model parameters is added to the loss criterion, i.e. $J(f) = \sum_{j=1}^p \beta_j^2$. In least absolute shrinkage and selection (LASSO) (Tibshirani, 1996), ℓ_1 regularisation is applied. In ℓ_1 regularisation, $J(f) = \sum_{j=1}^p |\beta_j|$. Readers interested in learning more about the LASSO can consult Hastie *et al.*, (2009).

To find the ridge regression coefficients, it is necessary to minimise the penalised RSS which is specified as

$$\hat{\underline{\beta}}^{ridge} = \arg \min_{\underline{\beta}} \left(\sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \xi \sum_{j=1}^p \beta_j^2 \right). \quad (2.28)$$

Alternatively, the size of the penalisation constraint can be made explicit as

$$\hat{\underline{\beta}}^{ridge} = \arg \min_{\underline{\beta}} \left(\sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 \right) \quad (2.29)$$

subject to $\sum_{j=1}^p \beta_j^2 \leq t$.

The optimal coefficients $\hat{\underline{\beta}}^{ridge}$ in (2.29) are equivalent to those obtained in (2.28), however, in (2.29), the size of the ℓ_2 penalty term is made explicit by the constraint $\sum_{j=1}^p \beta_j^2 \leq t$.

The term $\xi \sum_{j=1}^p \beta_j^2$ in (2.28) counters the potential instability of the OLS criterion in (2.23). The problem of instability in the OLS estimator arises as a result of correlation between predictor variables (multicollinearity). For example, instability is caused if a large positive coefficient for one variable can be offset by a similarly large negative coefficient for another variable. This would be the case when the two variables are highly correlated.

2.4.2 Ridge regression construction

The ridge estimator counteracts the potential instability of the OLS estimator through the addition of a constant $\xi \in \mathbb{R}$, to the diagonal entries of the Gram matrix, $X^T X$ before taking the inverse.

Consider (2.28) in matrix notation as

$$\phi(\underline{\beta}) = (\underline{y} - X\underline{\beta})^T (\underline{y} - X\underline{\beta}) + \xi \underline{\beta}^T \underline{\beta}. \quad (2.30)$$

By differentiating (2.30) with respect to $\underline{\beta}$, it follows that

$$\frac{\partial \phi(\underline{\beta})}{\partial \underline{\beta}} = -2\underline{Y}^T X + 2(X^T X) \underline{\beta} + 2\xi \underline{\beta}. \quad (2.31)$$

If (2.31) is set equal to $\underline{0}$, $\hat{\underline{\beta}}^{ridge}$ can be obtained as

$$\hat{\underline{\beta}}^{ridge} = (X^T X + \xi I)^{-1} X^T \underline{y} = (X^T X + \xi I)^{-1} X^T X \hat{\underline{\beta}}, \quad (2.32)$$

where I is a $p \times p$ identity matrix. When $\xi > 0$, $\hat{\underline{\beta}}^{ridge}$ is a biased estimator of $\underline{\beta}$. Further, in the special case where $X^T X = I$ (the orthonormal design case), (2.32) reduces to $\hat{\underline{\beta}}^{ridge} = (1 + \xi)^{-1} \hat{\underline{\beta}}^{ols}$. If $\xi = 0$, (2.32) is equivalent to the OLS estimator.

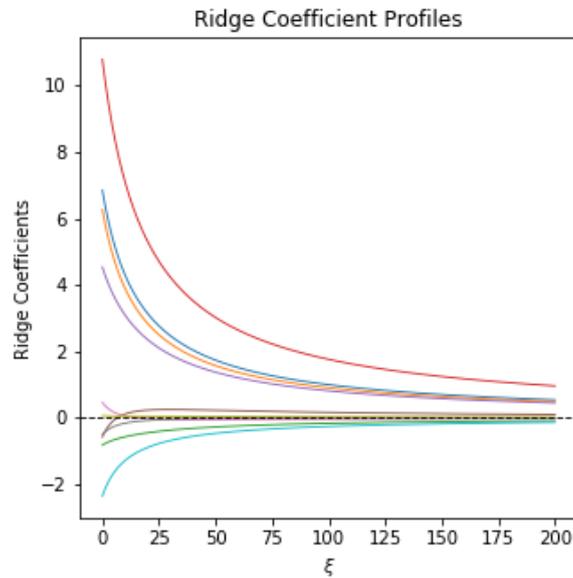


Figure 2.3: The ridge coefficients as a function of the shrinkage parameter on synthetic data.

In conclusion, ridge regression is a technique that can be applied to reduce the variance of the OLS estimator when the predictors are highly correlated. It achieves this at the cost of a small increase in bias. Therefore, ridge regression is appropriate when the OLS estimator has high variance. A further benefit is that ridge regression can be applied when the number of predictors greatly exceeds

the number of observations, i.e. $p > N$. This is not possible in OLS since the Gram matrix $X^T X$ is non-singular – and hence the OLS estimator is undetermined.

Unfortunately, ridge regression is not able to shrink coefficients to zero and as a result, it cannot perform variable selection and hence does not produce a parsimonious representation of the model.

Figure 2.3 demonstrates the manner in which the shrinkage parameter regulates the size of the ridge coefficients. As the size of ξ increases, the coefficients move toward 0 and toward each other. However, as is seen in the Figure 2.3 the coefficients are not set to 0.

2.4.3 Ridge regression from a Bayesian perspective

It turns out that the ridge estimator is a Bayesian estimator when $\underline{\beta}$ is given a suitable multivariate Gaussian prior. Suppose that $\underline{Y} = X\underline{\beta} + \underline{\epsilon}$ where $\underline{\epsilon} \sim N(\underline{0}, \sigma^2 I)$ and the variance σ^2 is known and I is a $N \times N$ identity matrix. This assumption implies that $\underline{Y} \sim N(X\underline{\beta}, \sigma^2 I)$. The likelihood function is proportional to

$$p(\underline{y}|\underline{\beta}) \propto \exp\left(-\frac{1}{2\sigma^2}(\underline{\beta} - \hat{\underline{\beta}})^T X^T X (\underline{\beta} - \hat{\underline{\beta}})\right). \quad (2.33)$$

The expression in (2.33) is in the form of the density function of a p dimensional multivariate normal distribution, i.e. $(\underline{y}|\underline{\beta}) \sim N(\hat{\underline{\beta}}, \sigma^2(X^T X)^{-1})$. Now, assume that each component of $\underline{\beta}$ is independently distributed normal with mean 0 and known variance $\sigma^2_{\underline{\beta}}$, i.e. $\underline{\beta} \sim N(\underline{0}, \sigma^2_{\underline{\beta}} I)$. The prior distribution on the parameters is given as

$$p(\underline{\beta}) \propto \exp\left(-\frac{\underline{\beta}^T \underline{\beta}}{2\sigma^2_{\underline{\beta}}}\right), \quad (2.34)$$

and posterior distribution of $\underline{\beta}$ is

$$p(\underline{\beta}|\underline{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\left((\underline{\beta} - \hat{\underline{\beta}})^T X^T X (\underline{\beta} - \hat{\underline{\beta}}) + k\underline{\beta}^T \underline{\beta}\right)\right), \quad (2.35)$$

where $k = \frac{\sigma^2}{\sigma^2_{\underline{\beta}}}$.

We can write $\underline{\beta} - \hat{\underline{\beta}} = (\underline{\beta} - \hat{\underline{\beta}}^{ridge}) + (\hat{\underline{\beta}}^{ridge} - \hat{\underline{\beta}})$ and $\underline{\beta} = (\underline{\beta} - \hat{\underline{\beta}}^{ridge}) + \hat{\underline{\beta}}^{ridge}$. Thus, the posterior density of $\underline{\beta}$ is given by

$$p(\underline{\beta}|\underline{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\left((\underline{\beta} - \hat{\underline{\beta}}^{ridge})^T (X^T X + kI) (\underline{\beta} - \hat{\underline{\beta}}^{ridge})\right)\right). \quad (2.36)$$

From (2.36) it is clear that the posterior distribution of $\underline{\beta}$ is a multivariate normal distribution with mean vector (equivalently the posterior mode) $\hat{\underline{\beta}}^{ridge}$ and covariance matrix $\sigma^2(X^T X + kI)^{-1}$.

Recall that in the Subsection 2.4.2, it was stated that in the case where the shrinkage parameter is

set to 0 the OLS estimator is retrieved. In (2.36) if $\sigma^2_{\underline{\beta}}$ is very large, the prior distribution becomes un-informative, and hence the ridge estimator approaches the OLS estimator.

2.4.4 Benefits of shrinkage

There are two motivating factors for using an alternative fitting technique to OLS. The first is to improve prediction accuracy and the second is to improve model interpretation.

If we assume that the true relationship between the response and the predictors is approximately linear, the OLS estimator will have low bias. If the number of observations is much larger than the number of predictors, the OLS estimator will also have low variance. However, in circumstances where this is not the case, the OLS estimator will suffer from high variability or may even be undefined (such as in the case for $p > N$). By using shrinkage techniques (such as ridge regression or the LASSO) it may be possible to improve prediction accuracy through a large reduction in variance at the cost of only a small increase in bias.

In addition, often some of the many variables used in a MLR model are not associated with the response. Hence, it may be beneficial to determine a smaller subset of the predictors that exhibit the most important effects. This will result in a parsimonious model (less complex and easier to interpret).

2.5 REGRESSION TREES

In addition to Hastie *et al.*, (2009), James *et al.*, (2013) and Murphy (2012) were used as references when writing this section. Regression trees are one of a broad class of non-linear techniques called adaptive basis function models. These models are given as

$$f(\underline{x}) = w_0 + \sum_{j=1}^M w_j \phi_j(\underline{x}). \quad (2.37)$$

In (2.37) the function ϕ_j is called a basis function and w_j is an associated weight. The basis functions are typically parametric, in which case $\phi_j(\underline{x}) = \varphi(\underline{x}, \underline{\theta}_j)$ where $\underline{\theta}_j$ denotes the parameters.

The methodology employed in constructing a regression tree involves partitioning the range of the input space into rectangular and disjoint regions. Simple models are fit within each region. The regions are constructed using a sequence of simple decision rules that are inferred from the relationship between the features and the response. These decision rules can be represented as a decision tree – which is a hierarchical data structure of nodes and branches.

A node is a data structure to which several sample observations are assigned. The root node represents the entire sample of input observations. A decision involves dividing a group of observations (a node) into two or more groups of observations (two or more nodes). Nodes that are

branched from other nodes in this manner are referred to as child nodes. The node from which the child nodes spawn is called a parent node. A terminal node does not spawn child nodes. A node that is neither the root node nor a terminal node is called an internal node. A branch represents the decision rule that segments the observations at a node. A subtree is any set of child nodes for which the root node is an internal node.

Consider (2.37) in the context of a regression tree. Assume for the sake of argument that the regression tree is piecewise constant – which means that a constant basis function is applied in each region of the input space. In this case the weights in (2.37) will be the mean response in a region of the input space. In addition, the estimated parameters of the regression tree encode the split variables (nodes) and threshold values (decision rules) which define these regions.

Tree models are applicable for both classification and regression problems and can accept as input both categorical and continuous variables. In this thesis, regression trees are used as the base learners for the ensemble techniques considered.

2.5.1 Tree construction

In a regression tree the response is assumed to be continuous. In this discussion, define $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ for $i = 1, \dots, N$ to be a set of observations and the corresponding responses, respectively. To construct a regression tree, it is first necessary to decide upon a set of split variables and a corresponding split point for each variable. In addition, it is necessary to decide upon a tree topology. At present we restrict ourselves to a greedy top down approach that uses recursive binary partitions. A greedy strategy is necessary because finding an optimal partition is an NP-complete problem (Hyafil and Rivest, 1976). It is noteworthy to mention that this strategy is used by popular fitting methods such as classification and regression trees (CART) (Breiman *et al.*, 1984), C4.5 (Quinlan, 1993) and ID3 (Quinlan, 1986).

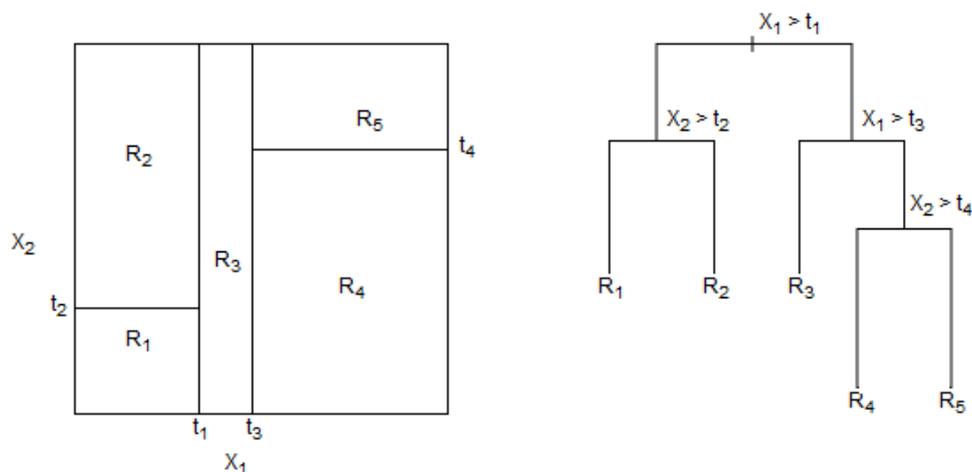


Figure 2.4: Recursive binary partitioning regression tree of two-dimensional feature space.

In Figure 2.4, the diagram on the left depicts two random variables X_1 and X_2 which span a two-dimensional space. The split points for segmenting the feature space are given by t_1, t_2, t_3 and t_4 . The decision tree on the right of Figure 2.4 demonstrates the rules that are used to split the feature space into regions R_1, R_2, R_3 and R_4 .

At the first step of the greedy approach the total feature space (root node) is split into two child nodes

$$\begin{aligned} R_1(j, s) &= \{\underline{x} \mid x_j \leq s\}, \\ R_2(j, s) &= \{\underline{x} \mid x_j > s\}. \end{aligned} \quad (2.38)$$

The equation in (2.38) gives an axis parallel split where j indexes the split feature and s the split point which encode the region. The response is modelled separately in each of these regions; typically using simple models, for example, a constant (in which case all observations falling into a region are assigned the same value as a response) or a linear regression model. The former leads to a regression tree that is piecewise constant. The implication of a greedy approach is that variables which are chosen early on are deemed to be the most explanatory for the purpose of predicting the response. Subsequent to the first partition, one or both regions R_1 and R_2 are further split using the same procedure. This process is followed until a stopping criterion is reached. The stopping point might constitute a specified minimum number of training observations in each terminal region or other heuristics that include: a threshold on the permissible reduction in cost due to a split, a maximum tree depth or a restriction on the permissible homogeneity of the distributions of the responses in each region. Regardless of the criteria, in the end there will be M terminal regions denoted R_1, R_2, \dots, R_M for which there is an associated prediction.

The choice of split variables and a split point are made so as to achieve a locally optimal maximum likelihood estimate (MLE) at each level of the tree. Using the squared error loss and assuming a piecewise constant model, the appropriate criterion for each binary split of a node is

$$\arg \min_{j,s} \left(\arg \min_{c_1} \sum_{\underline{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \arg \min_{c_2} \sum_{\underline{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right). \quad (2.39)$$

The specification in (2.39) attempts to segment the feature space so that the training samples in each region are as homogeneous as possible. Consequentially, the variance in each region will be minimised. Moreover, the criterion is greedy because the decision on how to split the region is based on a locally optimal choice given the current structure of the tree. This is in contrast to a strategy that tries to be globally optimal by making a choice that achieves the best possible future outcome.

The minimisation problem in (2.39) can be solved to obtain

$$\hat{y}_m = \arg \min_{c_m} \left(\sum_{\underline{x}_i \in R_m(j,s)} (y_i - c_m)^2 \right) = \frac{1}{N_m} \sum_{\underline{x}_i \in R_m(j,s)} y_i, \quad (2.40)$$

for $m = 1, \dots, M$ and where N_m denotes the number of training observations in region m . Finally, our estimated model is

$$\hat{f}(\underline{x}) = \sum_{m=1}^M \hat{y}_m I(\underline{x} \in R_m), \quad (2.41)$$

where the function I is the indicator function which is defined as

$$I(\underline{x} \in R_m) = \begin{cases} 1 & \underline{x} \in R_m \\ 0 & \text{otherwise} \end{cases}. \quad (2.42)$$

Therefore, a piecewise linear regression tree model constructed using a binary partitioning scheme assigns to each observation that is an element of a terminal region a single real valued response. It is apparent from (2.41) that under the squared error loss the prediction that obtains the best fit in a terminal node is the mean value of the training responses that lie within that terminal region.

To make a prediction for a test observation, we simply assign as a response the constant value in the terminal node to which the observation belongs. The terminal node would be determined by negotiating the various split variables and split points that make up the construction of the decision tree from the root node to a terminal node.

The reduction in MSE due to the first split is measured as the difference between the MSE of the unpartitioned input space and the sum of the MSE for each of the partitioned regions. The reduction is given as

$$\sum_{i=1}^N (y_i - y_0)^2 - \left(\sum_{\underline{x}_i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \sum_{\underline{x}_i \in R_2(j,s)} (y_i - \hat{y}_2)^2 \right), \quad (2.43)$$

where $y_0 = \frac{1}{N} \sum_{i=1}^N y_i$. Furthermore, the training error at any stage of the model fitting process is defined as

$$RSS(T) = \sum_{m=1}^{|T|} \sum_{\underline{x}_i \in R_m} (y_i - \hat{y}_m)^2. \quad (2.44)$$

In (2.44), the term $|T|$ defines the number of terminal regions for the tree at a particular stage of the fitting process.

2.5.2 Tree size

It is important to determine the optimal tree size, or equivalently the number of partitions of the input space, as it plays a role in overfitting or underfitting. The optimal tree size can be chosen in an adaptive manner from the training data using, for example, K-fold CV. It is directly related to model complexity in the sense that if we increase the size of the tree, we increase the model complexity and vice versa. Intuitively, an increase in complexity leads to an increase in variance since a deep

tree is closely fit to the training data and any noise present therein. Higher model complexity implies a greater number of regions which in turn implies fewer training observations in each region. Since a prediction for a test case is the conditional mean of the training responses, the predictive process is heavily influenced by the noise in the training data. Alternatively, to reduce tree flexibility we can decrease the number of splits. However, this will cause an increase in bias due to fewer regions being used for predicting observations. The aforementioned, demonstrates that tree size regulates the bias-variance trade-off.

Ideally, we would like to consider every possible tree and then choose the one that achieves the smallest error on test cases. This strategy fails for large trees due to computational inefficiency. Another approach is to proceed with growing a tree while the decrease in the error measure (such as the MSE), due to a new split, exceeds a certain threshold. This too fails because it is possible that certain poor splits early in the fitting process may actually result in better splits later in the fitting process.

A preferred strategy is to apply cost-complexity pruning. In this strategy we grow a large tree, say T_0 . Subsequently, we calculate a sequence of subtrees $T_M \subset \dots \subset T_1 \subset T_0$ with the smallest tree T_M containing only one split. Let $|T|$ denote the number of terminal nodes in a tree. Further, define

$$Q_m(T) = \frac{1}{N_m} \sum_{\mathbf{x}_j \in R_m} (y_i - \hat{y}_m)^2, \quad (2.45)$$

as a measure of error for each observation in the training data for a node. The cost complexity criterion is

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|. \quad (2.46)$$

The aim is to find a subtree $T_\alpha \subseteq T_0$ that minimises $C_\alpha(T)$ for α in (2.46). The parameter α in (2.46) is a regularisation parameter that controls tree complexity. To this end, we collapse internal nodes that lead to the smallest per node increase in $\sum_{m=1}^{|T|} N_m Q_m(T)$ or equivalently, we collapse a subtree that minimises the decrease of the cost complexity function given as

$$C_{\alpha_j}(T_0 - T_j) - C_{\alpha_j}(T_0) = \frac{N_j}{N} \sum_{\mathbf{x}_j \in R_j} (y_i - \hat{y}_j)^2 - \sum_{m=1}^{|T_j|} N_m Q_m(T_j) + \alpha_j(1 - |T_j|), \quad (2.47)$$

for $j = 1, \dots, M$ at each iteration. In (2.47) $T_0 - T_j$ is the tree obtained when replacing the subtree T_j with a root node. The idea is depicted in Figure 2.5.

If

$$\alpha_j = \frac{\frac{N_j}{N} \sum_{x_i \in R_j} (y_i - \hat{y}_j)^2 - \sum_{m=1}^{|T_j|} N_m Q_m(T_j)}{|T_j| - 1}, \quad (2.48)$$

(2.47) equals 0. Hence, the subtree for which node j minimises (2.47) is collapsed. This process yields two sequences, T_1, \dots, T_M and $\alpha_1, \dots, \alpha_M$. Subsequently, a search is performed through this finite sequence of trees for the one that minimises the criterion $C_\alpha(T)$ in (2.46).

The sequence of regularisation parameters $\alpha_1, \dots, \alpha_M$ previously obtained in the pruning process is not used to select the final tree since it will lead to overfitting. Instead, a different set of parameters $\zeta_i, i = 1, \dots, M$ is adaptively chosen using a CV procedure. The final tree is $T_{\hat{\zeta}}$. Note that if $\alpha = \infty$ the null tree is selected and if $\alpha = 0$ the full tree is selected.

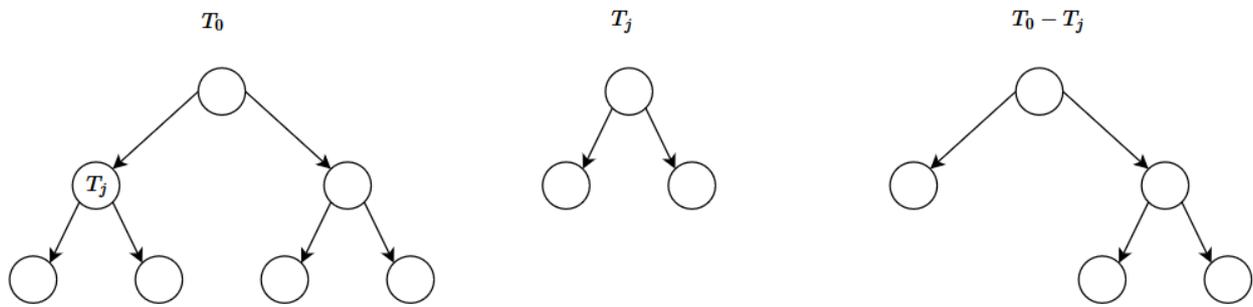


Figure 2.5: Collapsing internal nodes of a large tree unpruned tree.

A similar procedure to cost complexity pruning is weakest link pruning. Weakest link pruning entails forming a sequence of nested trees (until the null tree is obtained) from a large tree T_0 by substituting a subtree with a terminal node that minimises

$$\frac{RSS(T_0 - T_j) - RSS(T_0)}{|T_0| - |T_0 - T_j|}. \quad (2.49)$$

A tree $T_0 - T_j$ for $i = 1, \dots, M$ is selected from the resulting sequence using a CV procedure. The procedure for building a regression tree is given in Algorithm 2.1.

Algorithm 2.1: Tree algorithm (regression)

-
1. **repeat:** Constructing a tree using recursive binary splitting:
 2. **stop:** Heuristic stopping criterion. The resulting tree will be large and overfit to data
 3. Prune the tree using cost complexity pruning. The result is a nested sequence of subtrees
 4. Use K-fold CV to determine the optimal α for each tree in the nested sequence
 5. **end:** Select a nested subtree minimising the cost complexity using α from step 4.
-

2.5.3 Building a regression tree in Python

The scikit-learn Application Programming Interface (API) (Pedregosa *et al.*, 2011) provides a function for building a regression tree in the Python programming language. The interested reader can visit <https://scikit-learn.org/stable/index.html> for more details about this API and its functionality. The programming library was created by Cournapeau (2007) and was developed as part of a Google Summer of Code project. The first public version of the library was made available in 2010. Since its inception it has been actively developed by members from both academia and industry. To date there have been 1335 contributors who have collectively made 24155 submissions to the public repository.

Here is a description of some of the parameters (Pedregosa *et al.*, 2011) that this API provides for the `DecisionTreeRegressor` function (which is the function for building a regression tree in Python):

criterion: (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to the variance reduction as feature selection criterion and minimizes the ℓ_2 loss using the mean of each terminal node, "friedman_mse", which uses mean squared error with Friedman's improvement score for potential splits, and "mae" for the mean absolute error, which minimizes the ℓ_1 loss using the median of each terminal node.

splitter: (default="best")

The strategy used to choose the split at each node.

max_depth: (default=None)

The maximum depth of the tree.

2.6 THE BOOTSTRAP PROCEDURE

Unless otherwise stated, the material in this section has been adapted from Efron and Tibshirani (1993). In statistical inference, the objective is to infer properties of a population distribution F using an iid sample from that distribution. The iid sample is also referred to as a random sample where each observation is sampled from the population with equal probability. It is often of interest to estimate a parameter of the distribution F . To do so, a point estimator or statistic can be calculated. For example, if we wish to estimate the expected value of a random variable $\underline{X} \sim F$, an appropriate statistic would be the average or median of the sample. To determine the quality of this estimator

we can construct confidence intervals or perform hypothesis testing on the parameter. For this purpose, it is necessary to obtain some measure of accuracy.

Examples of procedures that are used to obtain accuracy measures include: Neyman-Pearson, which gives a uniformly most powerful test, Rao-Blackwell, which guarantees that certain estimators have minimum variance among all unbiased estimators, and Cramér-Rao which gives a bound on the variance of an unbiased estimator. In other cases, asymptotic theory can be used to construct approximate test statistics and confidence intervals. However, these procedures cannot always be used. If it is possible to obtain multiple samples from the population distribution (and hence determine the sampling distribution), Monte Carlo methods could be used. However, there is usually only a single sample available. An alternative strategy is the bootstrap (Efron, 1979).

The bootstrap is a computer-based resampling method that can be used to estimate the standard error of an estimator. This is achieved by estimating the sampling distribution of the statistic of interest. There is both a non-parametric and parametric bootstrap procedure. The non-parametric bootstrap can be used for inference about parameters in both parametric and non-parametric models. To demonstrate the applicability of the bootstrap procedure, consider using standard error to assess the accuracy of a statistic $t(\underline{x})$. In this notation $\underline{x} = (x_1, \dots, x_N)^T$, and $x_i \in \mathbb{R}$ for $i = 1, \dots, N$. The standard error yields a measure of the variability of an estimator around its expectation. If the standard error is small, the point estimate is good. Alternatively, if the standard error is large the point estimate is poor.

If the statistic of interest is the mean, i.e. $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$ of the sample the standard error is

$$\sqrt{\frac{s^2}{N}} \text{ with } s^2 = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N - 1}. \quad (2.50)$$

Hence there is no need for the bootstrap procedure in this scenario. However, for many estimators there is not a closed form solution. An example of such an estimator is the median. To obtain the standard error for the median, bootstrap samples $\underline{x}^* = (x_1^*, x_2^*, \dots, x_N^*)$ need to be generated. A bootstrap sample is obtained by randomly sampling N observations with replacement from the original sample. The standard error is estimated using a total of B bootstrap samples $\underline{x}^{*1}, \dots, \underline{x}^{*B}$. For each bootstrap sample a statistic $t(\underline{x}^{*b})$ is calculated. The bootstrap estimate of the standard error is calculated as the standard deviation of the bootstrap statistics as

$$\hat{s}_{boot} = \left[\sum_{b=1}^B (t(\underline{x}^{*b}) - \bar{t})^2 / (B - 1) \right]^{\frac{1}{2}}, \quad (2.51)$$

where $\bar{t} = \sum_{b=1}^B t(\underline{x}^{*b}) / B$.

The bootstrap procedure is a direct application of the plug-in principal. The plug-in principal uses an estimate of the population distribution to estimate some parameter of interest. Suppose that a parameter θ of the true distribution F needs to be estimated. The parameter of interest can be expressed as a function of the distribution as,

$$\theta = t(F). \quad (2.52)$$

To estimate (2.52) the sample of observations generated from F can be used to obtain

$$\hat{\theta} = t(\hat{F}), \quad (2.53)$$

by application of the plug-in principal. In (2.49) \hat{F} denotes an estimate of F . An example of \hat{F} is the empirical distribution function (EDF). The EDF is defined to be the distribution that puts probability mass $1/N$ on each of N observation in the sample. Therefore, the empirical distribution is defined as

$$\hat{F}(x) = \frac{1}{N} \sum_{j=1}^N I(x \leq x_j), \quad (2.54)$$

Therefore, \hat{F} is a discrete probability distribution. It can be shown that the EDF is a non-parametric MLE of the true underlying distribution. Further, the law of large numbers promises point wise convergence of $\hat{F} \rightarrow F$ as $N \rightarrow \infty$ (provided that the samples are iid from a population with finite variance for each observation in the support of F) (Ross, 2013). This holds since

$$\lim_{N \rightarrow \infty} N\hat{F}(\underline{x}) = \lim_{N \rightarrow \infty} \sum_{j=1}^N I(x \leq x_j) \sim \text{Bin}(N, F(\underline{x})). \quad (2.55)$$

In (2.55) $\hat{F}(\underline{x})$ tends to its expectation $F(\underline{x})$ as $N \rightarrow \infty$. This result confirms that \hat{F} is a consistent estimator for F .

In the parametric bootstrap it is assumed that a sample is generated from a parametric distribution. That is, $x_i \sim F_\theta$ in which case the bootstrap samples are generated from $F_{\hat{\theta}}$ where $\hat{\theta}$ is an estimate of θ .

2.6.1 Connection to Bayesian inference

Consider a discrete distribution with L possible categories. Let p_j be the probability that an observation belongs to category j with $j = 1, \dots, L$. Further, let \hat{p}_j be the empirical estimate of this probability p_j , i.e. the proportion of sample observations that belong to category j . Define the vectors $\underline{p} = (p_1, \dots, p_L)$ and $\underline{\hat{p}} = (\hat{p}_1, \dots, \hat{p}_L)$. Additionally, suppose that we wish to estimate $s(\underline{p})$; in which case $s(\underline{\hat{p}})$ is the estimate via the plug-in principal.

To derive a connection to Bayesian inference, select as a prior distribution for \underline{p} the symmetric Dirichlet distribution with parameter α . Hence, $\underline{p} \sim Di_L(\alpha \mathbf{1})$ which implies that the prior probability mass function is proportional to

$$\prod_{\ell=1}^L p_{\ell}^{\alpha-1}. \quad (2.56)$$

The posterior distribution is $\underline{p} \sim Di_L(\alpha \mathbf{1} + N\hat{\underline{p}})$ where N denotes the sample size. If the parameter $\alpha \rightarrow 0$, the prior becomes non-informative and the posterior is $\underline{p} \sim Di_L(N\hat{\underline{p}})$.

The bootstrap distribution is equivalent to the distribution obtained by sampling the category proportions from a multinomial distribution $N\hat{\underline{p}}^* \sim Mult(N, \hat{\underline{p}})$ where $Mult(N, \hat{\underline{p}})$ denotes a multinomial distribution with probability mass function

$$\binom{N}{\hat{p}_1^*, \dots, \hat{p}_L^*} \prod_{\ell} \hat{p}_{\ell}^{N\hat{p}_{\ell}^*}. \quad (2.57)$$

This is almost the same as the posterior distribution for \underline{p} obtained with a non-informative prior. It only differs in terms of the covariance matrix. It can therefore be concluded that the bootstrap distribution of $s(\hat{\underline{p}}^*)$ will be close to the posterior distribution of $s(\underline{p})$. Hence, the bootstrap distribution is an approximate, non-parametric and non-informative posterior distribution to the parameter of interest (Hastie *et al.*, 2009).

CHAPTER 3

COMBINATION STRATEGIES

3.1 INTRODUCTION

In this chapter methods for the combination of base learners into an ensemble are considered. There are two variants of combination: parallel combination and sequential combination. In the former, the aim is to exploit the independence of base learners to improve generalisation. In the later, the aim is equal, but through the exploitation of the dependence of base learners. Sequential methods, i.e. boosting, are not discussed in this thesis. Readers interested in learning about sequential methods can refer to Hastie *et al.*, (2009). In this chapter simple averaging and weighted averaging are discussed. In addition to this, stacking is discussed as it relates to the proposal of this thesis. In Section 3.6 of this chapter, a new proposal is given.

In parallel ensemble methods, the procedure is as follows: generate several base learners according to a certain strategy. For example, in bagging and random forest the base learners are constructed on bootstrap resampled versions of the original training data set. Subsequently, the predictions of the base learners are combined to obtain a single prediction. Dietterich (2000) argues that combination is beneficial due to three reasons, namely: the statistical issue, the computational issue and the representational issue.

The statistical issue is a consequence of insufficient data to be able to learn a model that is capable of good generalisation. If the quantity of training data is insufficient it is possible for a learning algorithm to return multiple models that explain the data equally well. Therefore, an incorrect model can be chosen – incorrect in the sense that the chosen model may turn out to have poor generalisation ability. Through combination, the risk of choosing the incorrect model is reduced. The statistical issue is presented in Figure 3.1. The outer curve represents the boundary of the space of all possible models that can be returned given a model class. The inner curve denotes the space of all possible models that are sufficiently accurate on the training data. The point f is the true underlying function that we desire to approximate. In Figure 3.1 averaging several approximations (denoted h_1, h_2, h_3, h_4) gives an approximation that is closer to f .

The computational issue refers to the idea that a learning procedure, which utilises a local search to approximate the true underlying function, may get stuck in local optima and hence will be unable to return the best possible model. The computational issue can occur regardless of whether there is enough training data – which means that the statistical issue has been avoided. Therefore, by running a learning procedure multiple times, using different starting points, it may be possible to construct a better approximation to f than would be possible by an individual model. The computation issue is also given in the Figure 3.1. Again, the outer boundary represents the space of

possible models that can be returned given the model class. The dotted lines represent the search path taken by a learning algorithm when trying to approximate f . The global optimum is located at f , hence the combination of multiple search paths may get closer to f .

The final issue that Dietterich (2000) alludes to is the representational issue. In the representational issue a model class is unable to adequately represent the true underlying function. In such a scenario, by combining models, it may be possible to expand the space that can be represented by the model class. The representation issue is also given in Figure 3.1. Now the true underlying function lies outside of the space that can be represented by the chosen model class. By averaging it may be possible to expand the space, thereby returning a better approximation to f .

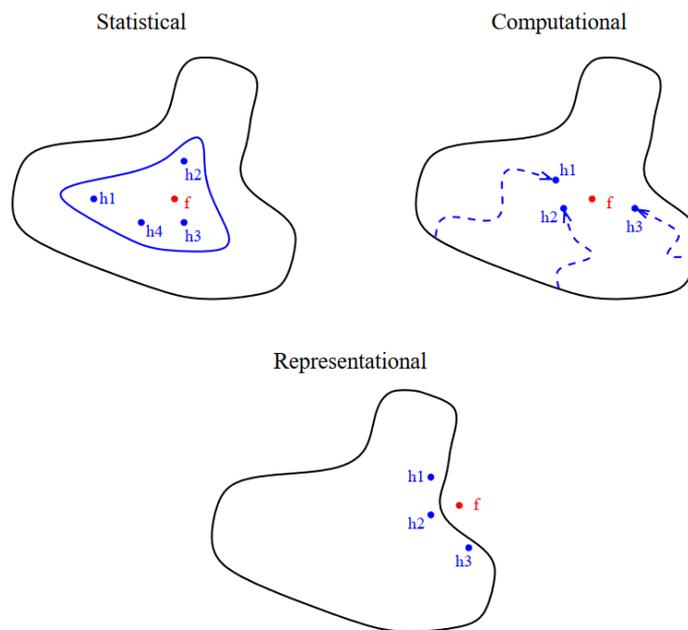


Figure 3.1: Three motivating reasons for why combination methods are an appropriate strategy in the context of ensemble learning.

Source: (Dietterich, 2000)

The statistical issue is associated with models that have high variance. The computational issue is common among models that have high computational variance (dependence on starting values). Methods that suffer from the representation issue are said to have high bias. It has been shown empirically by Xu *et al.*, (1992), Bauer and Kohavi (1999) and Opitz and Maclin (1999) that combination methods can reduce both variance and bias.

Unless otherwise indicated the material in this chapter has been adapted from Hastie *et al.*, (2009) and Zhou (2012).

3.2 AVERAGING

In this section let $\hat{f}^i, i = 1, \dots, B$ denote a set of base learners that have been learned from the training data. In addition, let f denote the true underlying function that we wish to approximate. Hence, $\hat{f}^i, i = 1, \dots, B$ are approximations to f .

3.2.1 Simple averaging

Simple averaging involves equally weighting the predictions of the base learners in an ensemble. The form of a simple averaging ensemble is given by

$$\hat{f}(\underline{x}) = \frac{1}{B} \sum_{i=1}^B \hat{f}^i(\underline{x}). \quad (3.1)$$

Two ensemble methods that utilise simple averaging are bagging and random forest. These techniques are discussed in Sections 3.3 and 3.4, respectively.

3.2.2 Weighted averaging

Weighted averaging builds an ensemble by averaging the outputs of the base learners using different weights, or rather, model specific weights. Hence, simple averaging is a special case of weighted averaging. The weights can be interpreted as a measure of importance given to an individual model. This means that in simple averaging each model is deemed to be as important as any other model in the ensemble. The weighted ensemble is given as

$$\hat{f}(\underline{x}) = \sum_{i=1}^B w_i \hat{f}^i(\underline{x}). \quad (3.2)$$

Typically, the weights are constrained by

$$w_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^B w_i = 1. \quad (3.3)$$

Perrone and Cooper (1993) showed that an ensemble using weighted averaging has a MSE given by

$$\begin{aligned} MSE &= \int_{-\infty}^{\infty} \left(\sum_{i=1}^B w_i \hat{f}^i(\underline{x}) - f(\underline{x}) \right)^2 p(\underline{x}) d\underline{x} \\ &= \int_{-\infty}^{\infty} \left(\sum_{i=1}^B w_i \hat{f}^i(\underline{x}) - f(\underline{x}) \right) \left(\sum_{j=1}^B w_j \hat{f}^j(\underline{x}) - f(\underline{x}) \right) p(\underline{x}) d\underline{x} \\ &= \sum_{i=1}^B \sum_{j=1}^B w_i w_j C_{ij}, \end{aligned} \quad (3.4)$$

where $C_{ij}, i = 1, \dots, B, j = 1, \dots, B$ is a covariance matrix defined according to

$$C_{ij} = \int_{-\infty}^{\infty} (\hat{f}^i(\underline{x}) - f(\underline{x})) (\hat{f}^j(\underline{x}) - f(\underline{x})) p(\underline{x}) d\underline{x}. \quad (3.5)$$

The optimal weights can be found by solving

$$\underline{w} = \arg \min_{\underline{w}} \sum_{i=1}^B \sum_{j=1}^B w_i w_j C_{ij}. \quad (3.6)$$

Through application of the method of Lagrange multipliers, we get

$$w_i = \frac{\sum_{j=1}^B C_{ij}^{-1}}{\sum_{k=1}^B \sum_{j=1}^B C_{kj}^{-1}}. \quad (3.7)$$

In (3.7) the notation C_{ij}^{-1} refers to element i, j of the inverse covariance matrix C . Hence, the weights can be solved in closed form. However, as can be seen from (3.7) the solution requires a non-singular covariance matrix C (as we are required to invert this matrix). Typically, C is not non-singular since the models in the ensemble are highly correlated. Indeed, the covariance matrix is usually singular or ill-conditioned which means that (3.7) is infeasible or unstable. In addition, to obtain the optimal weights in (3.7), we require access to the population distribution.

It turns out that any ensemble method can be regarded as a procedure that uses a variant of weighted averaging. Empirical studies conducted by Xu *et al.*, (1992), Ho *et al.*, (1994) and Kittler *et al.*, (1998) do not show evidence that weighted averaging is superior to simple averaging in all applications. These authors note, in particular, that the reason is due to the nature of the data - which is often noisy in practical applications. Hence, overfitting can easily occur in large weighted ensembles. Simple averaging avoids this problem as the weight does not need to be estimated. In general, if the models in the ensemble have similar performances, then simple averaging works well. On the other hand, if the models exhibit differing degrees of performance, weighted averaging may achieve better results.

3.3 BAGGING

Bagging (Breiman, 1996) is an ensemble technique that leverages bootstrap aggregation to improve predictive accuracy. It achieves this by reducing variance through combination. In the broader context of ensemble methods, bagging is a parallel learning technique. This means that the procedure exploits the independence between base learners that constitute the bagging ensemble. Recall from Subsection 2.6.1 that the bootstrap distribution represents an approximate, non-parametric and uninformative posterior distribution for a statistic. Therefore, bootstrap aggregation gives an approximate posterior average. The bagging procedure exploits this connection. However, using the bootstrap procedure has the added advantage that the posterior distribution can be obtained without having to specify a prior or sample from the posterior (Hastie *et al.*, 2009).

In addition to Hastie *et al.*, (2009) and Zhou (2012), James *et al.*, (2013) and Friedman and Hall (2000) has been consulted for this section.

3.3.1 Properties of bagging

Consider the setting of a regression problem. Recall that $\underline{X} \in \mathbb{R}^p$ and $Y \in \mathbb{R}$ are random variables with joint probability distribution $P(\underline{X}, Y)$. Let $\mathcal{T} = \{(\underline{x}_i, y_i), i = 1, \dots, N\}$ be a set of iid training observations. The significance of the iid assumption in this context ensures that the joint distribution $P(\underline{X}^{*b}, Y^{*b})$ for $b = 1, \dots, B$ of each bootstrap sample is the same as the distribution of the original training observations.

The idea of bagging is to fit multiple models using a different bootstrap sample for each model. Typically, the bootstrap samples are obtained using the non-parametric bootstrap procedure. Each model (built on a different bootstrap sample) gives a prediction $\hat{y}_0^b = \hat{f}^b(\underline{x}_0)$ for an observation \underline{x}_0 . The superscript b clarifies the index of the bootstrap sample that was used to construct the base learner. The learners/models that compose an ensemble are called base learners. The prediction for an observation \underline{x}_0 (using a total of $B \in \mathbb{N}$ models in the bagging ensemble) is

$$\hat{f}^B(\underline{x}_0) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(\underline{x}_0). \quad (3.8)$$

The bagging procedure is illustrated in Figure 3.2.

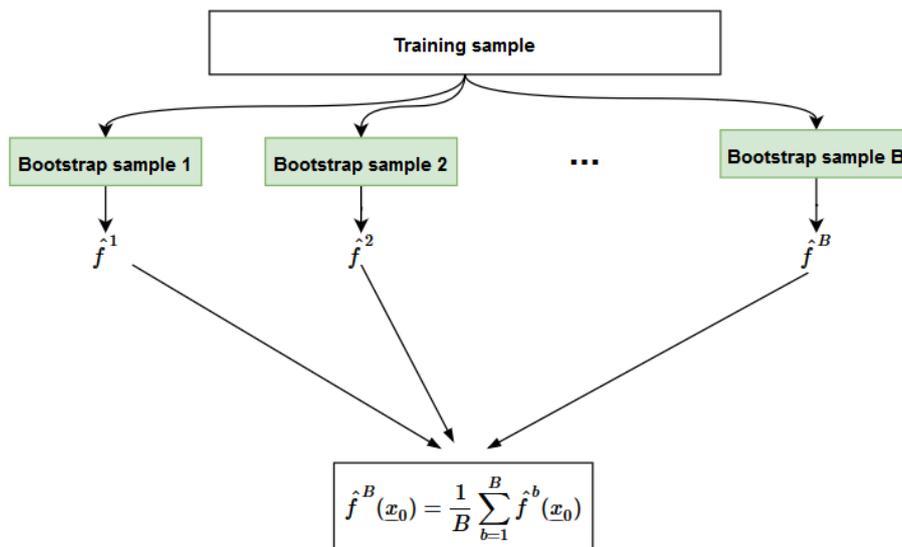


Figure 3.2: The bagging procedure.

The success of bagging relies on the instability of the base learners in the ensemble (Friedman and Hall, 2000). To clarify, consider the following deduction: Let a model constructed on a random training set be denoted by \hat{f} . An aggregated model is defined as

$$\hat{f}^*(x_0) = E_{\mathcal{T}}(\hat{f}(x_0)), \quad (3.9)$$

where $E_{\mathcal{T}}$ denotes the expectation with respect to \mathcal{T} , a random training sample. The prediction error, using the squared errors loss, for the aggregate estimator is

$$err^* = E_{Y, \underline{X}}(Y - \hat{f}^*(\underline{X}))^2. \quad (3.10)$$

The average prediction error of this aggregate estimator is

$$err = E_{\mathcal{T}} E_{Y, \underline{X}}(Y - \hat{f}(\underline{X}))^2 = E_{Y, \underline{X}}(Y^2) - 2E_{Y, \underline{X}}(Y \hat{f}^*(\underline{X})) + E_{Y, \underline{X}} E_{\mathcal{T}}(\hat{f}(\underline{X})^2). \quad (3.11)$$

By using the general result $(E_Z(Z))^2 \leq E_Z(Z^2)$ for a random variable Z , it follows that

$$err \geq E_{Y, \underline{X}}(Y^2) - 2E_{Y, \underline{X}}(Y \hat{f}^*(\underline{X})) + E_{Y, \underline{X}}(\hat{f}^*(\underline{X})^2) = E_{Y, \underline{X}}(Y - \hat{f}^*(\underline{X}))^2 = err^*. \quad (3.12)$$

The result in (3.12) suggests that \hat{f}^* has a lower mean squared error than \hat{f} with the difference being dependent on

$$\hat{f}^*(\underline{X})^2 \leq E_{\mathcal{T}}(\hat{f}(\underline{X})^2). \quad (3.13)$$

Therefore, the greater the variability in \hat{f} , the more effective bagging will be. This is an important point to acknowledge. It implies that perturbation of the training sample should cause a substantial change in the model's structure for bagging to be effective.

The prediction in (3.8) is a Monte Carlo estimate. If $B \rightarrow \infty$, (3.8) converges to

$$E_{\mathcal{T}^*}(\hat{f}_{\mathcal{T}^*}(\underline{X})), \quad (3.14)$$

where $\mathcal{T}^* = \{(\underline{x}_i^*, y_i^*), i = 1, \dots, N\}$ and $(\underline{X}^*, Y^*) \sim \hat{P}(\underline{X}, Y)$. $\hat{P}(\underline{X}, Y)$ is the EDF which assigns probability mass $1/N$ to each observation (\underline{x}_i, y_i) in the original sample. If the base learner is a stable function of the data, i.e. $\hat{f}^* \approx \hat{f}$, then it is possible that bagging will lead to a degradation in predictive ability.

To further motivate aggregation over independent estimators, consider a binary classification problem with the response $y \in \{-1, 1\}$. Suppose that the true estimator is f and that each base classifier has an independent generalisation error ϵ . Thus, for each of B base classifiers $f^b, b = 1, 2, \dots, B$, it holds that $P(f^b(\underline{x}) \neq f(\underline{x})) = \epsilon$ for an observation \underline{x} .

The bagged classifier is constructed by combining the base learners using

$$\hat{f}^B(\underline{x}) = \text{sign}\left(\sum_{b=1}^B \hat{f}^{*b}(\underline{x})\right), \quad (3.15)$$

where $\text{sign}(x)$ is defined as

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0. \\ 0 & x = 0 \end{cases} \quad (3.16)$$

Therefore, a misclassification occurs if at least half of the base learners make an error. The generalisation error can be bounded to

$$P\left(\hat{f}^{*B}(\underline{x}) \neq f(\underline{x})\right) = \sum_{k=0}^{\lfloor B/2 \rfloor} \binom{B}{k} (1-\epsilon)^k \epsilon^{B-k} \leq \exp\left(-\frac{1}{2}B(2\epsilon-1)^2\right), \quad (3.17)$$

by applying the Hoeffding inequality (Hoeffding, 1963). From (3.17) it follows that $\hat{f}^{*B}(\underline{x}) \rightarrow f(\underline{x})$ in probability as $B \rightarrow \infty$. In practice, however, due to limited training data being available it is not possible to obtain perfectly independent base learners.

In Friedman and Hall (2000) the decomposition of estimators into linear and higher order terms is studied. The authors show that bagging reduces the variability of the nonlinear elements that compose a statistical estimator by replacing these terms with estimates of their expected value. This implies that bagging leaves the linear component of an estimator unaffected. Hence, the usefulness of bagging is restricted to the class of nonlinear estimators, such as decision trees or estimators for which perturbation of the training samples causes a noticeable change in the structure of the model. It is further argued by the authors that there is an equivalent geometrical interpretation of this result. Higher order terms represent stochastic “bumps” on the parabolic loss surface of the objective function. Bagging replaces these bumps with an expected value using an empirical approximation. This causes the parabolic surface to become more regular. Hence, multiple local optima are replaced to make the surface of the objective function smoother and hereby reduce the difficulty of finding the global optimum. The global optimum value is achieved at the linear component of the estimator. In comparison, regularisation techniques affect both the linear and nonlinear elements of the estimator.

A further benefit of using the bootstrap procedure in bagging is that it allows for approximating the test error without using CV. This strategy is called out of bag (OOB) error estimation. Breiman (1996a) showed that the probability of the i^{th} training example being selected has a Poisson distribution with $\lambda = 1$. Therefore, the probability that the i^{th} training example will occur at least once is $1 - 1/e \approx 0.632$. Hence, approximately 36.8 percent of the original training examples will be absent from the bootstrap sample. Cases not included in a bootstrap data set are referred to as the

out of bag samples. Since these samples were not used to construct the model, they can be used to obtain an estimate for the test error.

Breiman (1994) states that fewer bootstrap replications are required for regression problems compared to classification problems. Further, he says that more bootstrap replications are required as the number of classes in a classification problem increases. In conclusion, we have shown that in general, averaging over identically distributed estimates reduces variance, and therefore yields a more stable estimate which improves prediction accuracy.

3.3.2 Building a bagged regressor in Python

The parameters of the scikit-learn API (Pedregosa *et al.*, 2011) for a bagged regressor include:

base_estimator: (default=None)

The base estimator to fit on random subsets of the dataset.

n_estimators: (default=10)

The number of base estimators in the ensemble.

base_estimator_criterion: (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the ℓ_2 loss using the mean of each terminal node, "friedman_mse", which uses mean squared error with Friedman's improvement score for potential splits, and "mae" for the mean absolute error, which minimizes the ℓ_1 loss using the median of each terminal node.

base_estimator_splitter: (default="best")

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

base_estimator_max_depth: (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than `min_samples_split` samples.

Note that the `DecisionTreeRegressor` function can be used as a base estimator.

3.4 RANDOM FOREST

In this section attention will be given to an adaptation to bagging called random forest (Breiman, 2001). To begin, recall that the bootstrap samples used to generate the trees in bagging are identically distributed. This means that the bias of the bagged ensemble is the same as that of the individual trees. However, an average of B iid random variables each having variance σ^2 , will have variance $\frac{1}{B}\sigma^2$. If the variables are not independent and instead have positive pairwise correlation ρ , then the variance of the averages is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \quad (3.18)$$

In (3.18) it is clear that as B increases in size the variance tends to $\rho\sigma^2$. Therefore, the size of the correlation between the trees in bagging limits the extent to which the variance can be reduced. As noted previously, there is often a strong correlation between the trees in bagged ensembles.

The idea in random forests is to combat this problem by reducing the correlation structure between the trees without increasing the variance too much. To make a prediction for a new point \underline{x} , in the context of a regression problem, and assuming B base learners, the function is

$$\hat{f}^{RF}(\underline{x}) = \frac{1}{B} \sum_{b=1}^B T_b(\underline{x}). \quad (3.19)$$

The difference between (3.19) and (3.8) lies in the manner in which the base learners T_b are constructed. Algorithm (3.1) highlights this difference.

Algorithm 3.1: Random forest (regression)

-
1. **for** $b = 1, \dots, B$:
 2. generate a bootstrap sample X^* of size N from the training data \mathcal{T}
 3. **repeat**: grow a random forest T_b to the bootstrap sample using recursive binary splitting:
 4. select m variables at random from the p variables
 5. pick the best variable/split-point among the m
 6. split the node into two child nodes
 7. **stop**: if the minimum terminal node size is n_{min}
 8. **end**: output the ensemble of trees T_1, \dots, T_B
-

In step 4 of Algorithm 3.1 a subset of predictors, on which to perform a split, is randomly selected. In this manner random forest attempts to reduce the correlation between the individual trees. Typically, $m = \sqrt{p}$ or even $m = 1$ where p is the number of predictors. As was the case in bagging,

not all estimators benefit from this randomisation procedure. Only highly non-linear estimators benefit from this modification.

3.4.1 Details about random forest

For regression modelling it is recommended that the default value for m is $\lfloor p/3 \rfloor$ and the minimum node size $n_{min} = 5$. However, these are complexity parameters that can be calibrated using the K-fold CV procedure.

3.4.1.1 Out of bag samples

The OOB error estimate for random forest is equivalent to the error estimate obtained by K-fold CV. To calculate the OOB error, the model error is calculated using observations that were not present in the construction of random forest. Thus, for each observation, and its corresponding response, i.e. $(x_i, y_i), i = 1, \dots, N$, we calculate the error using a random forest that is constructed excluding that observation. This process gives N error estimates which are subsequently averaged to obtain the OOB error.

3.4.1.2 Variable importance

It is possible to make a plot of a variable's importance when using a random forest. To do so, for each split used in the construction of the tree, the improvement in the split criterion is recorded. Recall that this is given in (2.40). This yields a measure of the importance associated with a splitting variable. These importance measures are accumulated over all trees in the forests and for each variable. The result can be compiled into a plot that will visually allow the reader to infer which variables the base learners deem important.

The OOB samples can also be used to construct an importance plot depicting the predictive strength of the variables in the data. For the b th tree, a prediction is made using the OOB samples. Subsequently, the value for the j th variable in the OOB observations is randomly permuted and another prediction is made. The change in accuracy caused by permuting the j th variable is recorded. This process is carried out for all variables and the results are averaged over all trees. This gives an estimate of the predictive importance for each variable in random forest.

3.4.1.3 Proximity plot

During the process of fitting a random forest a $N \times N$ proximity matrix can be calculated on the training data. This matrix is constructed using the OOB samples. Every pair of OOB observations that are assigned to the same terminal node when making a prediction have their proximity increased by 1. Thus, the entry at position i, j in the proximity matrix is incremented by 1. The indices i, j refer to the pair of OOB observations. The proximity matrix is depicted in a 2-dimensional space using multi-dimensional scaling (MDS). This plot yields an indication of which observations are close (as determined by random forest) in a high dimensional space.

Proximity plots often look similar regardless of the data on which they are constructed and so there is doubt as to their usefulness.

3.4.1.4 Random forest and overfitting

If the number of variables is large but the number of relevant variables is small, random forest is likely to perform poorly when m is small. This is because the chance that a relevant variable will be selected for a split is low. On the other hand, as the number of relevant variables increases, the performance of random forest is sturdy regardless of an increase in the number of noisy variables.

3.4.2 Building a random forest in Python

The parameters used to fine tune random forest regressor in the scikit-learn API (Pedregosa *et al.*, 2011) include

n_estimators: (default=10)

The number of trees in the forest.

criterion: (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to the variance reduction as a feature selection criterion, and "mae" for the mean absolute error.

max_depth: (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

base_estimator_criterion: (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the ℓ_2 loss using the mean of each terminal node, "friedman_mse", which uses mean squared error with Friedman's improvement score for potential splits, and "mae" for the mean absolute error, which minimizes the ℓ_1 loss using the median of each terminal node.

base_estimator_splitter: (default="best")

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

base_estimator_max_depth: (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than `min_samples_split` samples.

3.5 STACKING

The fundamental idea in stacking (Wolpert, 1992) involves training a model to combine other trained models. The model used to combine other models in the ensemble is called a second level learner or meta learner. The models that are combined by the meta learner are called first-level learners. The construction of a stacked ensemble requires two training phases. In the first phase, the first level learners are trained using the original training data set – both the original predictors and their corresponding responses. Subsequently, the first level models are used to create a new data set which is used as input into the meta learner during the second phase of training. The secondary data set is constructed using predictions from the first level learners. However, the meta model still uses the original response during the secondary training phase. To create the secondary data set, it is not advisable to use the same data set that was used to train the first level learners as this approach would lead to overfitting. Instead, a better approach is to use the predictions obtained from a CV procedure such as K-fold CV.

To build a stacked ensemble using K-fold CV the original training data set $\mathcal{T}: N \times p$, where N is the number of observations and p is the number of predictors, is randomly split into k parts $\mathcal{T}_1: N_1 \times p, \dots, \mathcal{T}_k: N_k \times p$ with approximately an equal number of observations in each part. Let the original training data set with the j th part removed be defined as $\mathcal{T}_{(-j)} = \mathcal{T} \setminus \mathcal{T}_j$. Each first level learner is trained on $\mathcal{T}_{(-j)}$ and predictions for each of these models are obtained using the held-out data set \mathcal{T}_j . The predictions obtained for each model during the K-fold CV procedure are then combined to form a new data set, say Z . The matrix Z is of size $N_j \times B$ where B is the number of first level learners. After Z has been formed the first level learners are retrained on the entire data set \mathcal{T} . The second level learner is trained using Z and the original response.

Breiman (1996b) affirmed the success of stacking in the context of regression problems. He considered, as first level learners, regression trees of different sizes and linear regression models with a differing number of variables. As meta models he used least squares linear models with a restriction that regression coefficient should be positive. Importantly, this constraint was shown to be vital to the success of the procedure.

Stacked classification was also studied by Wolpert (1992). Wolpert noted that it is important to consider the features used to construct the new training data as well as the types of learning algorithms used for training the meta model. Ting and Witten (1999) suggest that class probabilities are more amenable than crisp class labels as input into the meta learning phase. The authors

indicate that this approach allows the meta learner to consider the confidences of the individual classifiers. Furthermore, Ting and Witten (1999) suggest using multi-response linear regression as the meta learner.

3.6 PROPOSED STRATEGY

In this section the proposed contribution for this thesis is introduced. The proposal is a weighted averaging ensemble using ℓ_2 regularisation. Furthermore, the weights are constrained to be positive and sum to one. The proposed adaptation is applied to bagging and random forest giving two new weighting strategies. In the discussion, a motivation will be provided for this proposal. First, a derivation will be given which shows how the optimal weights can be obtained. The approach taken in this thesis relies on methods from convex optimisation which is the subject of Chapter 4.

3.6.1 Derivation

Before the derivation for finding the optimal weights is outlined, it is necessary to be clear about the context in which the proposal is made. Since the proposal is a weighted averaging ensemble, we restrict our attention to regression problems. For the purpose of model fitting and assessment, we apply the MSE criterion. Thus, the optimal weights are optimal in a least squares sense. Furthermore, as base learners we consider regression trees. Each regression tree is trained on bootstrap resampled version of the original training data set and the complexity parameters are optimised using fivefold CV. The results given in Chapter 6 is the average MSE over 10 experiments. More specific details regarding the experimental design is discussed in Chapter 5.

To establish our proposed weighted ensemble method, it is necessary to setup an optimisation problem that will yield a set of weights that minimises the MSE between the response and the weighted predictions of the base learners. Let $T^b, b = 1, \dots, B$ denoted a set of regression trees that have been trained on bootstrap resampled versions of the original training sample. Further, let $w_b, b = 1, \dots, B$ denote the weight assigned to each of these tree models. Thus, the weighted ensemble is

$$f(\underline{x}_i) = \sum_{b=1}^B w_b T^b(\underline{x}_i). \quad (3.20)$$

Consider the problem of solving for the optimal weights $w_b, b = 1, \dots, B$ with the constraint that $\sum_{b=1}^B w_b = 1$. We can apply the method of Lagrangian multipliers, to ensure that the constraint is enforced. The optimisation problem in this respect is

$$\mathcal{L}(\underline{w}, \lambda) = \sum_{i=1}^N \left(y_i - \sum_{b=1}^B w_b T^b(\underline{x}_i) \right)^2 + \lambda \left(\sum_{b=1}^B w_b - 1 \right). \quad (3.21)$$

In (3.21) N denotes the number of training observations and λ denotes the Lagrangian multiplier which enforces the constraint that the optimal weights sum to one. We can rewrite this problem using matrix notation as

$$\mathcal{L}(\underline{w}, \lambda) = (\underline{y} - Z\underline{w})^T (\underline{y} - Z\underline{w}) + \lambda(\underline{1}^T \underline{w} - 1), \quad (3.22)$$

where Z is a matrix of size $N \times B$ that has been constructed by stacking the predictions of the B tree models on N observations column wise. Note that this is the method used in stacking for fitting the meta learner. To find the solution to (3.22) we need to determine the partial derivative of (3.22) with respect to the weights and set the result equal to 0 The partial derivative is

$$\frac{\partial \mathcal{L}(\underline{w}, \lambda)}{\partial \underline{w}} = -2Z^T (\underline{y} - Z\underline{w}) + \lambda \underline{1}. \quad (3.23)$$

By setting (3.23) equal to zero we get

$$\begin{aligned} Z^T (\underline{y} - Z\underline{\hat{w}}) &= \frac{1}{2} \hat{\lambda} \underline{1}. \\ \Rightarrow -Z^T Z \underline{\hat{w}} &= \frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \Rightarrow \underline{\hat{w}} = -(Z^T Z)^{-1} \left(\frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \right). \end{aligned} \quad (3.24)$$

To ensure that that the constraint is enforced, it is necessary to consider the partial derivative of (3.22) with respect to λ . Thus, we get

$$\frac{\partial \mathcal{L}(\underline{w}, \lambda)}{\partial \lambda} = \underline{1}^T \underline{w} - 1. \quad (3.25)$$

By setting (3.25) to 0 we get that $\underline{1}^T \underline{\hat{w}} = 1$. To solve for $\hat{\lambda}$ we apply the constraint condition in (3.25) and use the final result in (3.24) to obtain

$$\begin{aligned} 1 &= \underline{1}^T \underline{\hat{w}} = -\underline{1}^T (Z^T Z)^{-1} \left(\frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \right) \\ \Rightarrow 1 - \underline{1}^T (Z^T Z)^{-1} Z^T \underline{y} &= -\frac{1}{2} \hat{\lambda} \underline{1}^T (Z^T Z)^{-1} \underline{1} \\ \Rightarrow \hat{\lambda} &= \frac{1 - \underline{1}^T (Z^T Z)^{-1} Z^T \underline{y}}{-\frac{1}{2} \underline{1}^T (Z^T Z)^{-1} \underline{1}}. \end{aligned} \quad (3.26)$$

Therefore, to obtain the optimal weights that minimise (3.21) we can substitute $\hat{\lambda}$ into the final result in (3.24) to get

$$\underline{\hat{w}} = -(Z^T Z)^{-1} \left(\frac{1}{2} \left(\frac{1 - \underline{1}^T (Z^T Z)^{-1} Z^T \underline{y}}{-\frac{1}{2} \underline{1}^T (Z^T Z)^{-1} \underline{1}} \right) \underline{1} - Z^T \underline{y} \right). \quad (3.27)$$

Consider solving for the minimising weights in (3.22) but including a ℓ_2 penalisation term. This implies that we need to consider

$$\mathcal{L}(\underline{w}, \lambda, \xi) = (\underline{y} - Z\underline{w})^T (\underline{y} - Z\underline{w}) + \lambda(\underline{1}^T \underline{w} - 1) + \xi \underline{w}^T \underline{w}. \quad (3.28)$$

In (3.28) ξ denotes the shrinkages parameter that regulates the complexity of the weighted ensemble by constraining the size of the optimal weights. Following the same procedure used previously, we get

$$\begin{aligned} \frac{\partial \mathcal{L}(\underline{w}, \lambda, \xi)}{\partial \underline{w}} &= -2Z^T (\underline{y} - Z\underline{w}) + \lambda \underline{1} + 2\xi \underline{w} \\ &\Rightarrow -Z^T Z \underline{w} - \xi \underline{w} = \frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \\ &\Rightarrow -(Z^T Z + \xi I) \underline{w} = \frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \\ &\Rightarrow \underline{w} = -(Z^T Z + \xi I)^{-1} \left(\frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \right). \end{aligned} \quad (3.29)$$

Furthermore, from the partial derivative of (3.28) with respect to λ we get

$$\frac{\partial \mathcal{L}(\underline{w}, \lambda, \xi)}{\partial \lambda} = \underline{1}^T \underline{w} - 1, \quad (3.30)$$

from which it follows that

$$1 = \underline{1}^T \underline{w} = -\underline{1}^T (Z^T Z + \xi I)^{-1} \left(\frac{1}{2} \hat{\lambda} \underline{1} - Z^T \underline{y} \right). \quad (3.31)$$

Thus, we get

$$\hat{\lambda} = \frac{1 - \underline{1}^T (Z^T Z + \xi I)^{-1} Z^T \underline{y}}{-\frac{1}{2} \underline{1}^T (Z^T Z + \xi I)^{-1} \underline{1}}. \quad (3.32)$$

From (3.29) it follows that

$$\underline{w} = -(Z^T Z + \xi I)^{-1} \left(\frac{1}{2} \left(\frac{1 - \underline{1}^T (Z^T Z + \xi I)^{-1} Z^T \underline{y}}{-\frac{1}{2} \underline{1}^T (Z^T Z + \xi I)^{-1} \underline{1}} \right) \underline{1} - Z^T \underline{y} \right). \quad (3.33)$$

The final constraint to impose will ensure that the optimal weights are positive. Unlike the previous cases, the positivity constraint means that a closed form solution for the weights cannot be obtained. Hence, consider the formulation of an optimisation problem within the context of quadratic programming. The standard form for such a problem is given by

$$\begin{aligned} & \text{minimise } \frac{1}{2} \underline{x}^T P \underline{x} + q^T \underline{x} \\ & \text{subject to } G \underline{x} \leq h \\ & \quad A \underline{x} = \underline{b}. \end{aligned} \tag{3.34}$$

Therefore, given the criterion in (3.28) with the constraint that $\hat{w}_b \geq 0, b = 1, \dots, B$, we can formulate an appropriate problem as

$$\begin{aligned} & \text{minimise } \frac{1}{2} \underline{w}^T (Z^T Z + \xi I) \underline{w} + \left(- (Z^T \underline{y})^T \underline{w} \right) \\ & \text{subject to } -I \underline{w} \leq \underline{0} \\ & \quad \underline{1}^T \underline{w} = 1, \end{aligned} \tag{3.35}$$

by multiplying the objective in (3.28) by $\frac{1}{2}$. From the result in (3.35) it can be seen that

$$\begin{aligned} P &= (Z^T Z + \xi I) \\ q^T &= - (Z^T \underline{y})^T \\ G &= -I \\ A &= \underline{1}^T \\ b &= 1 \end{aligned} \tag{3.36}$$

The problem in (3.35) can be solved using procedures from the field of convex optimisation. The details of how this can be done will be discussed extensively in the Chapter 4.

3.6.2 Motivation

The combination strategy proposed in this thesis differs from previously outlined, averaging, methods because it utilises regularisation, specifically ℓ_2 regularisation as specified in (3.28), and optimal weighting. The purpose of introducing a regularisation parameter is two-fold.

Firstly, regularisation makes it possible to solve the often-ill-condition optimisation problem that is a requisite to finding optimal weights. Recall the discussion on weighted averaging given in Subsection 3.2.2. It is noted that the optimal averaging weights are not easily solved, because the solution relies on inverting a covariance matrix which is often singular or ill-conditioned. The specific criteria for finding the optimal weights is given in (3.9) and (3.10). To overcome this problem, refer to the

discussion in Section 2.4 on regularisation. In this discussion it is stated that regularisation is a useful technique to overcome issues associated with optimisation problems that are ill-conditioned or undetermined (an undetermined problem does not have a unique solution). Further, recall that ℓ_2 regularisation solves this issue by adding a constant term to the diagonal entries of the Gram matrix $X^T X$ before the inverse is taken. In our proposed weighted ensemble, to determine the optimal weights, a matrix Z is constructed which consists of the predictions of the B base learners stacked column wise. This is similar to the procedure used in stacking, whereby a secondary data set is created for the purpose of fitting the second level learner. As shown in (3.27), Z needs to be inverted to find a set of optimal weights. Therefore, we need to assess $(Z^T Z)^{-1}$. Since the models (whose predictions compose the columns of the matrix Z) have been trained for the same purpose, $(Z^T Z)^{-1}$ will be ill-conditioned due to high multicollinearity. This suggests that computing the inverse might not be possible. If it can be computed, the solution is likely to be highly sensitive to small changes in the data and will possibly be inaccurate. Furthermore, in the event that $B > N$ the optimisation problem will not be feasible at all. This would occur if the number of models in the ensemble exceeded the size of the training set. Regularisation overcomes both of these issues.

The second reason for the introduction of the regularisation term is to facilitate flexibility in controlling the bias-variance trade-off. Consider the discussion on bagging in Section 2.7. The purpose of bagging is to reduce variance through averaging. Averaging will hopefully result in the bagged ensemble having a better generalisation ability than any one of the models that constitute it. Recall that regression trees are used as base learners in the proposed weighted ensemble. Regression trees are a non-linear method with low bias and high variance. In this case bagging is a useful technique for reducing variance. In Section 2.4 we note that regularisation is a method that can be used to control the bias-variance trade-off. Therefore, by using weights, where the size of the weights is controlled by regularisation, it is possible to control the fit of the final ensemble, i.e. the extent to which bias is traded for a reduction in variance. The aim is to find an optimal value for the shrinkage parameter so that the generalisation ability of the weighted ensemble will improve upon the generalisation ability of a bagged ensemble.

3.6.3 Building a weighted ensemble in Python

The purpose of the details in this subsection are to provide the reader with information regarding how the proposed weighted ensemble is implemented in the source code. A nice aspect of open source API's, such as the scikit-learn API (Pedregosa *et al.*, 2011), is that the details of the implementation (including the source code) are made freely available to the public. Therefore, it is most convenient for our purposes to adjust the implementation of the bagging regressor class in the scikit-learn API to use optimal regularised least squares weights as opposed to averaging. The adjustment to do so is quite simple. In Figure 3.3 is a code snippet that is used to find the optimal least squares weights.

```

def _get_weights(self, X, y):
    """Private function to find a set of weights
       that are optimal in a least squares sense.
       The weights incorporate the l2 penalty term
       and are positive.
    """
    X = matrix(scale(X))
    y = matrix(y)
    m, n = X.size
    I = matrix(0.0, (n, n))
    I[:,n + 1] = 1.0
    G = matrix([-I, matrix(0.0, (1, n)), I])
    h = matrix(n * [0.0] + [self.c] + n * [0.0])
    dims = {'l': n, 'q': [n + 1], 's': []}
    T = matrix(1.0, (1, n))
    s = matrix(1.0)

    return np.asarray(solvers.coneqp(X.T * X, -X.T * y, G, h, dims, T,
                                   s) ['x'])

```

Figure 3.3: Source code that is used to solve for the optimal least squares weights in the proposed weighted ensemble.

The input into the `_get_weights` function is a matrix Z and the response vector we wish to predict, y . Note that the parameters X and y in the `_get_weights` function are placeholders for these inputs. The matrix Z is of the form described in Section 3.5.1. Furthermore, from the code snippet in Figure 3.3 the reader will note the function `solvers.coneqp`. This function is part of the CVXOPT Python library (Andersen *et al.*, 2015). It is an implementation of the convex optimisation solver used to find the weights. The algorithm used is a primal-dual path-following algorithm. Details regarding this algorithm will be discussed at length in Chapter 4.

The columns of the matrix X have been scaled before solving for the weights. Scaling of the features is recommended when applying ridge regression because if the features are measured on different scales, they will have different contributions to the penalisation term. This will cause the shrinkage to be disproportionate across all variables. The parameter `self.c` is an object dependent reference to the shrinkage parameter used in the least squares weights. This parameter is fed into the `_get_weights` function by the `GridSearchCV` protocol. `GridSearchCV` is a scikit-learn implementation of an exhaustive K-fold CV search procedure, and it is discussed in detail in Chapter 5.

The major benefit of integrating the `get_weights` function into the existing scikit-learn API for a bagging regressor is that minimal work is required to make it possible to optimise the entire weighted ensemble. One can easily use existing scikit-learn optimisation functions (in particular

`GridSearchCV`) to find the optimal combination of parameters for the base learners and weights. `GridSearchCV` relies on specific implementation details regarding the model that needs to be optimised. The scikit-learn API assumes that the model being optimised implements the scikit-learn estimator interface. This means that either the estimator needs to provide a `score` function, or `scoring` must be passed. The `scoring` parameter is a method used to assess a model accuracy. Since the `get_weights` method is integrated into the bagging regressor estimator class it is not necessary for us to be concerned with this. We are only required to ensure that the shrinkage parameter is integrated into the initialisation procedure of the API.

CHAPTER 4

REVIEW OF OPTIMISATION THEORY

4.1 INTRODUCTION

Convex optimisation is a mathematical framework that constitutes both theory and algorithms and that can be applied to finding a minimising solution to a convex objective function. The application of this field has permeated many domains including automatic control systems, estimation and signal processing, communications and networks, electronic circuit design, data analysis and modelling, statistics, and finance (Boyd and Vandenberghe, 2004). In this chapter detailed consideration will be given to aspects of convex optimisation as it relates to the objective of this thesis. In particular, emphasis is given to the primal-dual path following algorithm that is utilised to find the optimal set of weights for the proposed weighting strategy.

The reader is referred to Appendix A.1 for fundamental definitions in convex optimisation, such as that of a convex set and convex function. In this chapter primal and dual formulations of quadratic programs are discussed. In addition, consideration is given to how the path following algorithm (used in the CVXOPT library) solves these problems. Unless otherwise stated, the material in this section has been adapted from the work of Boyd and Vandenberghe (2004).

4.2 PRIMAL OPTIMISATION PROBLEM

The general form of a primal optimisation problem is

$$\begin{aligned}
 &\text{minimise } f_0(\underline{x}) \\
 &\text{subject to } f_i(\underline{x}) \leq 0, \quad i = 1, \dots, m \\
 &\quad \quad \quad h_i(\underline{x}) = 0, \quad i = 1, \dots, p.
 \end{aligned} \tag{4.1}$$

In (4.1) the goal is to find a solution $\underline{x} \in \mathbb{R}^n$ that minimises the function f_0 among all possible \underline{x} that satisfy the inequality constraints $f_i \leq 0$, $i = 0, \dots, m$ and the equality constraints $h_i = 0$, $i = 1, \dots, p$. The variable \underline{x} is called the optimisation variable and $f_0: \mathbb{R}^n \rightarrow \mathbb{R}$ is called the objective function. Moreover, $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, \dots, m$ are called the inequality constraint functions and $h_i: \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, \dots, p$ are called the equality constraint functions. Equation (4.1) is in standard form because it complies with the accepted convention of having the righthand side of the equality and inequality constraints equal to 0.

The domain \mathcal{D} of the problem in (4.1) is defined to be the set of points that satisfy

$$\mathcal{D} = \left(\bigcap_{i=0}^m \text{dom } f_i \right) \cap \left(\bigcap_{i=1}^p \text{dom } h_i \right). \quad (4.2)$$

A point $\underline{x} \in \mathcal{D}$ is feasible if it satisfies both the equality and inequality constraints in (4.1). The optimisation problem in (4.1) is said to be feasible if there exists at least one feasible point. Otherwise it is an infeasible problem. The set of all feasible points is called the feasible set.

The optimal value p^* for the objective function in (4.1) is defined as the infimum of the objective function over all feasible points. This is denoted as

$$p^* = \inf_{\underline{x} \in \mathcal{D}} \{f_0(\underline{x}) : f_i(\underline{x}) \leq 0, i = 0, \dots, m, h_i(\underline{x}) = 0, i = 1, \dots, p\}. \quad (4.3)$$

It is accepted that p^* can be $\pm\infty$. If the problem in (4.1) is infeasible then $p^* = \infty$ and if the problem is unbounded below, $p^* = -\infty$. A function is unbounded if there does not exist a constant c such that for all $\underline{x} \in \mathbb{R}^n$, $f(\underline{x}) \geq c$. Further, \underline{x}^* is defined as the optimal solution if \underline{x}^* is feasible and $f_0(\underline{x}^*) = p^*$. The set of all optimal points is called the optimal set, which is denoted as

$$X_{opt} = \{\underline{x} : f_i(\underline{x}) \leq 0, i = 0, \dots, m, h_j(\underline{x}) = 0, j = 1, \dots, p, f_0(\underline{x}) = p^*\}. \quad (4.4)$$

The optimal value in (4.3) is attained or achieved if there exists an optimal point. If X_{opt} is the empty set, then the optimal value is not attained. This occurs whenever the problem in (4.1) is unbounded below.

For $\varepsilon > 0$, a point \underline{x} is ε – suboptimal if

$$f_0(\underline{x}) < p^* + \varepsilon. \quad (4.5)$$

The set of all ε – suboptimal points is called the ε suboptimal set. A point \underline{x} is locally optimal if there is a constant $R > 0$, $R \in \mathbb{R}$ such that

$$f_0(\underline{x}) = \inf_{\underline{z} \in \mathcal{D}} \{f_0(\underline{z}) : f_i(\underline{z}) \leq 0, i = 0, \dots, m, h_j(\underline{z}) = 0, j = 1, \dots, p, \|\underline{z} - \underline{x}\|_2 \leq R\}. \quad (4.6)$$

In (4.6), \underline{x} minimises f_0 over all points in the feasible set that are a distance of no more than R away from \underline{x} . Here $\|\cdot\|_2$ is the Euclidean norm. If \underline{x} is a feasible point and $f_i(\underline{x}) = 0$, then the inequality constraint $f_i(x) \leq 0$, is active at \underline{x} . If $f_i(\underline{x}) < 0$, then the constraint is inactive. The equality constraints are active at all feasible points. Moreover, a constraint is redundant if removing it does not change the feasible set.

The general form of a primal convex optimisation problem is

$$\begin{aligned} & \text{minimise } f_0(\underline{x}) \\ & \text{subject to } f_i(\underline{x}) \leq 0, \quad i = 1, \dots, m \\ & \quad \quad \quad a_i^T(\underline{x}) = b_i, \quad i = 1, \dots, p. \end{aligned} \quad (4.7)$$

In (4.7) the functions f_0, \dots, f_m are convex. In comparison, (4.7) has three additional requirements to (4.1), namely: the objective function is convex, the inequality constraint functions are convex and the equality constraint functions $h_i(\underline{x}) = a_i^T(\underline{x}) - b_i$ are affine.

4.3 DUAL OPTIMISATION PROBLEM

To formulate the dual problem of a primal problem, we begin by constructing a Lagrangian function. The Lagrangian function $L: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$ account for the constraints in (4.1) implicitly and is written as

$$L(\underline{x}, \lambda, \nu) = f_0(\underline{x}) + \sum_{i=1}^m \lambda_i f_i(\underline{x}) + \sum_{i=1}^p \nu_i h_i(\underline{x}). \quad (4.8)$$

The domain is defined as $\text{dom } L = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$. In (4.8), $\lambda_i \in \mathbb{R}, i = 1, \dots, m$ are associated with the $f_i(\underline{x}) < 0, i = 1, \dots, m$ in (4.1) and are called Lagrangian multipliers. Similarly, $\nu_i \in \mathbb{R}, i = 1, \dots, p$ are Lagrangian multipliers associated with the equality constraints $h_i(\underline{x}) = 0$. The vectors, $\underline{\lambda}^T = (\lambda_1, \dots, \lambda_m)$ and $\underline{\nu}^T = (\nu_1, \dots, \nu_p)$ are called dual variables.

The Lagrangian dual function $g: \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$ is defined as the minimum value of the Lagrangian over \underline{x} for $\underline{\lambda} \in \mathbb{R}^m$ and $\underline{\nu} \in \mathbb{R}^p$. Thus, the dual function is the pointwise infimum of a collection of affine functions of $(\underline{\lambda}, \underline{\nu})$, which makes it concave even when (4.1) is not convex. The dual function is written as

$$g(\underline{\lambda}, \underline{\nu}) = \inf_{\underline{x} \in \mathcal{D}} L(\underline{x}, \underline{\lambda}, \underline{\nu}) = \inf_{\underline{x} \in \mathcal{D}} \left[f_0(\underline{x}) + \sum_{i=1}^m \lambda_i f_i(\underline{x}) + \sum_{i=1}^p \nu_i h_i(\underline{x}) \right]. \quad (4.9)$$

If the Lagrangian function is unbounded below $g(\underline{\lambda}, \underline{\nu}) = -\infty$. Importantly, the dual function gives a lower bound on the optimal value in (4.3). That is, for $\underline{\lambda} \geq 0$ and any $\underline{\nu}$ we have

$$g(\underline{\lambda}, \underline{\nu}) \leq p^*. \quad (4.10)$$

To confirm that (4.10) holds, suppose that $\tilde{\underline{x}}$ is a feasible point. Hence, $f_i(\tilde{\underline{x}}) \leq 0$, $h_i(\tilde{\underline{x}}) = 0$ and $\underline{\lambda} \geq 0$ which means that

$$\sum_{i=1}^m \lambda_i f_i(\tilde{\underline{x}}) + \sum_{i=1}^p v_i h_i(\tilde{\underline{x}}) \leq 0. \quad (4.11)$$

Hence,

$$L(\tilde{\underline{x}}, \underline{\lambda}, \underline{v}) = f_0(\tilde{\underline{x}}) + \sum_{i=1}^m \lambda_i f_i(\tilde{\underline{x}}) + \sum_{i=1}^p v_i h_i(\tilde{\underline{x}}) \leq f_0(\tilde{\underline{x}}) \quad (4.12)$$

and

$$g(\underline{\lambda}, \underline{v}) = \inf_{\underline{x} \in \mathcal{D}} L(\underline{x}, \underline{\lambda}, \underline{v}) \leq L(\tilde{\underline{x}}, \underline{\lambda}, \underline{v}) \leq f_0(\tilde{\underline{x}}) \quad (4.13)$$

The result in (4.13) holds because $g(\underline{\lambda}, \underline{v}) \leq f_0(\tilde{\underline{x}})$ for every feasible point $\tilde{\underline{x}}$. The bound on p^* in (4.13) is only useful when $\underline{\lambda} \geq 0$, $(\underline{\lambda}, \underline{v}) \in \mathbf{dom} g$ and $g(\underline{\lambda}, \underline{v}) < \infty$. This condition is called dual feasible.

To obtain the best lower bound in (4.13), we need to

$$\begin{aligned} & \text{maximise } g(\underline{\lambda}, \underline{v}), \\ & \text{subject to } \underline{\lambda} \geq 0. \end{aligned} \quad (4.14)$$

The problem in (4.14) is called the Lagrangian dual problem for the primal problem in (4.1). The solution to (4.14) is the dual optimal pair $(\underline{\lambda}^*, \underline{v}^*)$ and the optimal value is d^* . The inequality

$$d^* \leq p^* \quad (4.15)$$

is called weak duality. Equation (4.15) holds when both d^* and p^* are infinite. If the primal problem is unbounded below $d^* = -\infty$ which implies that the dual problem is infeasible. Moreover, if $d^* = \infty$, $p^* = \infty$. The difference between the primal and dual optimal values is referred to as the optimal duality gap. If $d^* = p^*$ then strong duality holds.

Slater's condition is a simple constraint criterion that allows us to characterise the conditions under which strong duality will hold. Slater's condition states that there is a $\underline{x} \in \mathbf{relint} \mathcal{D}$ such that

$$\begin{aligned} f_i(\underline{x}) &< 0, i = 1, \dots, m \\ A\underline{x} &= \underline{b}, \end{aligned} \quad (4.16)$$

The definition of the relative interior is given in Appendix A.1. Slater's theorem states that strong duality holds if Slater's condition holds and the problem is convex.

4.4 CERTIFICATES OF SUB OPTIMALITY AND STOPPING CRITERIA

As noted in Section 4.3, a dual feasible point $(\underline{\lambda}, \underline{\nu})$ gives a lower bound on the optimal value p^* of a primal problem. Dual feasible points therefore bound the extent to which primal feasible points are suboptimal; void of knowing the value of p^* . If the point \underline{x} is primal feasible and $(\underline{\lambda}, \underline{\nu})$ is dual feasible, then

$$f_0(\underline{x}) - p^* \leq f_0(\underline{x}) - g(\underline{\lambda}, \underline{\nu}). \quad (4.17)$$

In (4.17) the primal feasible point \underline{x} is ε -suboptimal, where $\varepsilon = f_0(\underline{x}) - g(\underline{\lambda}, \underline{\nu})$. Furthermore, $(\underline{\lambda}, \underline{\nu})$ is ε -suboptimal with respect to the dual problem. The difference

$$f_0(\underline{x}) - g(\underline{\lambda}, \underline{\nu}), \quad (4.18)$$

is called the duality gap. In the event that the duality gap is zero, the primal feasible point is primal optimal, and furthermore the dual feasible point is dual optimal. This implies that

$$p^* \in [g(\underline{\lambda}, \underline{\nu}), f_0(\underline{x})] \text{ and } d^* \in [g(\underline{\lambda}, \underline{\nu}), f_0(\underline{x})]. \quad (4.19)$$

The condition in (4.19) allows us to define an optimal stopping criterion for an optimisation algorithm. To this end, suppose that an optimisation algorithm yields a sequence of primal feasible $\underline{x}^{(k)}$ and dual feasible $(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)})$ points, for $k = 1, 2, \dots$. Let $\varepsilon_{\text{abs}} > 0$ be a specified absolute accuracy for suboptimality. If the optimisation algorithm terminates when

$$f_0(\underline{x}^{(k)}) - g(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)}) \leq \varepsilon_{\text{abs}}, \quad (4.20)$$

the solution will be ε_{abs} suboptimal. Alternatively, a specified relative accuracy $\varepsilon_{\text{rel}} > 0$ between the primal and dual problems, is guaranteed if

$$g(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)}) > 0 \text{ and} \\ \frac{f_0(\underline{x}^{(k)}) - g(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)})}{g(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)})} \leq \varepsilon_{\text{rel}} \quad (4.21)$$

or

$$f_0(\underline{x}^{(k)}) < 0 \text{ and} \\ \frac{f_0(\underline{x}^{(k)}) - g(\underline{\lambda}^{(k)}, \underline{\nu}^{(k)})}{-f_0(\underline{x}^{(k)})} \leq \varepsilon_{\text{rel}}. \quad (4.22)$$

holds.

If either of the result in (4.21) or (4.22) hold, and if $p^* \neq 0$, we have that

$$\frac{f_0(\underline{x}^{(k)}) - p^*}{|p^*|} \leq \varepsilon_{\text{rel}}. \quad (4.23)$$

From (4.23) the principal of complementary slackness stems. Complementary slackness is described in Appendix A.5.

4.5 INTERIOR POINT METHODS

Interior points methods are a class of algorithms that can be used to solve convex optimisation problems that have inequality constraints. For example, consider

$$\begin{aligned} & \text{minimise } f_0(\underline{x}) \\ & \text{subject to } f_i(\underline{x}) \leq 0, i = 1, \dots, m \\ & \quad \quad \quad A\underline{x} = \underline{b}, \end{aligned} \quad (4.24)$$

where $f_0, \dots, f_m: \mathbb{R}^n \rightarrow \mathbb{R}$ are convex functions which are twice continuously differentiable, and the matrix $A \in \mathbb{R}^{p \times n}$ has a rank of $p < n$. Assume that (4.24) is solvable - which implies that an optimal \underline{x}^* exists. Further, assume that the problem is strictly feasible and hence that there exists an $\underline{x} \in \mathcal{D}$ that satisfies the equality constraint $A\underline{x} = \underline{b}$ and the inequality constraints $f_i(\underline{x}) < 0$ for all $i = 1, \dots, m$. This implies that Slater's constraint qualification holds and so there exists dual optimal variables $\underline{\lambda}^* \in \mathbb{R}^m, \underline{v}^* \in \mathbb{R}^p$, which in conjunction with \underline{x}^* satisfy the Karush–Kuhn–Tucker (KKT) conditions. The KKT conditions are

$$\begin{aligned} & A\underline{x}^* = \underline{b} \\ & f_i(\underline{x}^*) \leq 0, i = 1, \dots, m \\ & \quad \quad \quad \underline{\lambda}^* \geq 0 \\ & \nabla f_0(\underline{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(\underline{x}^*) + A^T \underline{v}^* = 0 \\ & \quad \quad \quad \lambda_i^* f_i(\underline{x}^*) = 0, i = 1, \dots, m \end{aligned} \quad (4.25)$$

The KKT conditions are further discussed in Appendix A.3. The main idea in interior point methods is to solve either of (4.24) or (4.25) by applying Newtons method to a sequence of equality constrained problems or to a sequence of modified version of the KKT conditions. Newtons method is a descent technique and is described in Appendix A.5.

4.5.1 Logarithmic barrier function and central path

As mentioned previously, interior point methods solve an inequality constrained problem by solving a sequence of appropriate equality constrained problems to which Newton's method can readily be applied. The first step in this process is to rewrite the problem, so that the inequality constraints are implicitly represented in the objective, as

$$\text{minimise } f_0(\underline{x}) + \sum_{i=1}^m I_-(f_i(\underline{x})) \quad (4.26)$$

subject to $A\underline{x} = \underline{b}$.

In (4.26) $I_-: \mathbb{R} \rightarrow \mathbb{R}$ is the indicator function for the non-positive real numbers, which is defined as

$$I_-(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0 \end{cases} \quad (4.27)$$

The problem with (4.26) is that the objective function is not usually differentiable, which means that Newton's method cannot be applied.

4.5.2 Logarithmic Barrier

To mitigate the aforementioned issue, an approximate indicator function to I_- is used. The approximation is defined as

$$\hat{I}_-(u) = -(1/t) \log(-u). \quad (4.28)$$

The domain of (4.28) is $\text{dom } \hat{I}_- = -\mathbb{R}_{++}$ and $t > 0$ is a parameter that controls the accuracy of the approximation.

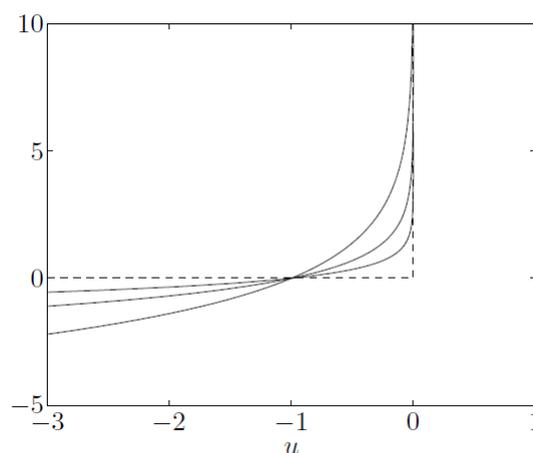


Figure 4.1: The dashed line shows the function I_- . The solid lines show the approximation \hat{I}_- for $t = 0.5, 1$ and 2 .

Source: Boyd and Vandenberghe (2004).

Both of L_- and \widehat{L}_- are convex and non-decreasing. Moreover, both are infinite for u positive. However, \widehat{L}_- is differentiable and closed. By replacing L_- with \widehat{L}_- in (4.26) we get the approximation

$$\begin{aligned} & \text{minimise } f_0(\underline{x}) + \sum_{i=1}^m -(1/t) \log(f_i(\underline{x})) \\ & \text{subject to } A\underline{x} = \underline{b}. \end{aligned} \tag{4.29}$$

The objective function in (4.29) is convex and differentiable, therefore Newton's method can be applied. The function

$$\phi(\underline{x}) = - \sum_{i=1}^m \log(-f_i(\underline{x})), \tag{4.30}$$

is called the logarithmic barrier function. The domain of the logarithmic barrier function is the set of strictly feasible points given by $\text{dom } \phi = \{\underline{x} \in \mathbb{R}^n \mid f_i(\underline{x}) < 0, i = 1, \dots, m\}$. Regardless of the size of t , $\phi(\underline{x}) \rightarrow \infty$ if $f_i(\underline{x}) \rightarrow 0$ for any $i = 1, \dots, m$. Further, as the parameter t gets larger, (4.29) becomes a better approximation of (4.26). This can be observed in Figure 4.1. However, for a large value for t , the objective $f_0(\underline{x}) + (1/t)\phi(\underline{x})$ is difficult to minimise using Newton's method since the Hessian will have significant variation near the boundary of the feasible set. To remedy this issue a sequence of problems of the same form in (4.29) should be solved but for increasingly larger values of t at each step. Each round of Newton minimisation uses the solution from the previous round as a starting value. This strategy is called the Barrier method.

4.5.3 Central Path

To demonstrate the strategy for minimising (4.30), consider the following problem

$$\begin{aligned} & \text{minimise } tf_0(\underline{x}) + \phi(\underline{x}) \\ & \text{subject to } A\underline{x} = \underline{b}, \end{aligned} \tag{4.31}$$

which has the same minimising solution as (4.29) but differs by a multiplicative factor of t . Assume that (4.31) is solvable using Newton's method. Moreover, assumed that there is a unique solution for each $t > 0$. This unique solution is denoted by $\underline{x}^*(t)$.

The central path of (4.31) is defined as the set of central points. Central points are of the form $\underline{x}^*(t)$ for which $t > 0$. The point $\underline{x}^*(t)$ is a strictly feasible point which signifies that

$$\begin{aligned} A\underline{x}^*(t) &= \underline{b} \\ f_i(\underline{x}^*(t)) &< 0, i = 1, \dots, m, \end{aligned} \tag{4.32}$$

and that there exists a $\hat{\underline{v}} \in \mathbb{R}^p$ such that

$$\begin{aligned} 0 &= t\nabla f_0(\underline{x}^*(t)) + \nabla\phi(\underline{x}^*(t)) + A^T\hat{\underline{v}} \\ &= t\nabla f_0(\underline{x}^*(t)) + \sum_{i=1}^m \frac{1}{-f_i(\underline{x}^*(t))} \nabla f_i(\underline{x}^*(t)) + A^T\hat{\underline{v}}. \end{aligned} \quad (4.33)$$

holds. The conditions in (4.32) and (4.33) are necessary and sufficient conditions for points on the central path.

4.5.4 Dual points from central path

Using (4.33) it is possible to show that every central path produces a dual feasible point. Recall that a dual feasible point is a lower bound on the optimal value p^* . To show this, consider

$$\lambda_i^*(t) = -\frac{1}{tf_i(\underline{x}^*(t))}, i = 1, \dots, m \quad (4.34)$$

$$\underline{v}^*(t) = \hat{\underline{v}}/t.$$

From (4.34) it is concluded that $\lambda_i^*(t) > 0$ since $f_i(\underline{x}^*(t)) < 0, i = 1, \dots, m$ as described in (4.32). Furthermore, using the definitions in (4.34), we can express (4.33) as

$$\nabla f_0(\underline{x}^*(t)) + \sum_{i=1}^m \lambda_i^*(t) \nabla f_i(\underline{x}^*(t)) + A^T \underline{v}^*(t) = 0. \quad (4.35)$$

Then, $\underline{x}^*(t)$ minimises the Lagrangian

$$L(\underline{x}, \underline{\lambda}, \underline{v}) = f_0(\underline{x}) + \sum_{i=1}^m \lambda_i f_i(\underline{x}) + \underline{v}^T (A\underline{x} - \underline{b}), \quad (4.36)$$

with $\underline{\lambda} = \underline{\lambda}^*(t)$ and $\underline{v} = \underline{v}^*(t)$, which implies that $\underline{\lambda}^*(t)$ and $\underline{v}^*(t)$ are a dual feasible pair. The dual function $g(\underline{\lambda}^*(t), \underline{v}^*(t))$ is thus finite, and

$$\begin{aligned} g(\underline{\lambda}^*(t), \underline{v}^*(t)) &= f_0(\underline{x}^*(t)) + \sum_{i=1}^m \lambda_i^*(t) f_i(\underline{x}^*(t)) + \underline{v}^*(t)^T (A\underline{x}^*(t) - \underline{b}) \\ &= f_0(\underline{x}^*(t)) - m/t. \end{aligned} \quad (4.37)$$

From (4.37) we note that the duality gap between the primal and dual optimal values is m/t . Hence, it holds that

$$f_0(\underline{x}^*(t)) - p^* \leq m/t. \quad (4.38)$$

The observation in (4.38) means that $\underline{x}^*(t)$ is no more than m/t -suboptimal. This result confirms the intuitive notion that $\underline{x}^*(t)$ converges to an optimal point as $t \rightarrow \infty$.

4.5.5 Unconstrained minimisation

From the result in (4.38) it was deduced that $\underline{x}^*(t)$ is m/t -suboptimal. Further, from Section 4.5 it is known that a certificate of this accuracy is provided by the dual feasible pair $\underline{\lambda}^*(t), \underline{v}^*(t)$. Hence, it is possible to solve problems of the form given in (4.24) with a guaranteed accuracy of ϵ that can be specified. One possibility to do so requires us to set $t = m/\epsilon$ and solve the equality constrained problem

$$\begin{aligned} & \text{minimise } (m/\epsilon)f_0(\underline{x}) + \phi(\underline{x}) \\ & \text{subject to } A\underline{x} = \underline{b}, \end{aligned} \tag{4.39}$$

using Newton's method. This approach is called the unconstrained minimisation method. Unfortunately, this method does not work well in general and is therefore not used in practice. It does work well for small problems with good starting points and ϵ not too small.

4.5.6 The barrier method

The barrier method, which does work well in general, is a simple adaptation to the unconstrained minimisation method. The procedure involves solving a sequence of unconstrained, or linearly constrained, minimisation problems. Each problem to be solved, subsequent to the first, uses the solution to the previous problem as a starting point. Therefore, a sequence of optimal values $\underline{x}^*(t)$ is determined with increasing values of t while $t < m/\epsilon$. When the algorithm stops, we are guaranteed to have an ϵ -suboptimal solution to the original problem. The barrier method is also called the path-following method. The algorithm is given as follows

Algorithm 4.1: The barrier method

-
1. **input:** strictly feasible point $\underline{x}, t := t^{(0)} > 0, \mu > 1$, tolerance $\epsilon > 0$
 2. **repeat:**
 3. centering step: compute $\underline{x}^*(t)$ by minimising $tf_0(\underline{x}) + \phi(\underline{x})$, s.t $A\underline{x} = \underline{b}$, starting at \underline{x}
 4. update $\underline{x} := \underline{x}^*(t)$
 5. **end:** if $m/t < \epsilon$
 6. Increase $t: t := \mu t$
-

Algorithm 4.1 shows that at each iteration, barring the first, a central point $\underline{x}^*(t)$ is computed using as a starting value the previously computed central point. In addition, for each iteration the value of t is increase by a factor $\mu > 1$. It is also possible for the algorithm to return a dual ϵ -suboptimal point

after step 3. Step 3 computes a central point. The first centering step, for which $\underline{x}^*(t^{(0)})$ is computed, is called the initial centering step. A complete loop from step 2 through 6 is called an outer iteration. The iterations required for Newton's method in the centering step are called inner iterations. For each iteration of Newton's method, we have a primal feasible point. However, we have a dual feasible point only after each centering step has been finished.

4.5.7 Accuracy of centering

Typically, when computing $\underline{x}^*(t)$ an exact centering strategy is used. The reason is twofold. Firstly, it is because the computational cost of exact centering is not significantly greater than that of inexact centering, i.e. we require only a few more Newton steps to find an accurate minimiser of $tf_0(\cdot) + \phi(\cdot)$ compared to the number of steps required to find a good minimiser of the same objective. The second reason is because by using exact centering, a pair of dual feasible points $\underline{\lambda}^*(t), \underline{v}^*(t)$ can be obtained. When using inexact centering these points are not exactly dual feasible. It is, however, possible to apply a correction term to (4.34) to alleviate this issue, provided that the point \underline{x} in step 4 is near the central path.

4.5.8 Choice of μ

The parameter μ controls the balance between the number of inner iterations and the number of outer iterations. It does so in the following manner: if μ is small (small implying near 1) then after each outer iteration t increases only by a small factor. This means that the point used in the next outer iteration provides a good starting point. Hence, fewer inner iterations will be required to obtain the next feasible point. However, when μ is small, many outer iterations will be required because each outer iteration only makes slight progress through the central path. Since each starting point is good, the iterates produced when μ is small follow the central path closely. In the case where μ is large the exact opposite occurs. By increasing t by a large factor, the current point is unlikely to be a good starting point for the following outer iteration. Therefore, the algorithm will require many more inner iterations while requiring fewer outer iterations. The fact that fewer outer iterations are needed is as a result of the duality gap being reduced by a large factor after each outer iteration. Additionally, when μ is large, the iterates are far from the central path. Typically, values within the range of 10 to 20 work well in practice.

4.5.9 Choice of $t^{(0)}$

The result of choosing $t^{(0)}$ too large will cause the first outer iteration to require many inner iterations. On the other hand, if $t^{(0)}$ is chosen too small, additional outer iterations will be necessary, and more inner iterations to solve the first centering step.

CHAPTER 5

DATA DESCRIPTION AND METHODOLOGY

5.1 INTRODUCTION

In this thesis five distinct data sets are used for the purpose of experimentation. The selection includes two practical or real-world data sets and three simulated data sets. The practical data sets include the Boston house price data set (Harrison and Rubinfeld, 1978) and the Ozone data set (Breiman and Friedman, 1985). The simulated data sets of Friedman (1991) are used and are called Friedman 1, Friedman 2 and Friedman 3. The real-world data sets are not transformed in any manner before the experiments are undertaken since it is not the purpose of this thesis to present the most accurate results. The intention of this thesis is to compare combination strategies. To ensure that the experiments are performed on the same basis, careful attention is given to the manner in which the data is split into training and testing sets. This will ensure that the base learners which are combined have an equivalent generalisation performance. Furthermore, for consistency across all experiments a random seed is set. This will ensure that the results, that are presented in Chapter 6, are entirely reproducible.

In this chapter a description of the data sets is given. In addition, the methodology used in the experiments will be outlined. Lastly, the specifications for determining the optimal settings of the complexity parameters will also be provided.

5.2 DATA DESCRIPTION

In this section a description of the five data sets is given. The description will include an outline of the strategy used to generate the simulated data sets. The data sets and experimental design used in the thesis was chosen based on a paper by Breiman (1994) where the bagging ensemble method is introduced.

5.2.1 The Ozone data set

The Ozone data set was used to study the relationship between atmospheric ozone concentration and meteorology in the Los Angeles Basin in 1976. The original source dataset is given in Breiman and Friedman (1985). In its raw state, the data set contains a total of 330 observations with 9 predictors. The response variable is O₃ that is a measurement of the maximal daily ozone concentration, in units of parts per million (ppm), at Sandbug AFB. A description of the predictors is given in Table 5.1.

Table 5.1: Description of the predictors included in the Ozone data set.

Predictor	Description
Vh	A numeric vector
Wind	Wind speed
Humidity	A numeric vector
Temp	Temperature
Ibh	Inversion base height
Dpg	Daggett pressure gradient
Ibt	A numeric vector
Vis	Visibility
Doy	Day of the year

The version of the Ozone data set used in this thesis has been source from the UCI machine learning repository (Dua and Graff, 2019).

5.2.2 The Boston house-prices data set

The Boston house-prices data set was originally created by Harrison and Rubinfeld (1978). The authors used this data set to study the impact of different factors on housing prices in Boston, Massachusetts. Each entry in the data set describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970.

The Boston house-prices data set used in this thesis has been sourced from the scikit-learn Python library (Pedregosa *et al.*, 2011). The reason for doing so is simply because it facilitates easy access to the data when programming in Python. The authors of the library interface state that the version of the data set used in the API has been sourced from the StatLib library which is maintained at Carnegie Mellon University. In the raw state, the data set contains a total of 506 observations with each observation being comprised of 13 predictors. The predictors are a combination of both numeric and categorical values. The response is MEDV, which represents the median value of owner-occupied homes measured in \$1000s. The response ranges in value from 5 – 50. The predictors, along with a description, are listed in Table 5.2.

Table 5.2: Description of the predictors included in the Boston housing prices data set.

Predictor	Description
Crim	Per capita crime rate by town
Zn	Proportion of residential land zoned for lots over 25,000 sq.ft
Indus	Proportion of non-retail business acres per town
Chas	Charles River dummy variable: 1 if tract bounds river and 0 otherwise
Nox	Nitrogen oxides concentration (parts per 10 million)
Rm	Average number of rooms per dwelling
Age	Proportion of owner-occupied units built prior to 1940
Dis	Weighted mean of distances to five Boston employment centres
Rad	Index of accessibility to radial highways
Tax	Full-value property-tax rate per \$10,000
PtRatio	Pupil-teacher ratio by town
Black	$1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
Lstat	Lower status of the population (percent) (indicator of poverty)

5.2.3 Friedman 1

The three simulated data sets were used by Friedman (1991). In the discussion that follow a description of how these data sets have been generated is given. The scikit-learn API (Pedregosa *et al.*, 2011) provides an implementation for all three of the Friedman data sets. These functions are utilised to produce the results in Chapter 6 and therefore a description of their usage is also included in this subsection.

In the simulated data set Friedman 1 there are 10 predictor variables which are generated using the uniform distribution on the interval $[0,1]$. The response is constructed using 5 of the total 10 predictor variables according to the following equation

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon, \quad (5.1)$$

where $\epsilon \sim N(0,1)$.

The scikit-learn interface for generating observations according to (5.1) is implemented in the `make_friedman1` function and has the following parameters:

n_samples: (default=100)

The number of samples.

n_features: (default=10)

The number of features. Should be at least 5.

noise: (default=0.0)

The standard deviation of the gaussian noise applied to the output.

random_state: (default=None)

Determines random number generation for dataset noise. Pass an int for reproducible output across multiple function calls.

For the purpose of experimentation, the only parameters that are considered for manipulation are the number of samples and the random state. The manner in which they are set will be described in Section 5.3.

5.2.4 Friedman 2 and Friedman 3

The data sets in Friedman 2 and Friedman 3 simulate the impedance and phase shift in altering current circuits. Each data set has 4 predictors with the response for Friedman 2 being constructed as follows

$$y = \left[\left(x_1^2 + \left(x_2 x_3 - \frac{1}{x_2 x_4} \right) \right)^2 \right]^{1/2} + \epsilon_2. \quad (5.2)$$

The response for Friedman 3 is constructed as

$$y = \tan^{-1} \left(\frac{x_2 x_3 - \frac{1}{x_2 x_4}}{x_1} \right) + \epsilon_3. \quad (5.3)$$

The variables x_1, x_2, x_3, x_4 are uniformly distributed over the following ranges

$$\begin{aligned} 0 &\leq x_1 \leq 100 \\ 20 &\leq \left(\frac{x_2}{\pi} \right) \leq 280 \\ 0 &\leq x_3 \leq 1 \\ 1 &\leq x_4 \leq 11. \end{aligned} \quad (5.4)$$

Further, $\epsilon_2 \sim N(0, \sigma_2^2)$ and $\epsilon_3 \sim N(0, \sigma_3^2)$. The variance parameters σ_2^2, σ_3^2 are selected so that the response has a 3:1 signal to noise ratio. Refer to Section 2.2.9 for a description on generating a response with a desired signal to noise ratio.

In the scikit-learn API the implementation for the Friedman 2 and Friedman 3 data sets are available through the functions `make_friedman2` and `make_friedman3`, respectively. The parameters for these functions are:

n_samples: (default=100)

The number of samples.

noise: (default=0.0)

The standard deviation of the gaussian noise applied to the output.

random_state: (default=None)

Determines random number generation for dataset noise. Pass an int for reproducible output across multiple function calls.

For the purposes of experimentation, as is in the case of the Friedman 1 data set, the only parameters considered for manipulation are the number of samples and the random state. The manner in which they are set will be described in the experimental methodology in Section 5.3.

5.2.5 Note on implementation

As previously stated, the simulated data sets used in this thesis have been created using the scikit-learn Python programming API. However, an inspection of the source code was carried out and it was found that in the case of the Friedman 2 and Friedman 3 data sets the desired signal to noise ratio of 3:1, as used in Breiman (1994), was not adhered to. Therefore, the necessary corrections were made in accordance with the details outline in Subsection 2.2.7.

5.3 METHODOLOGY

In this section the experimental methodology is outlined. In the experiments it quickly became apparent that it is challenging to obtain weights that provide a good generalisation ability. Hence, different strategies for fitting the weighted ensemble were considered. In the end, the strategy that achieved the lowest five-fold cross validation error was considered. This happened to be methodology 3.

The experimental design used in this thesis is adapted from the methodology used by Breiman (1994) where the bagging ensemble model was introduced. The bagging ensemble was compared to a decision tree on several data sets in both classification and regression problems. In the regression problems, the bagging ensemble comprised 25 regression trees as base learners. Each base learner, including the regression tree to which the bagged ensemble is compared, was trained using 10-fold CV. To assess goodness of fit, Breiman (1994) used the MSE. However, the results presented in the paper are an arithmetic mean of the MSE over 100 experiments.

The design used in this thesis deviates from the methodology used by Breiman (1994) in the following ways: Consideration is given to a more comprehensive range for the number of base learners in each ensemble method. In addition, we choose to use an additional ensemble method for comparative purposes - random forest regressor. Hence, results will be given for a regression tree, a bagged ensemble, a random forest ensemble and the proposed weighed ensemble for two cases. In the first case, the base learners are generated according to the same strategy used in bagging. In the second case, the base learners are generated according to the same strategy use in random forest. Furthermore, we do not use 10-fold CV to train each model. Instead, we opt for fivefold CV. This decision was made as a result of the computational burden associated with training the weighted ensemble technique. Lastly, we consider three metrics for assessing the fit of the ensemble methods considered. Namely, the MSE, MAE and R^2 score. The results in Chapter 6 are the arithmetic mean score over 10 experiments as opposed to 100 experiments.

5.3.1 Methodology 1

In methodology 1, all data sets are split into a training, testing and validation set. For each real-world data set, the data is first randomly divided into a training and testing set. Accordingly, for each simulated data set, the first step is to generate training and testing observations. We choose to generate 200 training and 1000 testing observations for each of Friedman 1, Friedman 2 and Friedman 3. This is in agreement with Breiman (1994).

Subsequent to the creation of the training and testing sets, each of the training sets is further divided into a different training set and a validation set. The size of the validation set was chosen to be 30 percent of the number of observations in the original training set. In this methodology the validation set is used to obtain the optimal weights in the weighed ensemble technique. Importantly, this implies that for procedures which do not require a separate validation set, i.e. regression tree, bagging ensemble and random forest ensemble, the validation set should be joined back onto the original training set before the models are fit. Therefore, the size of the training and testing sets used to fit the regression tree, bagging ensemble and random forest are: 481 and 25 observations in the Boston house prices data set, 315 and 15 observations in the Ozone data set and 200 and 1000 observations in each of the simulated data sets, respectively. The size of the training, testing and validations sets used to fit the weighted ensemble are: 337, 144 and 25 observations in the Boston house prices data set, 220, 95 and 15 observations in the Ozone data set and 140, 60 and 1000 observations in each of the simulated data sets, respectively.

The aforementioned details mean that the base learners used in the weighed ensemble are trained on fewer observations than the procedures to which it is compared. This was deemed to be appropriate for presenting a fair comparison of the performance of the different methods. Notwithstanding this, it is not desirable to optimise the weights over various different folds as would be the case in K-fold CV. In this scenario, the optimal weights would either have to be set to specific values, in which case K-fold CV can be used to find the best setting from among the available alternatives. On the other hand, a set of optimal weights can be found for each fold and the final set of optimal weights could be determined as an average of the optimal weights over all the folds. However, this result would not necessarily meet the constraints on the weights as is described in Section 3.5.

Breiman (1994) considered 25 bootstrap replications of the training set for the purpose of fitting the base learners in the bagging procedure. We deviate from this, and consider four different cases, namely: 25, 50, 100 and 250 bootstrap replications. After generating the bootstrap samples, a regression tree is grown on each bootstrap sample using fivefold CV to optimise the complexity parameters.

Subsequent to fitting the base learners, a prediction is obtained for each observation in the validation set (for the weighted ensembles only). The predictions are then stacked, column-wise, to form a matrix Z of size $N_{\text{validation}} \times B$ where $N_{\text{validation}}$ is the number of observations in the validation set and B is the number of base learners. The matrix Z along with the corresponding response vector for the observations in the validation set are passed into the function `get_weights(self, Z, y)`. The result returned by this function is a set of B weights that can be used to average the predictions of the B base learners in the weighted ensemble. The `get_weights(self, Z, y)` is described in Subsection 3.6.3.

To test the models, a prediction is obtained for each observation in the testing set. Note that due to the way the data is split, each model receives as input exactly the same testing observations. However, as mentioned previously, the weighted ensemble is trained on a different training set to that used in the regression tree and the equally weighted ensembles. To be clear, the combination of the training and validation sets used to fit the two weighted ensembles is the same as the training set used to fit the equally weighted ensembles. To assess each model's predictive accuracy, the MSE, MAE and R^2 are used.

The entire process, from splitting the original data sets to training and testing the models, was repeated a total of 10 times. The arithmetic mean MSE, MAE and R^2 over 10 experiments are calculated for each model. For each experiment a different value for the random seed is set. The random seed specifies the starting point for the pseudo random number generating process used in the scikit learn API. The value of the random seed in each experiment is set to the index of the experiment starting at 0. So, the seed was set to 0 for the first experiment, 1 for the second experiment and so on. The purpose for setting the random seed is to facilitate reproducibility of the results. The seed is set to a different value for each experiment to allow for different outcomes, i.e. if the seed were unchanged, the results for each of the 10 experiments would be the same. However, since the value of the seed is known for each experiment, the results are reproducible.

Unfortunately, this strategy did not yield a better outcome in comparison to methodology 3. The most possible reason is the difference in the number of observations used to train the base learners of the weighted ensemble compared to the number of observations used to train the base learners in the equally weighted ensemble methods. Hence, the benefit obtained from training on more observations completely outweighs any benefit obtained from optimal weighting. Different proportions for the size of the validation set were investigated. In the case of a smaller validation set (smaller than 30 percent), the performance remained poor compared to the other procedures. We can provide two possible reasons for this. First, the weights using a small validation set cannot accurately account for the generalisation ability of the individual models. Hence, the weighted ensemble may falsely place a high weight on a model that has a poor generalisation ability. Second,

the benefit from training on more observations is just too great to be overcome through optimal weighting. Hence, no reasonably sized validation set can be used.

5.3.2 Methodology 2

Methodology 2 and methodology 3 differ from methodology 1 in terms of the training set used to determine the optimal weights.

In methodology 2 the proposal is to use the OOB observations of each of B bootstrap samples (used to construct the ensemble) to create a secondary data set. A prediction is obtained for each of the B trees on all observations in the secondary data set. Subsequently, the predictions are stacked column wise to form the matrix Z . The weights are solved using the response vector corresponding to the observations in the OOB testing set. This proposal is outlined in Figure 5.1.

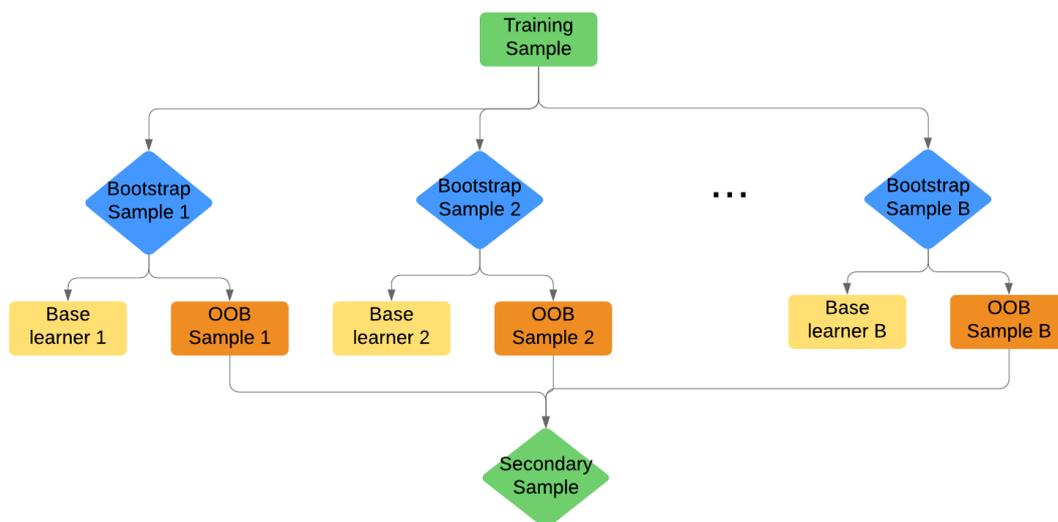


Figure 5.1: Methodology 2 involves using the OOB observations from each bootstrap sample to obtain the optimal weight for the weighted ensemble.

In this proposal, the training data set is the same as the training data set used to fit the equally weighted ensemble methods. However, the size of the data set used to obtain the optimal weights is variable, depending on the number of OOB observation.

5.3.3 Methodology 3

In methodology 3, the proposal involves using the same training data (that is used to optimise the complexity parameters of the base learners) for finding the optimal weights. The obvious concern with this method is that it will lead to overfitting. However, recall from the discussion on the bootstrap in Section 2.6 that on average only 63.2 percent of the observations in the original training set are included in a bootstrap resampled data set. Therefore, on average the training set is likely to be sufficiently different from the bootstrap samples used to construct the base learners. Empirically, this method demonstrated superiority over method 1 and method 2 in all experiments.

To determine the optimal weights, it is necessary to obtain a prediction for every observation in the original training data set and for every base learner in the ensemble. The resulting predictions are stacked column wise to form the matrix Z which is subsequently passed into the `get_weights(self, Z, y)` function to obtain the weights.

In this methodology, the training set used to fit the weighted ensemble is the same as that used to fit the equally weighted ensembles.

5.4 GRID SEARCH CV

The `GridSearchCV` interface, which is implemented in the scikit-learn API, provides a method for exhaustively searching over a specified grid of parameters using K-fold CV. The parameters of this function include

estimator: estimator object.

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.

param_grid: dict or list of dictionaries

Dictionary with parameters names (string) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

cv: int, cross-validation generator or an iterable, optional

Determines the cross-validation splitting strategy.

refit: boolean, string, or callable, default=True

Refit an estimator using the best found parameters on the whole dataset.

Internally, the `GridSearchCV` procedure uses the values specified in the parameter grid to find an optimal setting for the complexity parameters of the estimator. The optimal combination of parameters is the one for which the cross validated error is a minimum. Note that this may not be the globally optimal choice since it may be computationally impossible to test all possible settings of the complexity parameters of the estimator. Instead, the `GridSearchCV` procedure returns the optimal combination from among the choices specified in the parameter grid. In addition to finding the optimal parameters, the `GridSearchCV` procedure also refits the estimator using all the training data with the optimal set of complexity parameters. To produce the results of Chapter 6 the

estimators considered are: a regression tree, a random forest regressor, a bagging regressor (with regression trees as base learners) and two weighted ensemble techniques. The weighted ensemble techniques are called weighted 1 and weighted 2. In weighted 1 the base learners are grown in the same manner as those that comprise a bagging regressor, i.e. all features are considered when splitting a node. In weighted 2, the base learners are grown in the same manner as those that comprise random forest regressor, i.e. a random subset of the features are chosen when determining a node to split on. In the results of Chapter 6 the random subset was chosen to be \sqrt{p} .

To grow each base learner in the ensemble methods considered, the MSE criterion with the best splitting is used. The tree depth was optimised over the range spanning 2 to 25 splits. The range of possible values for the regularisation parameter was chosen from 0.01 to 1 with increments of 0.01.

5.5 CONQP FUNCTION

The `conqp` function is used to find the optimal weights for the weighted ensemble technique. The default parameters, as specified by Anderson *et al.*, (2004), were used for the purpose of optimising the weights. In this respect, the absolute accuracy was set to 1×10^{-7} and the relative accuracy was set to 1×10^{-6} .

CHAPTER 6

RESULTS

6.1 INTRODUCTION

In this chapter we present the empirical results for both the simulated and real-world experiments. In this regard, we consider the arithmetic mean MSE, MAE and R^2 score on five data sets over 10 experiments using fivefold CV. Methodology 3, as described in Subsection 5.3.3, was used in producing the results. The results confirm that the weighted ensemble techniques consistently outperform a single regression tree. This is an expected outcome. Of more interest is the observation that the weighted ensemble techniques are highly competitive against both bagging and random forest. In some instances, they outperform these techniques. As stated previously, Xu *et al.*, (1992), Ho *et al.*, (1994) and Kittler *et al.*, (1998) noted in empirical studies that weighting does not give superior results in all circumstances. These authors state that the reason is due to the nature of the data - which is often noisy in practical applications. Hence, overfitting can easily occur in large weighted ensembles. The results of this thesis indicate that the use of regularisation in determining the optimal weights mitigates this issue. The results also indicate that the proposed technique provides flexibility in controlling the bias-variance trade-off. It is argued in this chapter that bagging and random forest are special cases of the weighted ensemble techniques considered. That is, bagging and random forest are highly regularised weighted ensembles, not merely equally weighted ensembles. This deduction relies on the imposition of two constraints to the weights, namely: a positivity constraint and a normalisation constraint.

The Python source code for performing the experiments is given in Appendix B. In this regard, the configuration class manages all parameter settings for the experiments. The main class is used to run the experiments which are handled by the experiment class. The weighting strategies are implemented in the weighted bagging class. The data sets considered in the experiments are handled by the data class.

6.1.1 General remarks on the results

A lower test MSE and MAE indicate better generalisation ability. However, a larger R^2 score is better.

In the result tables of Section 6.2 and Section 6.3, the column Weighted 1 denotes the results for a weighted ensemble that is composed of base learners which are grown without randomised feature selection. The column Weighted 2 indicates the results for a weighted ensemble that is composed of base learners that are grown using a random subset of the features.

Each base learner in random forest and weighted 2 is grown using a random subset $m = \sqrt{p}$ of the features, where p is the number of features in the training set.

In each result table, the number of base learners (indicated by B) is only relevant to the results of the ensemble methods, namely: bagging, random forest and the weighted ensemble techniques and not the regression tree.

In the result tables, the red and green cells indicate, respectively, the worst and best performing ensemble method for a specific number of base learners, B. This colouring does not consider the performance of the regression tree because it remains the worst performing technique across all the data sets considered.

Bagging and weighted 1 are comparable and random forest and weighted 2 are comparable. That is, the base learners that compose the bagging ensemble and the weighted 1 ensemble are the same and the base learners that compose random forest ensemble and the weighted 2 ensemble are the same. By this we imply that the structure (split variables and split points) of the corresponding base learners in the comparable techniques are the same. This holds because the base learners are trained using the same random seed and hence the same bootstrap samples and, in the case of random forest and weighting 2, the same subset of the features. Hence, to determine the effect of optimal weighting in isolation, the performance of bagging and weighted 1 should be compared and the performance of random forest and weighted 2 should be compared. The relative performance between these groups indicates the effect of the randomised feature selection procedure in conjunction with the effect of optimal weighting.

The results for the regression tree remain stable because of the random seed which is set for each experiment. The improvement in each column (from top to bottom) for the ensemble techniques is a result of increasing the number of base learners.

6.2 SIMULATION RESULTS

In this section the results for the simulation study are given. The results are presented in tabular form, indicating the arithmetic mean test MSE, MAE and R^2 score over 10 experiments using fivefold CV to optimise the complexity parameters of the base learners. In addition, for each data set a summary of the results over the 10 experiments is given in the form of a box plot.

6.2.1 Friedman 1 results

Table 6.1: Arithmetic mean MSE over 10 experiments on the Friedman 1 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	11.563	5.515	6.455	5.447	6.333
50	11.563	5.305	6.264	5.264	5.852
100	11.563	5.230	6.154	5.170	5.673
250	11.563	5.123	6.096	5.085	5.591

Table 6.2: Arithmetic mean R^2 over 10 experiments on the Friedman 1 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.514	0.768	0.728	0.771	0.733
50	0.514	0.777	0.736	0.779	0.754
100	0.514	0.780	0.741	0.783	0.761
250	0.514	0.784	0.743	0.786	0.765

Table 6.3: Arithmetic mean MAE over 10 experiments on the Friedman 1 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	2.682	1.865	2.044	1.852	2.016
50	2.682	1.829	2.012	1.820	1.937
100	2.682	1.811	1.993	1.802	1.903
250	2.682	1.794	1.982	1.787	1.888

6.2.2 Friedman 2 results

Table 6.4: Arithmetic mean MSE over 10 experiments on the Friedman 2 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	36484.492	21211.052	24478.903	21243.913	23374.000
50	36484.492	20539.210	23598.991	20622.722	22262.546
100	36484.492	20534.077	23322.546	20565.302	21862.010
250	36484.492	20337.131	22999.163	20373.776	21377.002

Table 6.5: Arithmetic mean R^2 over 10 experiments on the Friedman 2 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.766	0.864	0.843	0.864	0.850
50	0.766	0.868	0.848	0.868	0.857
100	0.766	0.868	0.850	0.868	0.860
250	0.766	0.869	0.852	0.869	0.863

Table 6.6: The arithmetic mean MAE over 10 experiments on the Friedman 2 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	151.649	115.864	123.506	116.002	121.018
50	151.649	113.890	121.233	114.175	118.353
100	151.649	113.912	120.288	114.094	117.362
250	151.649	113.324	119.662	113.434	116.063

6.2.3 Friedman 3 results

Table 6.7: Arithmetic mean MSE over 10 experiments on the Friedman 3 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.0456	0.0259	0.0274	0.0263	0.0278
50	0.0456	0.0256	0.0272	0.0258	0.0270
100	0.0456	0.0253	0.0272	0.0256	0.0268
250	0.0456	0.0253	0.0266	0.0258	0.0268

Table 6.8: Arithmetic mean R^2 over 10 experiments on the Friedman 3 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.590	0.765	0.751	0.762	0.748
50	0.590	0.768	0.753	0.767	0.756
100	0.590	0.771	0.754	0.768	0.757
250	0.590	0.770	0.759	0.766	0.757

Table 6.9: Arithmetic mean MAE over 10 experiments on the Friedman 3 data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.150	0.120	0.122	0.121	0.123
50	0.150	0.119	0.121	0.120	0.123
100	0.150	0.118	0.121	0.119	0.121
250	0.150	0.118	0.120	0.119	0.121

6.2.4 Remarks

On Friedman 2 and Friedman 3, bagging achieves the optimal performance. On Friedman 1 the weighted 1 ensemble technique achieves the best performance. However, the performance of weighted 1 on Friedman 1 is only marginally better than bagging. Likewise, the performance of bagging on Friedman 2 and Friedman 3 is only marginally better than weighted 1. This suggests that

equal or near equal weighting is optimal. It also suggests that the weights of weighted 1 are near to being equal. Weighted 1 achieved a maximum improvement over bagging of 1.23 percent for the MSE, 0.52 percent for the R^2 score, and 0.70 percent for the MAE across the simulated data sets. In the worst case, weighted 1 saw a decrease in the generalisation performance of 1.97 percent for the MSE, 0.39 percent for the R^2 score, and 0.85 percent for the MAE. Likewise, weighted 2 achieved a maximum improvement over random forest of 8.28 percent for the MSE, 0.39 percent for the R^2 score, and 4.74 percent for the MAE. At worst weighted 2 saw a decrease in the generalisation performance, relative to random forest, of 1.46 percent for the MSE, 2.96 percent for the R^2 score and 1.65 percent for the MAE.

The effect of the shrinkage parameter on the size of the weights is discussed with reference to the bias-variance decomposition in Section 5.3. In brief, it is observed that as the size of the shrinkage parameter increases, the size of the weights converges to $\frac{1}{B}$, which is the point of equal weighting. Therefore, theoretically the weighting strategies should achieve the same result to bagging – if it is assumed that equal weighting is optimal. However, in the results we see that the performance of weighting 1 is slightly worse than that achieved by bagging. Thus, the optimal solution in weighting 1 is not the same as that obtained by equal weighting. It is postulated that this outcome is a consequence of the training strategy. The models which are given more weight are likely to be the ones whose bootstrap training samples are most similar to the original training sample. Recall that, on average 63.2 percent of the observations in the original training sample are included in the bootstrap samples. This implies that some bootstrap samples will be more similar to the original training sample than others. Hence, models trained on these bootstrap samples will be better at predicting the observations in the original training sample and will therefore receive a larger weight. Due to the positivity and normalisation constraint imposed on the weights, this implies that the weights of inferior models will be decreased but will be no less than zero.

In Section 6.4, when considering the bias-variance decomposition of the MSE we concur that equal weighting corresponds to the largest reduction in variance for the weighting strategies. Thus, in the results, where bagging achieves the lowest MSE, it is because the benefit of reducing the variance of the fitted ensemble is greater for improving generalisation than reducing the bias of the fitted ensemble.

The randomised feature selection procedure used in random forest and the weighted 2 technique has not improved performance.

There is a noticeable improvement in the performance of weighted 2 over random forest. On Friedman 1, it is possible that this is a consequence of the fact that only 5 out of 10 of the features are relevant to the response. Hence, it is possible that the randomisation procedure selects uninformative features and therefore produces base learners which may have an inadequate

generalisation performance. In this case, the optimal weighting ensemble is likely giving these base learners a low weight (close to zero). However, this does not fully explain the improvement in performance of weighted 2 over random forest on Friedman 2 and Friedman 3.

Increasing the size of the weighted ensemble techniques does not adversely affect the generalisation performance. Hence, through the application of regularisation we have mitigated overfitting in larger ensembles. The idea is as follows: in large ensembles with low regularisation a large weight is assigned to only a few models. Thus, the benefit of averaging is lost and those models with a large weight may not necessarily generalise well. The discussion in Section 6.4 elaborates on this observation.

6.3 REAL WORLD DATA SET RESULTS

6.3.1 The Boston house prices data set

Table 6.10: Arithmetic mean MSE over 10 experiments on the Boston house prices data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	21.665	14.636	14.827	14.397	13.956
50	21.665	14.366	14.028	14.347	12.950
100	21.665	14.459	13.364	14.473	12.625
250	21.665	14.295	12.969	13.216	11.795

Table 6.11: Arithmetic mean R^2 over 10 experiments on the Boston house prices data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.703	0.814	0.823	0.816	0.830
50	0.703	0.822	0.829	0.820	0.842
100	0.703	0.822	0.837	0.819	0.844
250	0.703	0.822	0.844	0.834	0.854

Table 6.12: Arithmetic mean MAE over 10 experiments on the Boston house prices data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	2.895	2.509	2.476	2.469	2.375
50	2.895	2.467	2.364	2.465	2.273
100	2.895	2.472	2.327	2.461	2.283
250	2.895	2.465	2.282	2.392	2.262

6.3.2 The Ozone data set

Table 6.13: Arithmetic mean MSE over 10 experiments on the Ozone data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	24.342	14.184	13.489	14.044	12.796
50	24.342	13.543	13.111	13.752	12.522
100	24.342	13.206	12.616	13.180	12.411
250	24.342	13.160	12.731	13.242	12.025

Table 6.14: Arithmetic mean R^2 over 10 experiments on the Ozone data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	0.551	0.729	0.744	0.731	0.756
50	0.551	0.738	0.749	0.733	0.759
100	0.551	0.748	0.758	0.748	0.762
250	0.551	0.750	0.756	0.749	0.769

Table 6.15: Arithmetic mean MAE over 10 experiments on the Ozone data set.

B	Tree	Bagging	Forest	Weighted 1	Weighted 2
25	3.690	2.917	2.842	2.883	2.793
50	3.690	2.822	2.823	2.841	2.782
100	3.690	2.779	2.772	2.788	2.767
250	3.690	2.780	2.765	2.786	2.705

6.3.3 Remarks

In the real-world data set experiments, it is clear that the randomised feature selection procedure, used in random forest and weighted 2, has improved performance over bagging and weighted 1. In addition, weighted 2 has resulted in an improvement in performance over random forest. Weighted 2 is certainly the best performing technique on the real-world data sets.

Consider the relative performance of random forest and weighted 2 on the Boston house prices data set. In terms of the MSE, weighted 2 has achieved at best a 9 percent improvement over random forest. The aforementioned outcome occurs for the largest ensemble tested, namely 250 base learners. In terms of the R^2 value, at best, weighted 2 achieved a 1.54 percent improvement over random forest for 50 base learners. In terms of the MAE, the largest margin is 4.07 percent which

occurred for 25 base learners. However, in general, as the size of the ensemble increases the overall generalisation performance does not significantly decrease and in some cases it improves.

On the Ozone data set, the largest improvement in the MSE achieved by weighted 2 over random forest is 5.5 percent which occurred for 250 base learners. In terms of the R^2 score and the MAE the equivalent values are 1.72 percent for 250 base learners, and 2.17 percent for 250 base learners.

The comparative performance of bagging and weighted 1 is varied. However, in some cases, the improvement of weighted 1 over bagging is quite good. For example, weighted 1 has achieved a 7.5 percent improvement in MSE over that of bagging for 250 base learners on the Boston house prices data set. At worst, the MSE for weighted 1 on the Boston house prices data set was only 0.09 percent behind the that of bagging – this occurred for 100 base learners.

The eye-catching observation in the results of the real-world experiments is the outright dominance of weighted 2. Neither of weighting 1 nor weighting 2 did as well in comparison to the alternative's strategies in the simulation experiments. What comes to mind as a possible explanation for this result is the difference in the size of the data sets used for training in the real-world and simulation experiments. Recall that in the simulation experiments, the training data sets included 200 observations. In the real-world data set experiments, we have 481 observations for the Boston house prices data set and 315 observations for the Ozone data set. It is postulated that the improvement in relative performance seen in the real-world data experiments is due to this additional data; which is beneficial for estimating the shrinkage parameter. As stated in the Section 2.2, as model complexity increases additional data is required to avoid overfitting. In addition to this, the absolute average number of observations not included in the bootstrap samples (which is 36.8 percent of the observations in the training data set) is greater in the real-world experiments than in the simulation experiments. Therefore, the weights will be determined (on average) using a data set that contains more unseen observations (relative to that of the simulation experiments in absolute terms) and therefore there is more certainty that a larger weight corresponds to a model that generalise well.

In general, for both the simulated and real-world experiments, the optimal solutions correspond to a large value for the shrinkage parameter, i.e. a large amount of regularisation. This reiterates the point that variance reduction is an important aspect of the success of the ensemble methods considered in this thesis.

6.4 BIAS VARIANCE DECOMPOSITION

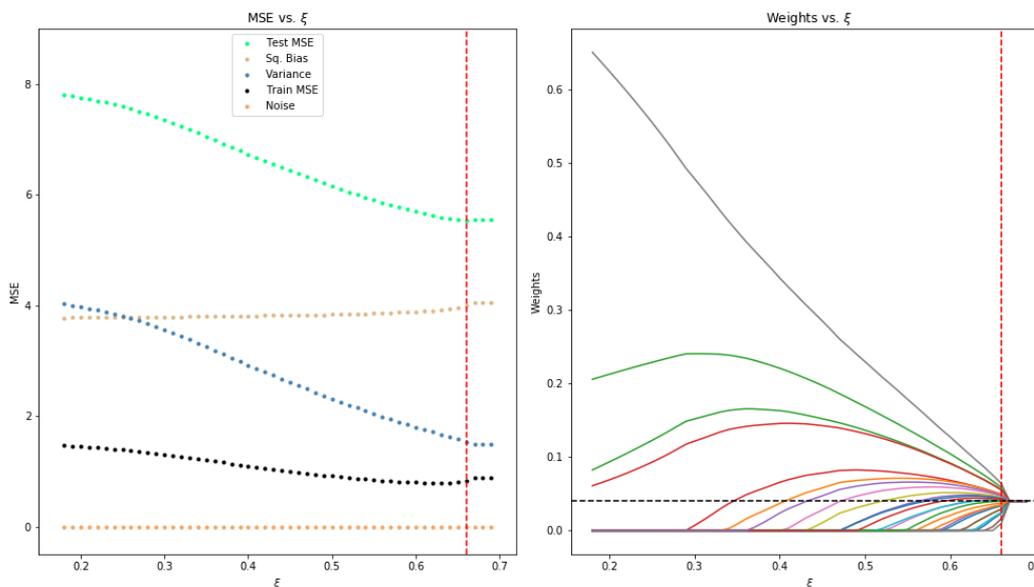


Figure 6.1: Bias-variance decomposition and weight profiles for 25 base learners on Friedman 1.

In Figure 6.1 we consider the bias-variance decomposition for the weighted 1 ensemble technique using 25 base learners on the Friedman 1 data set. Figure 6.1 demonstrates the manner in which regularisation trades a slight increase in bias for a large reduction in variance. In both figures the vertical red line indicates the size of the shrinkage parameter corresponding to the minimum test MSE.

The result on the left of Figure 6.1 was created using 500 simulation experiments. In each experiment, a training sample and testing sample is generated. The predictions on the testing sets are used to calculate the MSE, squared bias and variance. Note, that to create this figure, the tree size was not optimally trained in each experiment. Instead, the maximum tree depth was set to 11 splits - which is the average tree depth chosen by fivefold CV over 10 experiments.

The result on the right of Figure 6.1 indicates the effect of the shrinkage parameter on the size of the weights. The horizontal black line corresponds to equal weighting, or the point $\frac{1}{B}$. Since 25 base learners were used to obtain Figure 6.1, this line is at the point $1/25 = 0.04$. It is interesting to see that as the size of the regularisation parameter increases, the weights converge to being equal. Recall that the weights assigned to each base learner in the bagging procedure is $\frac{1}{B}$. Thus, not only is bagging a special case of a weighted ensemble – bagging is an equally weighted ensemble, it is a highly regularised weighted ensemble. This conclusion is given the specification of our proposal, i.e. ℓ_2 regularised weights that are constrained to be positive and sum to one.

It is even more interesting to note that a sparse solution corresponds to less regularisation. This is opposite to the behaviour observed in ridge regression which is seen in Figure 2.3. Typically, in ridge regression as the size of the regularisation parameter is increased, the ridge coefficients converge to zero. In addition, in ridge regression the coefficients are not sparse, i.e. equal to 0. However, in the weighted 1 ensemble, as the size of shrinkage parameter decreases in size, some of weights are approximately 0, i.e. at $\xi = 0.2$ it appears as if only 3 out of 25 models are active in the weighted 1 ensemble.

In addition, notice in Figure 6.1 (on the right) that the constraint on the weights have been adhered to. That is, each weight is positive, and it was confirmed that the weights sum to one for every setting of the regularisation parameter ξ .

Consider again the result on the left of Figure 6.1. This result demonstrates the flexibility of the weighted ensemble in controlling the trade-off between bias and variance. As the size of the regularisation parameter increases, the variance decreases and the bias increases. This is expected behaviour when using regularisation. In Figure 6.1 the optimal size of the shrinkage parameter corresponds to near equal weighting of the base learners. This implies that reducing variance is the main contributor to lowering the MSE. The weights corresponding to less regularisation, have a large variance and low bias. This indicates that the weighted ensemble is overfitting to the data. The overfitting is caused by the weights placing a high degree of importance on only a few models. Thus, in this case we have limited the extent to which we can benefit from averaging. By increasing the size of the regularisation parameter, we decrease the importance of anyone model in the ensemble. Thus, increasing the benefit of averaging through a reduction in variance. The optimal shrinkage parameter does not correspond to the smallest variance (which is at the point of equal weighting). Indeed, the optimal shrinkage parameter corresponds to a point that has slightly lower bias and higher variance than that of equal weighting.

CHAPTER 7

CONCLUSION

7.1 FINAL REMARKS

This thesis considers strategies for combining multiple base learners into a single ensemble model. To provide context, a review of literature from SLT is given. In this regard, we consider formalities of learning from data and also how to objectively assess the effectiveness of the resulting model. In addition to this, we consider various modelling techniques that are critical to this thesis such as regression trees.

The combination techniques we focus on are averaging and weighted averaging for combining base learners which are trained in parallel. We focus exclusively on regression problems. We propose an optimal weighting strategy that includes an ℓ_2 regularisation term into the optimisation criteria for finding the weights. In addition to this, the weights are constrained to be positive and sum to one. The reasons for this proposal are twofold. First, through the application of regularisation, we are able to overcome two problems that plague weighting strategies. The first problem is an ill condition optimisation problem caused by high multi-collinearity. The second problem is overfitting in large ensembles. Furthermore, we are able to control the bias-variance decomposition when fitting the ensemble – thus providing added flexibility into the modelling task.

To find the optimal weights, it is necessary to minimise a quadratic loss criterion that accounts for the error between the predictions of the weighted ensemble and the true response. Due to the constraints imposed on the weights the method of Lagrange multipliers is used. The positivity constraint means that there is no analytical solution to this problem. Thus, we consider methods from convex optimisation to aid our cause.

We provide context to principals of optimisation theory, specifically we consider primal and dual optimal problems. Attention is given to stopping criteria for iterative procedures. The most important concept we investigate is interior points methods which is the algorithm of choice for finding the weights in our proposed method.

The result demonstrate that the proposal has merit. In the simulation experiments, the technique was able to exclusively outperform the alternatives on one of the data sets (the Friedman 1 data set). On the two other data sets considered (the Friedman 2 and Friedman 3 data sets), the technique was only slightly worse than the best performing method. In the real-world data set experiments, the proposed technique was able to outperform the alternatives in all cases (the Boston house price data set and the Ozone data set). It is noted that the better performance on the real-world experiments is possibly due to the larger training samples. We demonstrate that the technique provides flexibility in controlling the bias-variance trade-off. In this regard, the optimal solutions in the simulation

experiments are shown to correspond to a point with larger variance and lower bias than that of an equally weighted alternative.

7.2 FURTHER RESEARCH

The natural path on from this thesis is to consider the application of regularised optimal weighting to classification problems. Let us firstly consider weighted voting. In weighted voting the idea is to give the predictions of classifiers that have better performance more weight. This is the same strategy employed in weighted averaging. Suppose that we have a set of B individual classifiers h_1, \dots, h_B . Further, assume that for an observation \underline{x} the predictions of the classifier h_i are given as a vector $(h_i^1(\underline{x}), \dots, h_i^k(\underline{x}))$ which denotes k possible class labels. Hence, h_i^j is the output of classifier h_i for class label c_j . In the case of weighted voting the class label c_t is predicted where

$$t = \arg \max_j \sum_{i=1}^B w_i \hat{h}_i^j(\underline{x}), \quad (7.1)$$

and where w_i is the weighted assigned to the predictions of classifier \hat{h}_i in the ensemble. This implies that the predicted class is the class that attains the highest weighted probability score. Furthermore, the weights in weighted voting can be constrained in the same manner as in weighted averaging, namely:

$$w_i \geq 0 \text{ and } \sum_{i=1}^B w_i = 1. \quad (7.2)$$

It is possible to solve for the weights directly in this case. Assume that the individual classifiers in the ensemble are conditionally independent and let $\hat{\underline{\ell}} = (\hat{\ell}_1, \dots, \hat{\ell}_B)^T$ be a vector containing the class predictions for the individual classifiers on an observation \underline{x} i.e. $\hat{h}_i(\underline{x}) = \hat{\ell}_i$. Also, let a_i denote the accuracy of classifier \hat{h}_i . By using a Bayesian discriminant function, we get that

$$\hat{h}^j(\underline{x}) = \log(p(c_j)p(\hat{\underline{\ell}} | c_j)). \quad (7.3)$$

Note that (7.3) gives the estimated probability of the observation \underline{x} belonging to the true class c_j , where the estimate is with respect to the ensemble classifier. Due to the conditional independence assumption, we have that

$$p(\hat{\underline{\ell}} | c_j) = \prod_{i=1}^B p(\hat{\ell}_i | c_j). \quad (7.4)$$

Given this, and through algebraic manipulation, we get

$$\begin{aligned}
\hat{h}^j(\underline{x}) &= \log p(c_j) + \sum_{i=1}^B \log p(\hat{\ell}_i | c_j) \\
&= \log p(c_j) + \log \left[\prod_{i=1, \hat{\ell}_i=c_j}^B p(\hat{\ell}_i | c_j) \prod_{i=1, \hat{\ell}_i \neq c_j}^B p(\hat{\ell}_i | c_j) \right] \\
&= \log p(c_j) + \log \left[\prod_{i=1, \hat{\ell}_i=c_j}^B a_i \prod_{i=1, \hat{\ell}_i \neq c_j}^B (1 - a_i) \right] \\
&\Rightarrow \hat{h}^j(\underline{x}) = \log p(c_j) + \sum_{i=1, \hat{\ell}_i=c_j}^B \log \frac{a_i}{1 - a_i} + \sum_{i=1}^B \log(1 - a_i)
\end{aligned} \tag{7.5}$$

In (7.5), $\sum_{i=1}^B \log(1 - a_i)$ does not depend on the class label c_j and the condition that $\hat{\ell}_i = c_j$ can be expressed as $\hat{h}_i^j(\underline{x})$, hence $\hat{h}^j(\underline{x})$ can be reduced to the form

$$\hat{h}^j(\underline{x}) = \log p(c_j) + \sum_{i=1, \hat{\ell}_i=c_j}^B \hat{h}_i^j(\underline{x}) \log \frac{a_i}{1 - a_i}. \tag{7.6}$$

From (7.6) we deduced that the optimal weights are given as

$$w_i \propto \log \frac{a_i}{1 - a_i}, \tag{7.7}$$

and therefore, the weights should be proportional to the performance of the individual classification models in the ensemble. It should be noted that the preceding arguments rely on the assumption that there is independence among the classifiers. Starting from the derivation above, one could adapt the criteria by introducing an ℓ_2 penalty term and defining an appropriate optimisation criterion.

REFERENCES

- Andersen, M., Dahl, J., and Vandenberghe, L. 2015. *CVXOPT*. Python software for convex optimization, version 1.1. Available: <http://cvxopt.org/>.
- Bauer, E. and Kohavi, R 1999. *An empirical comparison of voting classification algorithms: Bagging, boosting, and variants*. Machine Learning, 36(1-2), 105–139.
- Boyd, S. and Vandenberghe, L. 2004. *Convex Optimization*. Cambridge: Cambridge University Press.
- Breiman, L. 1994. *Bagging predictors*. Machine Learning, 24(2), 123–140.
- Breiman, L. 1996a. *Out-of-bag estimation*. University of California: Department of Statistics. Available: <https://www.stat.berkeley.edu/~breiman/OOBestimation.pdf>
- Breiman, L. 2001. *Random forests*. Machine Learning, 45(1), 5–32.
- Breiman, L. and Friedman, J. H. 1985. *Estimating optimal transformations for multiple regression and correlation*. Journal of the American Statistical Association, 80, 580-78.
- Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. 1984. *Classification and Regression Trees*. Chapman and Hall: Boca Raton, Florida.
- Dasarathy, V. and Sheela, B.V 1979. *Composite classifier system design: concepts and methodology*. Proceedings of the IEEE, 67(5), 708–713.
- Dietterich, T. G. 2000. *An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization*. Machine Learning, 40(2), 139–157.
- Dietterich, T. G. 2000. *Ensemble methods in machine learning*. International Workshop on Multiple Classifier Systems. Sardinia, Italy.
- Dua, D. and Graff, C. 2019. *UCI Machine Learning Repository*. University of California, School of Information and Computer Science: Irvine, CA. Available: [<http://archive.ics.uci.edu/ml>].
- Efron, B. 1979. *Bootstrap Methods: Another Look at the Jackknife*. Ann. Statist, 7(1), 1--26.

- Efron, B. and Tibshirani, R. 1993. *An Introduction to the Bootstrap*. Chapman & Hall: New York, NY.
- Freund, Y. 1995. *Boosting a weak learning algorithm by majority*. Information and Computation, 121(2), 256–285.
- Freund, Y. 2001. *An adaptive version of the boost by majority algorithm*. Machine Learning, 43(3), 293–318.
- Freund, Y. and Schapire R. E. 1997. *Decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55(1), 119–139.
- Friedman, J. H. 1991. *Multivariate adaptive regression splines (with discussion)*. Annals of Statistics, 19, 1-141.
- Friedman, J. H. and Hall, P. 2000. *On bagging and nonlinear estimation*. Journal of Statistical Planning and Inference, 137(3), 669–683.
- Hansen, L. K. and Salamon, P. 1990. *Neural network ensembles*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10), 993–1001.
- Harrison, Jr. and Rubinfeld, D. 1978. *Hedonic housing prices and the demand for clean air*. Journal of Environmental Economics and Management 5(1), 81-102.
- Hastie, T., Tibshirani, R. and Friedman, J. H. 2009. *The elements of statistical learning: data mining, inference, and prediction*. 2nd ed. Springer: New York.
- Ho, T. K., Hull, J. J. and Srihari, S. N. 1994. *Decision combination in multiple classifier systems*. IEEE Transaction on Pattern Analysis and Machine Intelligence, 16(1), 66–75.
- Hoeffding, W. 1963. *Probability inequalities for sums of bounded random variables*. Journal of the American Statistical Association. 58 (301), 13–30.
- Hoerl, A. and Kennard, R. 1970. *Ridge regression: Biased estimation for nonorthogonal problems*. Technometrics 12(1), 55-67.

- Hyafil, L. and Rivest, R. 1976. *Constructing Optimal Binary Decision Trees is NP-Complete*. Information Processing Letters, 5(1), 15-17.
- Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. 1991. *Adaptive mixtures of local experts*. Neural Computation, 3(1), 79–87.
- James, G., Witten, D., Hastie, T., and Tibshirani, R. 2017. *An introduction to statistical learning*. Springer: New York.
- Jordan, M. J. and Jacobs, R. A. 1994. *Hierarchical mixtures of experts and the EM algorithm*. Neural Computation, 6(2), 181–214.
- Kearns, M. and Valiant, L.G. 1989. *Cryptographic limitations on learning Boolean formulae and finite automata*. Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. ACM Press: New York, NY.
- Kittler, J., Hatef, M., Duin, R. and Matas, J. 1998. *On combining classifiers*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3), 226–239.
- Kuncheva, L. I. 2005. *Combining pattern classifiers, methods and algorithms*. Wiley Interscience: New York, New York:
- Lam, L. and Suen, S. Y. 1997. *Application of majority voting to pattern recognition: An analysis of its behavior and performance*. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, 27(5), 553–568.
- Murphy, P. (2012). *Machine learning: a probabilistic perspective*. MIT Press: Cambridge, MA.
- Opitz, D. and Maclin, R. 1999. *Popular ensemble methods: An empirical study*. Journal of Artificial Intelligence Research, 11, 169–198.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. 2011. *Scikit-learn, Machine Learning in Python*, 12, 2825-2830.
- Perrone, M. P. and Cooper, L. N. 1993. *When networks disagree: Ensemble method for neural networks*. Artificial Neural Networks for Speech and Vision. Chapman & Hall, New York, NY.

- Quinlan, J. R. 1986. *Induction of Decision Trees*. Machine Learning, 1(1), 81–106.
- Quinlan, J. R. 1993. *C4.5. Programs for Machine Learning*. Morgan Kaufmann Publishers.
- Rogova, G. 1994. *Combining the results of several neural network classifiers*. Neural Networks, 7(5), 777–781.
- Ross, S.M. 2013. *Simulation*. Academic Press.
- Schapire, R. E. 1990. *The strength of weak learnability*. Machine Learning, 5(2), 197–227.
- Tibshirani, R. 1996. *Regression shrinkage and selection via the lasso*. Journal of the Royal Statistical Society, 267-288.
- Ting, K. M. and Witten, I. H. 1999. *Issues in stacked generalization*. Journal of Artificial Intelligence Research, 10, 271–289.
- Wolpert, D. H. 1992. *Stacked generalization*. Neural Networks, 5(2), 241–259.
- Woods, K., Kegelmeyer, W. P and Bowyer, K. 1997. *Combination of multiple classifiers using local accuracy estimates*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(4), 405–410.
- Xu, L., Krzyzak, A. and Suen, C. Y. 1992. *Methods of combining multiple classifiers and their applications to handwriting recognition*. IEEE Transactions on Systems Man and Cybernetics, 22(3), 418–435.
- Zhou, Z. H. 2012. *Ensemble Methods*. Foundations and Algorithms. Chapman & Hall: Boca Raton, FL.

APPENDIX A

A.1 DEFINITIONS

A.1.1 Line

Given two points $\underline{x}_1, \underline{x}_2 \in \mathbb{R}^n$ with $\underline{x}_1 \neq \underline{x}_2$, the line passing through \underline{x}_1 and \underline{x}_2 is defined to be the set of points

$$\{\underline{z} \mid \underline{z} = \theta \underline{x}_1 + (1 - \theta) \underline{x}_2, \underline{z} \in \mathbb{R}^n, \theta \in \mathbb{R}\}. \quad (\text{A.1})$$

A.1.2 Affine set

A set $C \subseteq \mathbb{R}^n$ is by definition affine if the line passing through any two distinct points in the set is contained in the set. This implies that: $\theta \underline{x}_1 + (1 - \theta) \underline{x}_2 \in C$ for $\underline{x}_1, \underline{x}_2 \in C$ with $\underline{x}_1 \neq \underline{x}_2$ and $\theta \in \mathbb{R}$. In general, the combination of k such points $\theta_1 \underline{x}_1 + \dots + \theta_k \underline{x}_k$ with the constraint that $\sum_k \theta_k = 1$ is called an affine combination. Through inductive reasoning it can be shown that an affine set contains every affine combination of its points. Furthermore, it is possible to express an affine set as the sum of a subspace and an offset. This latter representation is given as

$$C = V + \underline{x}_0 = \{\underline{v} + \underline{x}_0 \mid \underline{v} \in V\}, \quad (\text{A.2})$$

where V is a subspace associated to C of the form

$$V = C - \underline{x}_0 = \{\underline{x} - \underline{x}_0 \mid \underline{x} \in C\}, \quad (\text{A.3})$$

and $\underline{x}_0 \in C$.

A.1.3 Affine hull

The set of all affine combinations of C is called the affine hull of C and it is denoted as: $\mathbf{aff} C$, where

$$\mathbf{aff} C = \{\theta_1 \underline{x}_1 + \dots + \theta_k \underline{x}_k \mid \underline{x}_1, \dots, \underline{x}_k \in C, \theta_1 + \dots + \theta_k = 1\}. \quad (\text{A.4})$$

A.1.4 Interior and relative interior

The interior of a set C is defined to be the collection of points $\underline{x} \in C$ that are not boundary points. A point $\underline{x} \in C$ is a boundary point if a neighbourhood around \underline{x} contains at least one point that is in C and a least one point that is not in C . The relative interior of a set is defined to be its interior relative to its affine hull. Hence, the relative interior of a set C , which is denoted $\mathbf{relint} C$, is defined to be

$$\mathbf{relint} C = \{\underline{x} \in C \mid B(\underline{x}, r) \cap \mathbf{aff} C \subseteq C \text{ for some } r > 0\}, \quad (\text{A.5})$$

where $B(\underline{x}, r) = \{\underline{z} \mid \|\underline{z} - \underline{x}\| \leq r\}$ is a ball of radius $r \in \mathbb{R}$ centred at the point \underline{x} and $\|\cdot\|$ is any norm function.

A.1.5 Convex set

A set $C \subseteq \mathbb{R}^n$ is defined to be convex if the line segment between any two points in C lies completely in C . Hence, for any two points: $\underline{x}_1, \underline{x}_2 \in C$, and any $\theta \in \mathbb{R}$, for which it holds: $0 \leq \theta \leq 1$, then $\theta \underline{x}_1 + (1 - \theta) \underline{x}_2 \in C$. Notice that the definition of a convex set differs from that of an affine set due to the constraint imposed on θ .

A.1.6 Cone

A set $C \subseteq \mathbb{R}^n$ is called a cone if for every $\underline{x} \in C$ and $\theta \geq 0$ it is true that $\theta \underline{x} \in C$.

A.1.7 Convex cone

The set C is called a convex cone if it is convex and a cone. The implication of this is that for any points $\underline{x}_1, \underline{x}_2 \in C$ for which $\theta_1, \theta_2 \geq 0$: $\theta_1 \underline{x}_1 + \theta_2 \underline{x}_2 \in C$. Geometrically this structure can be described as a two-dimensional pie slice with apex 0 and edges passing through the point \underline{x}_1 and \underline{x}_2 .

A.1.8 Dual cone

Let the set C be a cone. The set $C^* = \{\underline{z} \mid \underline{x}^T \underline{z} \geq 0 \text{ for all } \underline{x} \in C\}$ is called the dual cone of C . An interesting fact about the dual cone is that it is always a convex cone regardless of whether the cone C is convex.

A.1.9 Nonnegative orthant

The non-negative orthant, denoted \mathbb{R}_+^n , is defined as a set of points that contain only non-negative components. Hence, $\mathbb{R}_+^n = \{\underline{x} \in \mathbb{R}^n \mid \underline{x} \geq 0\}$. The symbol \mathbb{R}_+ denotes the set of non-negative numbers given by $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$.

A.1.10 Self-dual cone

The cone \mathbb{R}_+^n is self-dual which means that it is its own dual. The implication of the latter property is that $\underline{x}^T \underline{z} \geq 0$ for all $\underline{x} \geq 0 \Leftrightarrow \underline{z} \geq 0$.

A.1.11 Second-order cone

The second order cone is defined in the Euclidean space $\mathbb{R}^{n+1} = \mathbb{R}^n \times \mathbb{R}$, with the standard inner product imposed, as $\mathcal{L}^{n+1} = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} \mid \|x\|_2 \leq t\}$. This is also sometimes called the ice cream cone or Lorentz cone.

A.1.12 Positive semi-definite cone

The cone of positive semi-definite matrices is defined as $S_+^n = \{X \in S^n \mid X \geq 0\}$. Here S^n denotes the set of symmetric matrices of shape $n \times n$ and $X \geq 0$ denotes that the matrix X is positive semi-definite.

A.1.13 Convex function

A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ (which is a mapping from \mathbb{R}^n to \mathbb{R}) is convex if the domain of f , which is denoted $\mathbf{dom} f$, is a convex set and if for all $\underline{x}, \underline{z} \in \mathbf{dom} f$ with the constraint that $0 \leq \theta \leq 1$, we have

$$f(\theta \underline{x} + (1 - \theta)\underline{z}) \leq \theta f(\underline{x}) + (1 - \theta)f(\underline{z}). \quad (\text{A.6})$$

The condition in (A.6) is equivalent to declaring that a line segment from the point $(\underline{x}, f(\underline{x}))$ to the point $(\underline{z}, f(\underline{z}))$ should lie above the graph of f . In adjunct to the definition in (A.6), a function is convex if and only if it is convex on a line that intersect its domain. Therefore, f is convex if for all $\underline{x} \in \mathbf{dom} f$ and $\underline{v} \in \mathbb{R}^n$, the function $g(t) = f(\underline{x} + t\underline{v})$ is convex on the domain given by the set $\{t | \underline{x} + t\underline{v} \in \mathbf{dom} f(\cdot)\}$.

Yet another characterisation of a convex function stipulates that if a function f is differentiable, which implies that its gradient ∇f exists for all $\underline{x} \in \mathbf{dom} f$, then f is convex if and only if $\mathbf{dom} f$ is a convex set and the property

$$f(\underline{z}) \geq f(\underline{x}) + \nabla f(\underline{x})^T (\underline{z} - \underline{x}), \quad (\text{A.7})$$

holds for all $\underline{x}, \underline{z} \in \mathbf{dom} f$. The expression in (A.7) is the first order condition for convexity and it implies that the first order approximation of the Taylor series expansion of f should be a global under estimator of f if it is to be convex. The converse statement to (A.7) is also true. Moreover, (A.7) implies that if $\nabla f = 0$, then for all $\underline{z} \in \mathbf{dom} f$: $f(\underline{z}) \geq f(\underline{x})$. Hence \underline{x} is a global minimiser of f .

The first order condition for strict convexity states that f is strictly convex if and only if $\mathbf{dom} f$ is a convex set and for $\underline{x}, \underline{z} \in \mathbf{dom} f$ with $\underline{x} \neq \underline{z}$ it holds that

$$f(\underline{z}) > f(\underline{x}) + \nabla f(\underline{x})^T (\underline{z} - \underline{x}). \quad (\text{A.8})$$

A.2 COMPLEMENTARY SLACKNESS

In the discussion that follows it is assumed that the primal and dual optimal values are attained, and also that strong duality holds. Given these assertions, it holds that

$$\begin{aligned} f_0(\underline{x}^*) &= g(\underline{\lambda}^*, \underline{v}^*) \\ &= \inf_{\underline{x} \in \mathcal{D}} \left(f_0(\underline{x}) + \sum_{i=1}^m \lambda_i^* f_i(\underline{x}) + \sum_{i=1}^p v_i^* h_i(\underline{x}) \right) \\ &\leq f_0(\underline{x}^*) + \sum_{i=1}^m \lambda_i^* f_i(\underline{x}^*) + \sum_{i=1}^p v_i^* h_i(\underline{x}^*) \leq f_0(\underline{x}^*). \end{aligned} \quad (\text{A.9})$$

The first line arises from the assumption of strong duality and the second line is due to the definition of the dual function. The third line holds since the infimum is less than or equal to the Lagrangian function when evaluated at a primal optimal point, i.e. $\underline{x} = \underline{x}^*$. Finally, the last inequality is due to the fact that the Lagrangian terms are positive, i.e. $\lambda_i^* \geq 0$ and because $f_i(\underline{x}^*) \leq 0, i = 1, \dots, m$, and $h_i(\underline{x}^*) = 0, i = 1, \dots, p$. Thus,

$$\sum_{i=1}^m \lambda_i^* f_i(\underline{x}^*) = 0, \quad (\text{A.10})$$

and hence,

$$\lambda_i^* f_i(\underline{x}^*) = 0, i = 1, \dots, m, \quad (\text{A.11})$$

since every term in the sum in (A.10) is non-positive. This condition in (A.11) has been designated the qualifier of complementary slackness. This condition holds for any primal optimal point \underline{x}^* and any dual optimal point $(\underline{\lambda}^*, \underline{v}^*)$ for which strong duality holds. Alternatively, complementary slackness can be characterised by the conditions

$$\lambda_i^* > 0 \Rightarrow f_i(\underline{x}^*) = 0, \quad (\text{A.12})$$

or equivalently

$$f_i(\underline{x}^*) < 0 \Rightarrow \lambda_i^* = 0. \quad (\text{A.13})$$

The condition in (A.13) states that the optimal Lagrange multiplier at index i is zero unless the corresponding constraint function is active at the optimum.

A.3 KARUSH-KUHN-TUCKER CONDITIONS

In this section consideration is given to the Karush-Kuhn-Tucker (KKT) optimality conditions. Throughout this material presented here it is assumed that $f_i, i = 1, \dots, m$ and $h_i, i = 1, \dots, p$ are differentiable functions with open domains. The assumption of an open domain stems from the definition of differentiability as a two-sided limit.

The KKT conditions are important in the context of optimisation. In some instances, it is possible to find a solution to the KKT conditions analytically. However, more often an algorithm, which is used to solve a convex optimisation problem, can be interpreted to be a method for solving the KKT conditions.

A.3.1 KKT conditions for non-convex problems

Let \underline{x}^* and $(\underline{\lambda}^*, \underline{v}^*)$ be any primal and dual optimal points for which there is zero duality gap. Then, due to the point \underline{x}^* being a minimising solution for the Lagrangian function $L(\underline{x}, \underline{\lambda}^*, \underline{v}^*)$ over \underline{x} , the gradient must 0 at \underline{x}^* . Therefore,

$$\nabla f_0(\underline{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(\underline{x}^*) + \sum_{i=1}^p v_i^* \nabla h_i(\underline{x}^*) = 0. \quad (\text{A.14})$$

From (A.14) it follows that the following set of conditions hold:

$$\begin{aligned} f_i(\underline{x}^*) &\leq 0, i = 1, \dots, m \\ \lambda_i^* &\geq 0, i = 1, \dots, m \\ h_i(\underline{x}^*) &= 0, i = 1, \dots, p \\ \lambda_i^* f_i(\underline{x}^*) &= 0, i = 1, \dots, m \end{aligned} \quad (\text{A.15})$$

$$\nabla f_0(\underline{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(\underline{x}^*) + \sum_{i=1}^p v_i^* \nabla h_i(\underline{x}^*) = 0.$$

The set of conditions in (A.15) defines the KKT conditions. The result in (A.15) implies that for any optimisation problem that has a differentiable objective function and a set of constraint functions for which strong duality holds, any pair of primal and dual optimal points must satisfy the KKT conditions.

A.3.2 KKT conditions for convex problems

The KKT conditions are a sufficient criterion for points to be primal and dual optimal if the primal problem is convex. Recall that the primal problem is convex if $f_i, i = 1, \dots, m$ are convex and $h_i, i = 1, \dots, p$ are affine. Thus, if the points $\tilde{\underline{x}}$ and $(\tilde{\underline{\lambda}}, \tilde{\underline{v}})$ satisfy the KKT conditions, $\tilde{\underline{x}}$ and $(\tilde{\underline{\lambda}}, \tilde{\underline{v}})$ are primal dual optimal, with zero duality gap. The first two conditions in (A.16) imply that $\tilde{\underline{x}}$ is primal feasible. The final condition in (A.16) implies that the gradient of the Lagrangian function is zero at the optimal point $\tilde{\underline{x}}$. Remember that this holds since $L(\underline{x}, \tilde{\underline{\lambda}}, \tilde{\underline{v}})$ is convex in \underline{x} for $\tilde{\lambda}_i \geq 0$. Whence it follows that $\tilde{\underline{x}}$ minimises $L(\tilde{\underline{x}}, \tilde{\underline{\lambda}}, \tilde{\underline{v}})$ over \underline{x} and we conclude that

$$\begin{aligned} g(\tilde{\underline{\lambda}}, \tilde{\underline{v}}) &= L(\tilde{\underline{x}}, \tilde{\underline{\lambda}}, \tilde{\underline{v}}) \\ &= f_0(\tilde{\underline{x}}) + \sum_{i=1}^m \tilde{\lambda}_i f_i(\tilde{\underline{x}}) + \sum_{i=1}^p \tilde{v}_i h_i(\tilde{\underline{x}}) \\ &= f_0(\tilde{\underline{x}}). \end{aligned} \quad (\text{A.16})$$

The result in the last line of (A.16) follows from the fact that $h_i(\tilde{\underline{x}}) = 0$ and $\tilde{\lambda}_i f_i(\tilde{\underline{x}}) = 0$. Hence, from the observation that $g(\tilde{\underline{\lambda}}, \tilde{\underline{v}}) = f_0(\tilde{\underline{x}})$ we conclude that $\tilde{\underline{x}}$ and $(\tilde{\underline{\lambda}}, \tilde{\underline{v}})$ have zero duality gap which implies that they are primal and dual optimal.

To summarise the preceding results, any points that adhere to the KKT conditions are primal and dual optimal with zero duality gap if the optimization problem is convex with differentiable objective and constraint functions. Furthermore, if the same problem satisfies Slater's constraint qualifications, the

KKT conditions are necessary and sufficient for optimality. In this latter scenario, the optimal duality gap is zero and the dual optimal value is attained.

A.4 UNCONSTRAINED OPTIMISATION PROBLEMS

In the section that follows the current, Newtons method will be discussed. Newtons method gives a procedure for solving optimisation problems of the form

$$\text{minimise } f(\underline{x}). \quad (\text{A.17})$$

Furthermore, Newtons method is used in the path following algorithm that is the subject of the last section of this chapter. The task in (A.17) is called an unconstrained optimisation problem. Here the objective function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is convex and twice continuously differentiable. In regard to developing theory for solving (A.17) it is assumed in the discussion that follows, that there exists an optimal point \underline{x}^* . In agreement with prior notation, the optimal value, is denoted

$$\inf_{\underline{x}} f(\underline{x}) = f(\underline{x}^*) = p^*. \quad (\text{A.18})$$

Further, due to f being differentiable and convex, the condition that

$$\nabla f(\underline{x}^*) = 0, \quad (\text{A.19})$$

is a necessary and sufficient for a point \underline{x}^* to be optimal.

The solution to (A.17) is equivalent to the solution to (A.19). However, the latter is a set of n equations in n unknowns $\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n$. Commonly, (A.19) is solved using an iterative algorithm. In this regard the procedure involves computing a converging sequence of solutions $\underline{x}^{(0)}, \underline{x}^{(1)}, \dots \in \mathbf{dom} f$ with $f(\underline{x}^{(k)}) \rightarrow p^*$ as $k \rightarrow \infty$. The sequence of solutions is called a minimising sequence for the problem (A.17) and the algorithm terminates when $f(\underline{x}^{(k)}) - p^* \leq \varepsilon$, where $\varepsilon > 0$ is some specified tolerance.

A.4.1 Initial point and sublevel set

Iterative methods that solve problems of the form in (A.18) require a suitable starting point, which we denote as $\underline{x}^{(0)}$. The starting point must be contained within the domain of the objective function, i.e. $\underline{x}^{(0)} \in \mathbf{dom} f$. In addition, the sublevel set

$$S = \{\underline{x} \in \mathbf{dom} f \mid f(\underline{x}) \leq f(\underline{x}^{(0)})\}, \quad (\text{A.20})$$

must be closed. The condition in (A.20) is satisfied for all $\underline{x}^{(0)} \in \mathbf{dom} f$ if all the function sublevel sets are closed. Moreover, continuous functions for which $\mathbf{dom} f = \mathbb{R}^n$ are closed, whence the initial sublevel set condition is satisfied by any $\underline{x}^{(0)}$.

A.5 DECENT METHOD

In the discussion that ensues it is assumed that the objective function is strongly convex on the sublevel set S where S is defined as in (A.20). The latter condition implies that there exists a constant $m > 0$ for which

$$\nabla^2 f(\underline{x}) \geq mI, \quad (\text{A.21})$$

for all $\underline{x} \in S$. Newton's method is fundamentally a decent technique. To this end the aim is to find a sequence of minimising points $\underline{x}^{(k)}, k = 1, \dots$, where

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} + t^{(k)} \Delta \underline{x}^{(k)}, \quad (\text{A.22})$$

with $t^{(k)} \geq 0$ unless $\underline{x}^{(k)}$ is optimal. In (A.22) k denotes the iteration number and $\Delta \underline{x}^{(k)}$ is the step direction - in which case $t^{(k)}$ represents the step size. Note, however, that the step direction does not need to have unit norm. The method is a decent method because the sequence of minimising points satisfies

$$f(\underline{x}^{(k+1)}) < f(\underline{x}^{(k)}). \quad (\text{A.22})$$

The condition in (A.23) means that all minimising points lie in the initial sublevel set which in turn means that they are in the domain of the objective function.

Due to the property of convexity it holds that for all k and $\underline{x}^{(k)} \in S$,

$$\nabla f(\underline{x}^{(k)})^T (\underline{z} - \underline{x}^{(k)}) \geq 0. \quad (\text{A.23})$$

This condition implies that $f(\underline{z}) \geq f(\underline{x}^{(k)})$ where $\underline{z} \in \mathbf{dom} f$. Therefore, the search direction satisfies

$$\Delta f(\underline{x}^{(k)})^T \Delta \underline{x}^{(k)} < 0, \quad (\text{A.24})$$

which means that the search direction should make an acute angle with the negative gradient. The general structure of a decent algorithm is given as

Algorithm A.1: Decent method

-
1. **input:** starting point $\underline{x} \in \mathbf{dom} f$
 2. **repeat:**
 3. determine the decent direction $\Delta \underline{x}$
 4. line search.
 5. update. $\underline{x} := \underline{x} + t \Delta \underline{x}$
 6. **end:** when stopping criteria is reached
-

In Algorithm A.1, point 3 refers to the selection of the step size t that determines where along the search line $\{\underline{x} + t \Delta \underline{x} | t \in \mathbb{R}_+\}$ the next iterate will be. Details regarding how to determine the decent direction will be given in Subsection A.5.1. In the following subsection the matter of line search will be elaborated on.

A.5.1 Line search

There are two categories of line search procedures. The first category is exact line search and the second is inexact line search. In the exact procedure, the step size t is found to minimise f along a ray $\{\underline{x} + t \Delta \underline{x} | t \geq 0\}$. Hence,

$$t = \arg \min_{s \geq 0} f(\underline{x} + s \Delta \underline{x}). \quad (\text{A.25})$$

However, in practice it is common to use an inexact line search procedure. In the latter case, the step length is chosen to approximately minimise f along the ray $\{\underline{x} + t \Delta \underline{x} | t \geq 0\}$. In fact, it may be sufficient to merely reduce the value f . One method of inexact line search that is particularly effective and simple is backtracking line search. The backtracking procedure relies on two constants α, β with $0 < \alpha < 0.5$ and $0 < \beta < 1$. Algorithm A.2 outlines the procedure.

Algorithm A.2: Backtracking line search

-
1. **input:** a descent direction $\Delta \underline{x}$ for f at $\underline{x} \in \text{dom } f, \alpha \in (0,0.5), \beta \in (0,1), t := 1$
 2. **repeat:**
 3. $t := \beta t$
 4. **end:** $f(\underline{x} + t \Delta \underline{x}) \leq f(\underline{x}) + \alpha t \nabla f(\underline{x})^T \Delta \underline{x}$
-

The qualifier “backtracking” comes from the fact that the first step size is of size 1 with subsequent steps being reduced by a factor β until the stopping condition $f(\underline{x} + t \Delta \underline{x}) \leq f(\underline{x}) + \alpha t \nabla f(\underline{x})^T \Delta \underline{x}$ is realised. Since $\Delta \underline{x}$ is a descent direction, $\nabla f(\underline{x})^T \Delta \underline{x} < 0$, therefore for small enough t we have

$$f(\underline{x} + t \Delta \underline{x}) \approx f(\underline{x}) + t \nabla f(\underline{x})^T \Delta \underline{x} < f(\underline{x}) + \alpha t \nabla f(\underline{x})^T \Delta \underline{x}. \quad (\text{A.26})$$

The expression in (4.53) demonstrates that the backtracking line search does terminate. The result relies on the first order Taylor approximation. The constant α is interpreted as the proportion of the decrease in f , predicted by linear extrapolation, that is acceptable.

A.5.2 Newton step

For $\underline{x} \in \text{dom } f$, the vector

$$\Delta \underline{x}_{nt} = -\nabla^2 f(\underline{x})^{-1} \nabla f(\underline{x}), \quad (\text{A.27})$$

is called the Newton step, for the function f , at the point \underline{x} . If the Hessian matrix $\nabla^2 f(\underline{x})$ is positive definite, then

$$\nabla f(\underline{x})^T \Delta \underline{x}_{nt} = -\nabla f(\underline{x})^T \nabla^2 f(\underline{x})^{-1} \nabla f(\underline{x}) < 0, \quad (\text{A.28})$$

holds except if $\nabla f(\underline{x}) = 0$, in which case \underline{x} is optimal. The Newton step can be interpreted and motivate in several ways – we discuss two of these perspectives in Subsections A.5.2.1 and A.5.2.2.

A.5.2.1 Minimiser of second-order approximation

The Newton step is a second-order Taylor approximation, denoted by \hat{f} , of f at the point \underline{x} , where

$$\hat{f}(\underline{x} + \underline{v}) = f(\underline{x}) + \nabla f(\underline{x})^T \underline{v} + \frac{1}{2} \underline{v}^T \nabla^2 f(\underline{x}) \underline{v}. \quad (\text{A.29})$$

Equation (A.29) is a convex quadratic function of \underline{v} , that is minimised when $\underline{v} = \Delta \underline{x}_{nt}$. Thus, the Newton step $\Delta \underline{x}_{nt}$ is precisely what should be added to the point \underline{x} to minimise the second-order approximation of f at the point \underline{x} .

A.5.2.2 Steepest decent direction in Hessian norm

Additionally, the Newton step is the steepest descent direction at \underline{x} , for the quadratic norm defined by the Hessian $\nabla^2 f(\underline{x})$. The quadratic norm is given by

$$\|\underline{u}\|_{\nabla^2 f(\underline{x})} = (\underline{u}^T \nabla^2 f(\underline{x}) \underline{u})^{1/2}. \quad (\text{A.30})$$

The implication of A.30 is that the Newton step should be a good search direction, and a very good search direction when \underline{x} is near \underline{x}^* .

A.5.3 Newton decrement

The quantity

$$\lambda(\underline{x}) = \left(\nabla f(\underline{x})^T \nabla^2 f(\underline{x})^{-1} \nabla f(\underline{x}) \right)^{1/2}, \quad (\text{A.31})$$

is called the Newton decrement at the point \underline{x} . The Newton decrement is of paramount importance to the specification of a stopping criterion for Newtons method by virtue of the fact that it can be related to the quantity $f(\underline{x}) - \inf_{\underline{z}} \hat{f}(\underline{z})$, where \hat{f} is the second order approximation of f at \underline{x}

$$f(\underline{x}) - \inf_{\underline{z}} \hat{f}(\underline{z}) = f(\underline{x}) - \hat{f}(\underline{x} + \Delta \underline{x}_{nt}) = \frac{1}{2} \lambda(\underline{x})^2. \quad (\text{A.32})$$

Equation (A.32) implies that the term $\frac{1}{2} \lambda(\underline{x})^2$ is an estimate of $f(\underline{x}) - p^*$ based on a quadratic approximation of f at \underline{x} . Algorithm A.3 outlines Newtons method.

Algorithm A.3: Newtons method

-
1. **input:** starting point $\underline{x} \in \text{dom} f(\cdot)$ and tolerance $\epsilon > 0$
 2. **repeat:**
 3. compute the Newton step $\Delta \underline{x}_{nt} := -\nabla^2 f(\underline{x})^{-1} \nabla f(\underline{x})$
 4. compute the Newton decrement $\lambda^2(\underline{x}) := \nabla f(\underline{x})^T \nabla^2 f(\underline{x})^{-1} \nabla f(\underline{x})$
 5. **end:** if $\lambda^2/2 \leq \epsilon$
 6. line search - choose step size t by backtracking line search
 7. update $\underline{x} := \underline{x} + t \Delta \underline{x}_{nt}$
-

The gradient and Hessian (which is used in the Newton minimisation procedure) of the logarithmic barrier function ϕ , is given by

$$\begin{aligned} \nabla \phi(\underline{x}) &= \sum_{i=1}^m -\frac{1}{f_i(\underline{x})} \nabla f_i(\underline{x}) \\ \nabla^2 \phi(\underline{x}) &= \sum_{i=1}^m \frac{1}{f_i(\underline{x})^2} \nabla f_i(\underline{x}) \nabla f_i(\underline{x})^T + \sum_{i=1}^m -\frac{1}{f_i(\underline{x})} \nabla^2 f_i(\underline{x}). \end{aligned} \quad (\text{A.33})$$

APPENDIX B

B.1 SOURCE CODE

B.1.1 Weighted bagging class

```
1 """
2 Disclaimer: Much of this code has been copied from the Sklearn
3 (http://scikit-learn.org) implementation of bagging regressor. I
4   have adapted
5 their code to weight each of the models in the ensemble optimally
6   in a least
7 squares sense.
8 """
9 import itertools
10 from abc import ABCMeta, abstractmethod
11 import numbers
12 from warnings import warn
13
14 import numpy as np
15 from sklearn.externals.six import with_metaclass
16 from sklearn.utils.validation import has_fit_parameter,
17   check_is_fitted
18 from sklearn.utils.random import sample_without_replacement
19 from sklearn.ensemble.base import BaseEnsemble,
20   _partition_estimators
21
22 from sklearn.utils import *
23 from joblib import Parallel, delayed
24 from sklearn.base import RegressorMixin
25 from sklearn.tree import DecisionTreeRegressor
26 from sklearn.metrics import r2_score
27 from sklearn.preprocessing import scale
28
29 from cvxopt import solvers, matrix
30 solvers.options['show_progress'] = False
31
32 MAX_INT = np.iinfo(np.int32).max
33
34 def _generate_indices(random_state, bootstrap, n_population,
35   n_samples):
36     """Draw randomly sampled indices."""
37     # Draw sample indices
38     if bootstrap:
39         indices = random_state.randint(0, n_population, n_samples)
40     else:
41         indices = sample_without_replacement(n_population,
42   n_samples,
43   random_state=
44   random_state)
45     return indices
```

```
41
42
43 def _generate_bagging_indices(random_state, bootstrap_features,
44                               bootstrap_samples, n_features,
45                               n_samples,
46                               max_features, max_samples):
47     """Randomly draw feature and sample indices."""
48     # Get valid random state
49     random_state = check_random_state(random_state)
50
51     # Draw indices
52     feature_indices = _generate_indices(random_state,
53                                       bootstrap_features,
54                                       n_features, max_features)
55
56     sample_indices = _generate_indices(random_state,
57                                       bootstrap_samples,
58                                       n_samples, max_samples)
59
60     return feature_indices, sample_indices
61
62 def _parallel_build_estimators(n_estimators, ensemble, X, y,
63                               sample_weight,
64                               seeds, total_n_estimators, verbose)
65 :
66     """Private function used to build a batch of estimators within
67     a job."""
68     # Retrieve settings
69     n_samples, n_features = X.shape
70     max_features = ensemble._max_features
71     max_samples = ensemble._max_samples
72     bootstrap = ensemble.bootstrap
73     bootstrap_features = ensemble.bootstrap_features
74     support_sample_weight = has_fit_parameter(ensemble.
75     base_estimator_,
76     "sample_weight")
77
78     if not support_sample_weight and sample_weight is not None:
79         raise ValueError("The base estimator doesn't support
80         sample weight")
81
82     # Build estimators
83     estimators = []
84     estimators_features = []
85
86     for i in range(n_estimators):
87         if verbose > 1:
88             print("Building estimator %d of %d for this parallel
89             run "
90                 "(total %d)..." % (i + 1, n_estimators,
```

```
119     predictions = np.asarray([estimator.predict(X[:, features])
120                               for estimator, features in zip(estimators,
121                                                             estimators_features)])
122     return np.sum(predictions * weights, axis=0).reshape(-1, 1)
123
124 class BaseBagging(with_metaclass(ABCMeta, BaseEnsemble)):
125     """Base class for Bagging meta-estimator.
126     Warning: This class should not be used directly. Use derived
127     classes
128     instead.
129     """
130
131     @abstractmethod
132     def __init__(self,
133                 base_estimator=None,
134                 n_estimators=10,
135                 max_samples=1.0,
136                 max_features=1.0,
137                 bootstrap=True,
138                 bootstrap_features=False,
139                 oob_score=False,
140                 warm_start=False,
141                 n_jobs=None,
142                 random_state=None,
143                 verbose=0,
144                 c=0):
145         super(BaseBagging, self).__init__(
146             base_estimator=base_estimator,
147             n_estimators=n_estimators)
148
149         self.max_samples = max_samples
150         self.max_features = max_features
151         self.bootstrap = bootstrap
152         self.bootstrap_features = bootstrap_features
153         self.oob_score = oob_score
154         self.warm_start = warm_start
155         self.n_jobs = n_jobs
156         self.random_state = random_state
157         self.verbose = verbose
158         self.c = c
159
160     def _get_weights(self, X, y):
161         """Private function to find a set of weights that are
162         optimal in a least
163         squares sense. The weights incorporate the l2 penalty term
164         and are
165         positive
166         """
```

```
total_n_estimators))
81
82     random_state = np.random.RandomState(seeds[i])
83     estimator = ensemble._make_estimator(append=False,
84                                         random_state=
random_state)
85
86     # Draw random feature, sample indices
87     features, indices = _generate_bagging_indices(random_state
,
88
bootstrap_features,
89
bootstrap,
90
n_features,
n_samples,
91
max_features,
max_samples)
92
93     # Draw samples, using sample weights, and then fit
94     if support_sample_weight:
95         if sample_weight is None:
96             curr_sample_weight = np.ones((n_samples,))
97         else:
98             curr_sample_weight = sample_weight.copy()
99
100         if bootstrap:
101             sample_counts = np.bincount(indices, minlength=
n_samples)
102             curr_sample_weight *= sample_counts
103         else:
104             not_indices_mask = -indices_to_mask(indices,
n_samples)
105             curr_sample_weight[not_indices_mask] = 0
106
107             estimator.fit(X[:, features], y, sample_weight=
curr_sample_weight)
108
109         else:
110             estimator.fit((X[indices])[:, features], y[indices])
111
112     estimators.append(estimator)
113     estimators_features.append(features)
114     return estimators, estimators_features
115
116
117 def _parallel_predict_regression(estimators, estimators_features,
X, weights):
118     """Private function used to compute predictions within a job.
    """
```

```

164     X = matrix(scale(X))
165     y = matrix(y)
166     m, n = X.size
167     I = matrix(0.0, (n, n))
168     I[:,n + 1] = 1.0
169     G = matrix([-I, matrix(0.0, (1, n)), I])
170     h = matrix(n * [0.0] + [self.c] + n * [0.0])
171     dims = {'l': n, 'q': [n + 1], 's': []}
172     T = matrix(1.0, (1, n))
173     s = matrix(1.0)
174     return np.asarray(solvers.coneqp(X.T * X, -X.T * y, G, h,
175     dims, T,
176                                     s)['x'])
177
178 def fit(self, X, y, X_val=None, y_val=None, sample_weight=None
179 ):
180     """Build a Bagging ensemble of estimators from the
181     training
182     set (X, y).
183     Parameters
184     -----
185     X : {array-like, sparse matrix} of shape = [n_samples,
186     n_features]
187     The training input samples. Sparse matrices are
188     accepted only if
189     they are supported by the base estimator.
190     y : array-like, shape = [n_samples]
191     The target values (class labels in classification,
192     real numbers in
193     regression).
194     sample_weight : array-like, shape = [n_samples] or None
195     Sample weights. If None, then samples are equally
196     weighted.
197     Note that this is supported only if the base estimator
198     supports
199     sample weighting.
200     Returns
201     -----
202     self : object
203     """
204     if X_val is not None and y_val is not None:
205         return self._fit(X, y, self.max_samples, X_val=X_val,
206         y_val=y_val,
207                             sample_weight=sample_weight)
208     return self._fit(X, y, self.max_samples, sample_weight=
209     sample_weight)
210
211 def _fit(self, X, y, max_samples=None, max_depth=None, X_val=
212 None,

```

```
202         y_val=None, sample_weight=None):
203         """Build a Bagging ensemble of estimators from the
training
204         set (X, y).
205         Parameters
206         -----
207         X : {array-like, sparse matrix} of shape = [n_samples,
n_features]
208             The training input samples. Sparse matrices are
accepted only if
209             they are supported by the base estimator.
210         y : array-like, shape = [n_samples]
211             The target values (class labels in classification,
real numbers in
212             regression).
213         max_samples : int or float, optional (default=None)
214             Argument to use instead of self.max_samples.
215         max_depth : int, optional (default=None)
216             Override value used when constructing base estimator.
Only
217             supported if the base estimator has a max_depth
parameter.
218         sample_weight : array-like, shape = [n_samples] or None
219             Sample weights. If None, then samples are equally
weighted.
220             Note that this is supported only if the base estimator
supports
221             sample weighting.
222         Returns
223         -----
224         self : object
225         """
226         random_state = check_random_state(self.random_state)
227
228         X, y = check_X_y(
229             X, y, ['csr', 'csc'], dtype=None, force_all_finite=
False,
230             multi_output=True
231         )
232         if sample_weight is not None:
233             sample_weight = check_array(sample_weight, ensure_2d=
False)
234             check_consistent_length(y, sample_weight)
235
236         # Remap output
237         n_samples, self.n_features_ = X.shape
238         self._n_samples = n_samples
239         y = self._validate_y(y)
240
```

```
241     # Check parameters
242     self._validate_estimator()
243
244     if max_depth is not None:
245         self.base_estimator_.max_depth = max_depth
246
247     # Validate max_samples
248     if max_samples is None:
249         max_samples = self.max_samples
250     elif not isinstance(max_samples, (numbers.Integral, np.
integer)):
251         max_samples = int(max_samples * X.shape[0])
252
253     if not (0 < max_samples <= X.shape[0]):
254         raise ValueError("max_samples must be in (0, n_samples
]")
255
256     # Store validated integer row sampling value
257     self._max_samples = max_samples
258
259     # Validate max_features
260     if isinstance(self.max_features, (numbers.Integral, np.
integer)):
261         max_features = self.max_features
262     elif isinstance(self.max_features, np.float):
263         max_features = self.max_features * self.n_features_
264     else:
265         raise ValueError("max_features must be int or float")
266
267     if not (0 < max_features <= self.n_features_):
268         raise ValueError("max_features must be in (0,
n_features]")
269
270     max_features = max(1, int(max_features))
271
272     # Store validated integer feature sampling value
273     self._max_features = max_features
274
275     # Other checks
276     if not self.bootstrap and self.oob_score:
277         raise ValueError("Out of bag estimation only available
"
278                             " if bootstrap=True")
279
280     if self.warm_start and self.oob_score:
281         raise ValueError("Out of bag estimate only available"
282                             " if warm_start=False")
283
284     if hasattr(self, "oob_score_") and self.warm_start:
```

```
285         del self.oob_score_
286
287         if not self.warm_start or not hasattr(self, 'estimators_')
288         :
289             # Free allocated memory, if any
290             self.estimators_ = []
291             self.estimators_features_ = []
292
293             n_more_estimators = self.n_estimators - len(self.
294             estimators_)
295
296             if n_more_estimators < 0:
297                 raise ValueError('n_estimators=%d must be larger or
298                 equal to '
299                 'len(estimators_)=%d when warm_start
300                 ==True'
301                 % (self.n_estimators, len(self.
302                 estimators_)))
303
304             elif n_more_estimators == 0:
305                 warn("Warm-start fitting without increasing
306                 n_estimators does not "
307                 "fit new trees.")
308                 return self
309
310             # Parallel loop
311             n_jobs, n_estimators, starts = _partition_estimators(
312             n_more_estimators,
313
314             self.
315             n_jobs)
316             total_n_estimators = sum(n_estimators)
317
318             # Advance random state to state after training
319             # the first n_estimators
320             if self.warm_start and len(self.estimators_) > 0:
321                 random_state.randint(MAX_INT, size=len(self.
322                 estimators_))
323
324             seeds = random_state.randint(MAX_INT, size=
325             n_more_estimators)
326             self._seeds = seeds
327
328             all_results = Parallel(n_jobs=n_jobs, verbose=self.verbose
329             )(
330                 delayed(_parallel_build_estimators)(
331                 n_estimators[i],
332                 self,
333                 X,
334                 y,
```

```
323         sample_weight,
324         seeds[starts[i]:starts[i + 1]],
325         total_n_estimators,
326         verbose=self.verbose)
327     for i in range(n_jobs)
328
329     # Reduce
330     self.estimators_ += list(itertools.chain.from_iterable(
331         t[0] for t in all_results))
332     self.estimators_features_ += list(itertools.chain.
333 from_iterable(
334         t[1] for t in all_results))
335     if X_val is not None and y_val is not None:
336         Z = np.zeros((X_val.shape[0], len(self.estimators_)))
337         for i, model in enumerate(self.estimators_):
338             Z[:, i] = model.predict(X_val)
339         self.weights_ = self._get_weights(Z, y_val.reshape(-1,
340 1))
341     else:
342         Z = np.zeros((X.shape[0], len(self.estimators_)))
343         for i, model in enumerate(self.estimators_):
344             Z[:, i] = model.predict(X)
345         self.weights_ = self._get_weights(Z, y.reshape(-1, 1))
346     if self.oob_score:
347         self._set_oob_score(X, y)
348
349     return self
350
351 @abstractmethod
352 def _set_oob_score(self, X, y):
353     """Calculate out of bag predictions and score."""
354
355 def _validate_y(self, y):
356     if len(y.shape) == 1 or y.shape[1] == 1:
357         return column_or_id(y, warn=True)
358     else:
359         return y
360
361 def _get_estimators_indices(self):
362     # Get drawn indices along both sample and feature axes
363     for seed in self._seeds:
364         # Operations accessing random_state must be performed
365         # identically
366         # to those in `_parallel_build_estimators()`
367         random_state = np.random.RandomState(seed)
368         feature_indices, sample_indices =
369         _generate_bagging_indices(
370             random_state, self.bootstrap_features, self.
371 bootstrap,
```

```
367         self.n_features_, self._n_samples, self.  
368         _max_features,  
369         self._max_samples)  
370  
371         yield feature_indices, sample_indices  
372  
373     @property  
374     def estimators_samples_(self):  
375         """The subset of drawn samples for each base estimator.  
376         Returns a dynamically generated list of indices  
377         identifying  
378         the samples used for fitting each member of the ensemble,  
379         i.e.,  
380         the in-bag samples.  
381         Note: the list is re-created at each call to the property  
382         in order  
383         to reduce the object memory footprint by not storing the  
384         sampling  
385         data. Thus fetching the property may be slower than  
386         expected.  
387         """  
388         return [sample_indices for _, sample_indices in  
389                 self._get_estimators_indices()]  
390  
391     class Weighted(BaseBagging, RegressorMixin):  
392         """A Bagging regressor.  
393         A Bagging regressor is an ensemble meta-estimator that fits  
394         base  
395         regressors each on random subsets of the original dataset and  
396         then  
397         aggregate their individual predictions (either by voting or by  
398         averaging)  
399         to form a final prediction. Such a meta-estimator can  
400         typically be used as  
401         a way to reduce the variance of a black-box estimator (e.g., a  
402         decision  
403         tree), by introducing randomization into its construction  
404         procedure and  
405         then making an ensemble out of it.  
406         This algorithm encompasses several works from the literature.  
407         When random  
408         subsets of the dataset are drawn as random subsets of the  
409         samples, then  
410         this algorithm is known as Pasting [1]_. If samples are drawn  
411         with  
412         replacement, then the method is known as Bagging [2]_. When  
413         random subsets  
414         of the dataset are drawn as random subsets of the features,
```

```
then the method
400 is known as Random Subspaces [3]_. Finally, when base
estimators are built
401 on subsets of both samples and features, then the method is
known as
402 Random Patches [4]_.
403 Read more in the :ref:`User Guide <bagging>`.
404 Parameters
405 -----
406 base_estimator : object or None, optional (default=None)
407     The base estimator to fit on random subsets of the dataset
.
408     If None, then the base estimator is a decision tree.
409 n_estimators : int, optional (default=10)
410     The number of base estimators in the ensemble.
411 max_samples : int or float, optional (default=1.0)
412     The number of samples to draw from X to train each base
estimator.
413     - If int, then draw `max_samples` samples.
414     - If float, then draw `max_samples * X.shape[0]` samples.
415 max_features : int or float, optional (default=1.0)
416     The number of features to draw from X to train each base
estimator.
417     - If int, then draw `max_features` features.
418     - If float, then draw `max_features * X.shape[1]` features
.
419 bootstrap : boolean, optional (default=True)
420     Whether samples are drawn with replacement. If False,
sampling
421     without replacement is performed.
422 bootstrap_features : boolean, optional (default=False)
423     Whether features are drawn with replacement.
424 oob_score : bool
425     Whether to use out-of-bag samples to estimate
426     the generalization error.
427 warm_start : bool, optional (default=False)
428     When set to True, reuse the solution of the previous call
to fit
429     and add more estimators to the ensemble, otherwise, just
fit
430     a whole new ensemble. See :term:`the Glossary <warm_start
>`.
431 n_jobs : int or None, optional (default=None)
432     The number of jobs to run in parallel for both `fit` and `
predict`.
433     ``None`` means 1 unless in a :obj:`joblib.parallel_backend
`
context.
434     ``-1`` means using all processors. See :term:`Glossary <
n_jobs>`
```

```
435     for more details.
436     random_state : int, RandomState instance or None, optional (
437     default=None)
438         If int, random_state is the seed used by the random number
439         generator;
440         If RandomState instance, random_state is the random number
441         generator;
442         If None, the random number generator is the RandomState
443         instance used
444         by `np.random`.
445     verbose : int, optional (default=0)
446         Controls the verbosity when fitting and predicting.
447     Attributes
448     -----
449     estimators_ : list of estimators
450         The collection of fitted sub-estimators.
451     estimators_samples_ : list of arrays
452         The subset of drawn samples (i.e., the in-bag samples) for
453         each base
454         estimator. Each subset is defined by an array of the
455         indices selected.
456     estimators_features_ : list of arrays
457         The subset of drawn features for each base estimator.
458     oob_score_ : float
459         Score of the training dataset obtained using an out-of-bag
460         estimate.
461     oob_prediction_ : array of shape = [n_samples]
462         Prediction computed with out-of-bag estimate on the
463         training
464         set. If n_estimators is small it might be possible that a
465         data point
466         was never left out during the bootstrap. In this case,
467         `oob_prediction_` might contain NaN.
468     References
469     -----
470     .. [1] L. Breiman, "Pasting small votes for classification in
471         large
472         databases and on-line", Machine Learning, 36(1),
473         85-103, 1999.
474     .. [2] L. Breiman, "Bagging predictors", Machine Learning,
475         24(2), 123-140,
476         1996.
477     .. [3] T. Ho, "The random subspace method for constructing
478         decision
479         forests", Pattern Analysis and Machine Intelligence,
480         20(8), 832-844,
481         1998.
482     .. [4] G. Louppe and P. Geurts, "Ensembles on Random Patches",
483         Machine
```

```
469         Learning and Knowledge Discovery in Databases, 346-361,  
2012.  
470     """  
471  
472     def __init__(self,  
473                 base_estimator=None,  
474                 n_estimators=10,  
475                 max_samples=1.0,  
476                 max_features=1.0,  
477                 bootstrap=True,  
478                 bootstrap_features=False,  
479                 oob_score=False,  
480                 warm_start=False,  
481                 n_jobs=1,  
482                 random_state=None,  
483                 verbose=0,  
484                 c=0):  
485         super(Weighted, self).__init__(  
486             base_estimator,  
487             n_estimators=n_estimators,  
488             max_samples=max_samples,  
489             max_features=max_features,  
490             bootstrap=bootstrap,  
491             bootstrap_features=bootstrap_features,  
492             oob_score=oob_score,  
493             warm_start=warm_start,  
494             n_jobs=n_jobs,  
495             random_state=random_state,  
496             verbose=verbose,  
497             c=c)  
498  
499     def predict(self, X):  
500         """Predict regression target for X.  
501         The predicted regression target of an input sample is  
502         computed as the  
503         mean predicted regression targets of the estimators in the  
504         ensemble.  
505         Parameters  
506         -----  
507         X : {array-like, sparse matrix} of shape = [n_samples,  
n_features]  
508             The training input samples. Sparse matrices are  
509             accepted only if  
510             they are supported by the base estimator.  
511         Returns  
512         -----  
513         y : array of shape = [n_samples]  
514             The predicted values.  
515         """
```

```
513     check_is_fitted(self, "estimators_features_")
514     # Check data
515     X = check_array(
516         X, accept_sparse=['csr', 'csc'], dtype=None,
517         force_all_finite=False
518     )
519
520     # Parallel loop
521     n_jobs, n_estimators, starts = _partition_estimators(self.
522 n_estimators,
523                                                         self.
524 n_jobs)
525
526     all_y_hat = Parallel(n_jobs=n_jobs, verbose=self.verbose)(
527         delayed(_parallel_predict_regression)(
528             self.estimators_[starts[i]:starts[i + 1]],
529             self.estimators_features_[starts[i]:starts[i +
530 1]],
531             X, self.weights_)
532         for i in range(n_jobs))
533
534     # Reduce
535     y_hat = sum(all_y_hat)
536     return y_hat
537
538 def _validate_estimator(self):
539     """Check the estimator and set the base_estimator_
540 attribute."""
541     super(Weighted, self)._validate_estimator(
542         default=DecisionTreeRegressor())
543
544 def _set_oob_score(self, X, y):
545     n_samples = y.shape[0]
546
547     predictions = np.zeros((n_samples,))
548     n_predictions = np.zeros((n_samples,))
549
550     for estimator, samples, features in zip(self.estimators_,
551 self.
552 estimators_samples_,
553                                                         self.
554 estimators_features_):
555         # Create mask for OOB samples
556         mask = ~indices_to_mask(samples, n_samples)
557
558         predictions[mask] += estimator.predict((X[mask, :])[:,
559 features])
560         n_predictions[mask] += 1
```

```
555         if (n_predictions == 0).any():
556             warn("Some inputs do not have OOB scores. "
557                 "This probably means too few estimators were used
558                 "to compute any reliable oob estimates.")
559             n_predictions[n_predictions == 0] = 1
560
561         predictions /= n_predictions
562
563         self.oob_prediction_ = predictions
564         self.oob_score_ = r2_score(y, predictions)
```

B.1.2 Experiment class

```
1 import glob
2 import errno
3 import pickle
4 import warnings
5 import multiprocessing
6
7 import numpy as np
8 from tqdm import tqdm
9 import matplotlib.pyplot as plt
10 from sklearn.ensemble import BaggingRegressor
11 from sklearn.metrics import mean_squared_error
12 from sklearn.tree import DecisionTreeRegressor
13 from sklearn.model_selection import GridSearchCV
14 from sklearn.exceptions import DataConversionWarning
15 from cvxopt import solvers
16
17 warnings.filterwarnings(action='ignore', category=
18     DataConversionWarning)
19 warnings.filterwarnings(action='ignore', category=
20     DeprecationWarning)
21 solvers.options['show_progress'] = False
22
23 from data import Data
24 from config import Config
25 from weighted_bagging import Weighted
26
27 config = Config()
28
29 def store_params(file_name, results):
30     """Function to store results
31     """
32     results['name'] = file_name
33     pickle_out = open('./data/results/{}_result.pkl'.format(
```



```

75     clf.fit(x, y)
76     return clf, {'parameters': clf.best_params_, 'score': clf.
best_score_}
77
78
79 def optimize_weighted(x, y, seed, x_val=None, y_val=None):
80     """Function to exhaustively search for best parameters of a
weighted
81     regression trees
82     """
83     parameters = {'n_estimators': config.n_estimators,
84                  'base_estimator__criterion': config.criterion,
85                  'base_estimator__splitter': config.splitter,
86                  'base_estimator__max_depth': config.max_depth,
87                  'c': config.c, 'random_state': [seed]}
88     clf = GridSearchCV(Weighted(DecisionTreeRegressor(random_state
=seed)),
89                       parameters,
90                       cv=config.cv_folds)
91     if x_val is not None and y_val is not None:
92         clf.fit(x, y, X_val=x_val, y_val=y_val)
93     else:
94         clf.fit(x, y)
95     return clf, {'parameters': clf.best_params_, 'score': clf.
best_score_}
96
97
98 def tree_experiment(name, validation):
99     """Function to run a number of experiments using a single
regression tree
100     """
101     results = {'loss': 0.0, 'max_depth': 0.0}
102     for i in tqdm(range(config.n_experiments)):
103         if validation:
104             x_train, x_test, _, y_train, y_test, _ = Data(name).
compile(
105                 validation=validation)
106         else:
107             x_train, x_test, y_train, y_test = Data(name).compile
()
108             clf, params = optimize_tree(x_train, y_train, i)
109             results['max_depth'] += \
110                 params['parameters']['max_depth'] / config.
n_experiments
111             results['loss'] += \
112                 mean_squared_error(y_test,
113                                     clf.predict(x_test)) / config.
n_experiments
114             store_params(name + "_tree", results)

```

```
115
116
117 def bagging_experiment(name, validation):
118     """Function to run a number of experiments using bagged
119     regression trees
120     """
121     results = {'loss': 0.0, 'max_depth': 0.0}
122     for i in tqdm(range(config.n_experiments)):
123         if validation:
124             x_train, x_test, _, y_train, y_test, _ = Data(name).
125             compile(
126                 validation=validation)
127         else:
128             x_train, x_test, y_train, y_test = Data(name).compile
129             ()
130             clf, params = optimize_bagging(x_train, y_train, i)
131             results['max_depth'] += \
132                 params['parameters']['base_estimator__max_depth'] /\
133                 config.n_experiments
134             results['loss'] += mean_squared_error(y_test,
135                                                 clf.predict(x_test))
136
137             / \
138                 config.n_experiments
139     store_params(name + "_bagging", results)
140
141
142 def weighted_experiment(name, validation):
143     """Function to run a number of experiments using weighted
144     regression trees
145     """
146     results = {'loss': 0.0, 'max_depth': 0.0}
147     for i in tqdm(range(config.n_experiments)):
148         if validation:
149             x_train, x_test, x_val, y_train, y_test, y_val = Data(
150             name).compile(
151                 validation=validation)
152             clf, params = optimize_weighted(x_train, y_train, i,
153             x_val, y_val)
154         else:
155             x_train, x_test, y_train, y_test = Data(name).compile
156             ()
157             clf, params = optimize_weighted(x_train, y_train, i)
158             results['max_depth'] += \
159                 params['parameters']['base_estimator__max_depth'] /\
160                 config.n_experiments
161             results['loss'] += mean_squared_error(y_test, clf.predict(
162             x_test)) / \
163                 config.n_experiments
164     store_params(name + "_weighted", results)
```

```
155
156
157 def experiment_loop(validation):
158     """Function to run a all experiments on all data sets
159     """
160     for dataset in config.datasets:
161         tree_experiment(dataset, validation)
162         bagging_experiment(dataset, validation)
163         weighted_experiment(dataset, validation)
164
165
166 def plot_mse(x, y, x_test, y_test):
167     mse = []
168     alphas = [round(x, 1) for x in np.arange(0.0, 1.0, 0.01)]
169     for c in tqdm(alphas):
170         clf = Weighted(DecisionTreeRegressor(max_depth=9),
171                       n_estimators=100, c=c, random_state=1)
172         clf.fit(x, y)
173         mse.append(mean_squared_error(y_test, clf.predict(x_test))
174 )
175     fig = plt.figure(figsize=(5, 5))
176     ax = fig.add_subplot(111)
177     ax.plot(alphas, mse, label='MSE', color='black')
178     ax.set(xlabel='Alpha', ylabel='Mean squared error (MSE)',
179           title='MSE as a function of the regularisation
180 parameter')
181     ax.axvline(x=alphas[np.argmin(mse)], color='r', linestyle='--'
182 )
183     ax.set_xticks(list(ax.get_xticks()) + alphas[np.argmin(mse)])
184     ax.grid()
185     ax.axis('tight')
186     ax.legend()
187     plt.show()
188     fig.savefig(str('mse_vs_alpha') + ".pdf")
189
190
191 def plot_weights(x: np.ndarray, y: np.ndarray):
192     n_models = 500
193     alphas = [round(x, 3) for x in np.arange(0.1, 1.1, 0.01)]
194     coefs = []
195     for c in alphas[::-1]:
196         clf = Weighted(n_estimators=n_models, c=c, random_state=1)
197         clf.fit(x, y)
198         coefs.append(clf.weights_.reshape(-1, ))
199     fig = plt.figure(figsize=(5, 5))
200     ax = fig.add_subplot(111)
201     ax.plot(alphas, coefs)
202     ax.axhline(y=1.0/n_models, color='black', linestyle='--')
203     ax.set(xlabel='Alpha', ylabel='Weights',
```

```

201         title='Weights as a function of the regularisation
parameter')
202     ax.axis('tight')
203     plt.show()
204     fig.savefig(str('weights_vs_alpha') + ".pdf")
205
206
207 def scatter(y_true, y_pred):
208     fig = plt.figure(figsize=(5, 5))
209     ax = fig.add_subplot(111)
210     ax.scatter(y_true, y_pred, c='red')
211     ax.set(xlabel='Alpha', ylabel='Weights',
212           title='True response against predicted response')
213     ax.axis('tight')
214     plt.show()
215     fig.savefig(str('response_vs_response') + ".pdf")

```

B.1.3 Main class

```

1 import argparse
2
3 from experiments import *
4
5
6 def main(run_experiment, validation, print_results):
7     if run_experiment:
8         experiment_loop(validation)
9     if print_results:
10        get_params()
11
12
13 if __name__ == "__main__":
14     parser = argparse.ArgumentParser(
15         description='Project')
16     parser.add_argument('--run_experiment', metavar='', type=bool,
17                         default=False, help='Run all experiments')
18     parser.add_argument('--validation', metavar='', type=bool,
19                         default=True, help='Use a validation set
20         to find '
21                         'optimal weights')
22     parser.add_argument('--print_results', metavar='', type=bool,
23                         default=False, help='Print experiment
24         results')
25     args = parser.parse_args()
26     main(args.run_experiment, args.validation, args.print_results)

```

B.1.4 Configuration class

```
1 import numpy as np
2
3
4 class Config:
5     def __init__(self):
6         self.datasets = ['friedman_1', 'friedman_2',
7                         'friedman_3', 'boston', 'ozone']
8
9         self.seed = None
10        self.n_experiments = 100
11        self.cv_folds = 10
12        self.n_estimators = [25]
13        self.criterion = ['mse']
14        self.splitter = ['best']
15        self.max_depth = range(1, 25)
16        self.signal_to_noise = 3
17        self.samples_simulated = 1200
18        self.n_simulated = 1000
19        self.n_boston = 25
20        self.n_ozone = 15
21        self.validation_size = 0.3
22        self.c = [round(x, 1) for x in np.arange(0, 1.1, 0.1)]
```

B.1.5 Data class

```
1 import pickle
2
3 import numpy as np
4 import pandas as pd
5 from sklearn.datasets import *
6 from sklearn.model_selection import train_test_split
7
8 from config import Config
9
10 config = Config()
11
12
13 class Data:
14     """This class is responsible for importing and returning the
15     data sets that
16     will form the basis for experimentation in this project. A
17     total of 5
18     data sets are represented, of which 3 are simulated. In
19     respect of the
20     simulated data sets we use the Sklearn (http://scikit-learn.org)
21     implementations of freidman_1, firedman_2 and friedman_3.
22     Furthermore, the
23     Boston data is also sourced from Sklearn. The final dataset,
```

```

Ozone, is
20 sourced from the UCI machine learning repository
21 (https://archive.ics.uci.edu/ml/datasets.html).
22 """
23 def __init__(self, name: str):
24     # the name of the dataset that the class needs to operate
    on
25     self.name = name
26
27 @staticmethod
28 def ozone():
29     """Load a locally store csv file containing the Ozone
    dataset.
30     """
31     df = pd.read_csv('data/ozone.csv')
32     return df.iloc[:, 1:], df.iloc[:, 0]
33
34 @staticmethod
35 def convert_to_pandas(x, y, col_names=None, default_names=
    False):
36     """ Function to convert a ndarray to a pandas dataframe.
37
38     Params
39     =====
40     x : (ndarray) predictor matrix of shape = [n_samples,
    n_features]
41     y : (ndarray) response of shape = [n_samples]
42     col_names : (optional string) names of predictors
43     default_names : boolean, optional (default=False) if
    specified column
44     names will be generated
45     Return
46     =====
47     x, y : the predictor and response matrices as pandas
    dataframes
48     """
49     n_features = x.shape[1]
50     if default_names:
51         col_names = ['x_{}'.format(str(i)) for i in range(
    n_features)] + \
52             ['response']
53     if col_names is not None:
54         if len(col_names) != (n_features + 1):
55             raise ValueError("A total of {} column names
    should be "
56                               "specified for x and y. Currently
    {} "
57                               "specified".format(n_features +
    1,

```

```
58                                     len(col_names)
59     ))
60     return pd.DataFrame(x, columns=col_names[:-1]), \
61            pd.DataFrame(y, columns=col_names[-1:])
62     return pd.DataFrame(x), pd.DataFrame(y)
63
64     def compile(self, write=False, to_pandas=False, validation=
65     False):
66         """ Compiles a particular data set and returns a train
67         test split
68
69         Params
70         =====
71         write : optional, (default=False) parameter specifying if
72         the data sets
73         should be saved to the local directory
74         to_pandas : optional, (default=False) if the returned data
75         sets should
76         be saved as pandas dataframes
77
78         Returns
79         =====
80         x_train, x_test, y_train, y_test split of compiled data
81         set
82         """
83         if self.name == 'friedman_1':
84             x, y = make_friedman1(n_samples=config.
85             samples_simulated,
86                                   n_features=10, random_state=1)
87             if to_pandas:
88                 x, y = self.convert_to_pandas(x, y, default_names=
89                 True)
90             x_train, x_test, y_train, y_test = \
91             train_test_split(x, y,
92                             test_size=config.n_simulated,
93                             random_state=1)
94             elif self.name == 'friedman_2':
95                 x, y = make_friedman2(n_samples=config.
96                 samples_simulated)
97                 noise = \
98                 np.random.normal(scale=np.std(y) / config.
99                 signal_to_noise,
100                                   size=y.shape)
101                 y += noise
102                 if to_pandas:
103                     x, y = self.convert_to_pandas(x, y, default_names=
104                     True)
105                 x_train, x_test, y_train, y_test = \
106                 train_test_split(x, y, test_size=config.
```

```
n_simulated)
95     elif self.name == 'friedman_3':
96         x, y = make_friedman3(n_samples=config.
samples_simulated)
97         noise = \
98             np.random.normal(scale=np.std(y) / config.
signal_to_noise,
99                                 size=y.shape)
100         y += noise
101         if to_pandas:
102             x, y = self.convert_to_pandas(x, y, default_names=
True)
103         x_train, x_test, y_train, y_test = \
104             train_test_split(x, y, test_size=config.
n_simulated)
105     elif self.name == 'boston':
106         data_dict = load_boston()
107         x, y = data_dict.data, data_dict.target
108         if to_pandas:
109             x, y = \
110                 self.convert_to_pandas(x, y,
111                                         col_names=list(
112                                             feature_names) +
113                                             ["MDEV"])
114             x_train, x_test, y_train, y_test \
115                 = train_test_split(x, y, test_size=config.n_boston
)
116     elif self.name == 'ozone':
117         x, y = self.ozone()
118         x_train, x_test, y_train, y_test = \
119             train_test_split(x, y, test_size=config.n_ozone)
120     else:
121         raise KeyError('Invalid data set type {}'.format(self.
name))
122     if write:
123         self.write(x_train, x_test, y_train, y_test)
124     if validation:
125         x_test, x_val, y_test, y_val = \
126             train_test_split(x_test, y_test,
127                             test_size=config.validation_size)
128         return x_train, x_test, x_val, y_train, y_test, y_val
129     else:
130         return x_train, x_test, y_train, y_test
131
132 def write(self, x_train, x_test, y_train, y_test):
133     """Write the data set to local directory
134     """
135     p_out = open('./data/train_test/{}'.format(self.name), 'wb
```

