A MACHINE LEARNING ENSEMBLE APPROACH TO TWITTER SENTIMENT CLASSIFICATION

Sebastian Dankfried Kirsch

A Dissertation submitted to
the School of Computing Sciences of The University of East Anglia
in partial fulfilment of the requirements for the degree of
MASTER OF SCIENCE.
AUGUST, 2017

SUPERVISOR(S), MARKERS/CHECKER AND ORGANISER

The undersigned hereby certify that the markers have independently marked the dissertation entitled "**A Machine Learning Ensemble Approach to Twitter Sentiment Classification**" by **Sebastian Dankfried Kirsch**, and the external examiner has checked the marking, in accordance with the marking criteria and the requirements for the degree of **Master of Science**.

Supervisor: _____
Dr. Wenjia Wang

Markers: _____
Marker 1: Dr. Wenjia Wang

_____
Marker 2: Dr. Jason Lines

External Examiner: _____
Checker/Moderator

Moderator: _____
Dr. Wenjia Wang

DISSERTATION INFORMATION AND STATEMENT

Dissertation Submission Date: **August, 2017**

Student:	**Sebastian Dankfried Kirsch**
Title:	**A Machine Learning Ensemble Approach to Twitter Sentiment Classification**
School:	**Computing Sciences**
Course:	**Computing Science**
Degree:	**M.Sc.**
Duration:	**2016-2017**
Organiser:	**Dr. Wenjia Wang**

STATEMENT:

Unless otherwise noted or referenced in the text, the work described in this dissertation is, to the best of my knowledge and belief, my own work. It has not been submitted, either in whole or in part for any degree at this or any other academic or professional institution.

Permission is herewith granted to The University of East Anglia to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

_____

Signature of Student

Abstract

The work at hand investigates the application of machine learning ensemble techniques to classify tweets by sentiments they express. Millions of people around the globe use social media to express their opinions on topics ranging from politics to sports. Analyzing these sentiments readily available online presents private and public sector institutions with significant opportunities to optimize their business operations and policies by better understanding their customers and constituents. Machine learning methods hold great potential for analyzing data at volumes and speeds that are beyond the capacity of any human analyst. However, navigating the complexities of human language presents considerable difficulties to even the best learning algorithms. Ensemble methods are an attempt to deal with complex machine learning problems by combining the predictions generated by multiple algorithms akin to a group of human experts trying to reach a consensus on a difficult problem. The aim of this work is to investigate the performance of ensemble methods on the problem of classifying tweets by sentiment.

The work makes use of publicly available datasets of tweets already classified by sentiment as well as tweets collected specifically for the purpose of this research. Chapters 3 to 5 describe how the data has been cleaned and processed into matrix format, how additional semantic features have been added and how feature selection has been performed. Chapters 6 and 7 contain the selection of suitable algorithms for ensemble learning and some experiments with established ensemble strategies. In Chapter 8 novel ensemble strategies "Hybrid Boost" and "Bayesian Hybrid Boost" are proposed and their implementation is described.

Results of the experiments conducted demonstrate that the newly developed ensemble strategies outperform all tested single algorithms as well as bagging and stacking by a considerable margin resulting in up to 7% lower classification error.

Acknowledgements

Primarily, I would like to thank my supervisor Dr. Wenjia Wang. He has been an invaluable guide in writing this work and conducting the underlying research. His enthusiasm and encouragement have sustained my motivation to tackle the difficult and arduous task of developing my own novel solutions rather than reproducing existing research. His many suggestions made it possible to turn these ideas into a coherent piece of academic work.

Furthermore, I would like to extend my gratitude to my parents, whose unwavering moral and financial support made it possible to embark on the endeavour of obtaining an advanced degree in computing science.

Finally, I want to thank my girlfriend for her patience and acceptance of many late-night and weekend hours at the computer laboratory as well as her support in the form of moral encouragement and many delicious and healthy dishes.

Sebastian Kirsch

at Norwich, UK.

Table of Contents

Abstract iv

Acknowledgements v

Table of Contents vi

List of Tables ix

List of Figures x

List of Abbreviations xii

1 Introduction 1
- 1.1 Aims and Objectives . . . 1
- 1.2 Background and Motivation . . . 2
- 1.3 Research Questions . . . 3

2 Literature Review 4
- 2.1 Sentiment Features . . . 4
- 2.2 Sentiment Classification on Twitter . . . 5
- 2.3 Ensemble Learning and Sentiment Classification . . . 6

3 Methodology 7
- 3.1 Design of System Framework . . . 7
- 3.2 Design of Experiments . . . 9
- 3.3 Methods and Metrics for Evaluation . . . 13
  - 3.3.1 Terminology . . . 14
- 3.4 Tools and Practical Implementation . . . 14
  - 3.4.1 Tools . . . 14
  - 3.4.2 Implementation of Experiments . . . 14

4 Data Collection and Preprocessing 17
- 4.1 Datasets . . . 17
- 4.2 Cleaning . . . 17
- 4.3 Data Transformation . . . 18
  - 4.3.1 From Text to Vector Format . . . 18
  - 4.3.2 Data Reduction and Sparsity . . . 18
  - 4.3.3 Text Processing . . . 19

5 Feature Engineering **20**
5.1 Feature Construction . . . 20
5.1.1 Semantic Features . . . 20
5.1.2 Topical Features . . . 21
5.1.3 Evaluation of Manually Constructed Features . . . 22
5.2 Feature Selection . . . 24
5.2.1 Model Based Selection . . . 24
5.2.2 Feature Correlation . . . 26
5.2.3 Recursive Selection . . . 26

6 Single Algorithms and Model Tuning **28**
6.1 Performance Evaluation Metrics . . . 28
6.2 Classifiers . . . 29
6.2.1 Decision Trees . . . 29
6.2.2 Support Vector Machines . . . 31
6.2.3 Random Forests . . . 33
6.2.4 Naive Bayes . . . 35
6.2.5 Neural Networks . . . 36
6.2.6 Bagged Trees . . . 37
6.2.7 Generalized Boosting Machines . . . 38
6.2.8 Penalized Linear Models . . . 39
6.3 Final Model Selection . . . 43

7 Established Ensemble Strategies **44**
7.1 Simple Majority Voting . . . 44
7.2 Bagging . . . 44
7.2.1 Implementation . . . 44
7.2.2 Experiments and Results . . . 45
7.3 Stacking . . . 45
7.3.1 Implementation . . . 46
7.3.2 Experiments and Results . . . 47

8 New Ensemble Strategies **48**
8.1 A Hybrid Approach to Boosting . . . 48
8.1.1 Adaboost: Background and Problems . . . 48
8.1.2 Hybrid Boost: A Novel Approach to Boosting . . . 49
8.1.3 Experiments and Results . . . 52
8.2 Bayesian Ensemble Approaches . . . 54
8.2.1 Background: Bayesian Averaging and Model Combination . . . 55
8.2.2 Bayesian Hybrid Boost . . . 56
8.2.3 Experiments and Results . . . 58

9 Evaluation and Discussion **60**
9.1 Evaluation of Results . . . 60
9.1.1 Evaluation of Model Differences . . . 61
9.1.2 Evaluation of Single Models . . . 62
9.1.3 Evaluation of Ensemble Strategies . . . 64
9.1.4 Performance on Different Data . . . 66
9.2 Discussion of Results . . . 67

10 Conclusion **70**
10.1 Conclusion 70
10.2 Suggestion for Further Work 71

Bibliography **72**

List of Tables

6.1 Decision Tree performance over different features on the test set 29
6.2 Linear kernel SVM performance over different features on the test set . 32
6.3 Random forest performance summary over different features on the test set 33
6.4 Naive Bayes performance over different features on the test set 35
6.5 Neural Network performance over different features on the test set . . . 36
6.6 Tree bagging performance over different features on the test set 37
6.7 Tree boosting performance over different features on the test set 38
6.8 Penalized Discriminant Analysis performance over different features on the test set 40
6.9 Elastic Net performance over different features on the test set 41

List of Figures

3.1 Model Building: Systemic Framework 7
3.2 Model Building: Process Overview 9
3.3 Structure of R files 15

5.1 LDA: Log Likelihood for different topics 22
5.2 Change in accuracy by feature and model 23
5.3 Changes in sensitivity and specificity by feature and model 23
5.4 Change in accuracy by number of features 25

6.1 C5.0 tree and rule model performance with different numbers of trees and boosting iterations on the cross validated training set 30
6.2 C5.0 ROC Curve 31
6.3 SVM Performance with different cost parameters on the test set 32
6.4 ROC curve of the linear SVM model with cost one 32
6.5 Random forest performance with different numbers of trees on the test set 33
6.6 Change in Random Forest performance on the cross validated training set depending on the number of variables randomly selected for each split 34
6.7 Random Forest ROC curve 34
6.8 Change in performance of feed forward neural nets on the cross validated training set depending on the number of hidden layers and the rate of weight decay 36
6.9 Neural net ROC curve 37
6.10 Change in performance of general boosting machines on the cross validated training set depending on the number of boosting iterations . . . 39
6.11 GBM ROC 39
6.12 Change in performance of Penalized Discriminant Analysis on the cross validated training set depending on shrinkage penalty factor 40
6.13 ROC curve of a Penalized Discriminant model with the best determined tuning parameters on the 127 feature subset 41

6.14 Change in performance of Elastic Net models depending the regularization factor and the mixing percentage between the Lasso and Ridge Regression 42
6.15 Elastic Net ROC curve using the best determined tuning parameters and the 127 feature subset 42

7.1 Model performance across multiple Boosting iterations 45
7.2 Model performance using different level 1 classifiers 47

8.1 Model performance across multiple Boosting iterations 53
8.2 Classification accuracies achieved by various boosting strategies 54
8.3 Summary of the performance of new ensembles 59

9.1 Change in accuracy by feature and model 61
9.2 Number of Observations classified differently per model pair 62
9.3 Distribution of base accuracies of default models fitted on the simple bag of words dataset (left) and on the feature enhanced dataset (right) for 10 different training/test splits 63
9.4 Distribution of single tuned algorithm accuracies on the feature enhanced data set using 10 different training/test splits 63
9.5 Distribution bagged model accuracies on the feature enhanced data set using 10 different training/test splits 64
9.6 Distribution of stacking accuracies on the feature enhanced data set using 10 different training/test splits 65
9.7 Distribution of accuracies obtained through different ensemble strategies on the feature enhanced data set using 10 different training/test splits . 65
9.8 Distribution of accuracies obtained through different ensemble strategies on the feature enhanced data set using 10 different training/test splits . 67

Chapter 1

Introduction

Sentiment analysis is concerned with determining opinions expressed by people in various forums and contexts especially, but not only, on the Internet. The field is closely related to Natural Language Processing (Liu, 2012) and can be greatly enhanced by machine learning. Machine learning comprises of many different algorithms, which have their individual strengths and weaknesses. Ensemble methods aim to combine multiple machine learning algorithms to magnify the strengths and counterbalance the weaknesses of individual learners. In the pursuit of this goal ensemble strategies ranging from trivial to arcane have been developed and often proved to achieve performance superior to that of single algorithms (Dietterich et al., 2000).

1.1 Aims and Objectives

The overall aim of this dissertation is to investigate how ensemble methods can be applied to the problem of sentiment classification and what performance they can achieve using tweets obtained from Twitter. To achieve this goal it begins with a survey of the existing literature on sentiment classification and ensemble machine learning to establish an understanding of the field and find benchmarks for evaluating my own work. Secondly, it outlines my methodology explaining what tools and resources I have used and how I designed experiments, illustrates the research process, and finally, describes methods of evaluation. The next three chapters walk the reader through the steps of the machine learning process I have undertaken such as data selection, data cleaning and preprocessing, feature engineering, as well as the choice and tuning of machine learning algorithms. The objective of the processes outlined in these chapters is to prepare the data and the models for rigorously investigating and testing ensemble

strategies. Chapter 7 describes some experiments I have undertaken with existing ensemble methods. Chapter 8 is the core chapter of this dissertation as it describes 2 novel ensemble strategies, hybrid boost and Bayesian hybrid boost, which I have developed based on AdaBoost and Bayesian model combination. In these two chapters, I attempt to provide an answer to my research questions. In Chapter 9 I present an evaluation of my work as well as a discussion against the light of other research in the field. Finally, the work ends with a conclusion and suggestions for future work.

1.2 Background and Motivation

Social Media is used by millions of people around the globe to express opinions on topics ranging from politics to sports. Organizations in the private and public sector can benefit enormously from analyzing social media data. For example, a better understanding of opinions held in the population can help businesses develop more targeted marketing strategies or government agencies to formulate more successful policies. Manual analysis is usually not feasible due to the large volume of the data. Other options such as lexicon methods, which simply classify tweets by the occurrence of words that are connected to certain sentiments, are often not sufficient to deal with the intricacies presented by human language Machine learning methods can analyze much larger quantities of data than human analysts while having a much greater capacity to identify features in human language than lexicon methods. The versatility and strength of machine learning mainly stem from a large toolkit of algorithms which have their individual strengths and weaknesses. Ensemble learning aims to combine the strengths and versatility of multiple algorithms by attempting to combine the predictions those different algorithms reach on the same data. Ensemble learning is thus, akin to a group of experts, who jointly attempt to solve a difficult problem. The collectively attained solution may often be superior to individual ones as the incorporation of different viewpoints presumably helps to minimize individual biases. The challenges presented by dissecting human language are often complex and intricate. Accordingly, it appears natural that collectively reached solutions to language problems stand a good chance of outperforming individual ones. Still, existing research in the application of ensemble

machine learning to language problems is not extensive. My motivation in composing this work and conducting the research that underlies it is to expand the current body of knowledge with an investigation of the possibilities presented by established ensemble methods and by attempting to develop novel ensemble strategies for the problem of social media sentiment classification.

1.3 Research Questions

The following research questions have guided my research:

1. How do ensemble methods perform on the problem of Twitter sentiment classification?

2. What potential exists for improving the performance ensemble methods achieve on the problem of Twitter sentiment classification?

3. Can existing methods be adapted or new ensemble methods be developed to improve classification accuracy on the problem of Twitter sentiment classification?

Chapter 2

Literature Review

Given the breadth of the field, a solid body of literature already exists on various aspects of sentiment analysis through machine learning. Pang et al. (2008) and Liu (2012) provide a general overview of the field of sentiment analysis including the various approaches, stages of classification, common algorithms and methods as well as linguistic and semantic challenges.

2.1 Sentiment Features

As with other machine learning challenges, strong features are vital to improving model performance in the field of supervised sentiment classification. This section discusses work which has inspired my feature construction and selection efforts. Hatzivassiloglou and Wiebe (2000) have addressed the problem of identifying subjectivity in a text by studying the role of adjectives and their gradability. They concluded that adjectives, especially in conjunction with words such as "and", "or", "but", are strong indicators of sentiment orientation. According to the findings of Pak and Paroubek (2010) subjective tweets generally have higher proportions of adjectives and personal pronouns while objective ones tend to contain more common and proper nouns. Wilson et al. (2005) addressed context dependent subjectivity and the challenge of extracting opinion polarity from phrases and sentences. Ding et al. (2008) used a lexicon based approach and relied on the distance between opinion words and product features to determine the orientation of product reviews that contain conflicting sentiment words. Wang et al. (2010) on the other hand used a regression approach to determine not only sentiment expressed in reviews but also the relative importance each reviewer places on different

aspects that lead to the expressed sentiment. Furthermore Ding and Liu (2010) have studied coreference resolution and the challenge of dealing with sentences that do not contain clearly defined objects. Kim and Hovy (2006) developed a method to extract topics, opinions, and opinion holders from online news media by identifying opinion bearing words and assigning semantic roles. Furthermore Saif et al. (2012) have found that topical features are much stronger determinants of sentiment than semantic ones. The most commonly used method for extracting topics from multiple documents is Latent Dirichlet Allocation (Blei et al., 2003) which will be discussed in more detail in section 5.1.2.

2.2 Sentiment Classification on Twitter

Others including Davidov et al. (2010) have performed sentiment analysis on Twitter with Bollen et al. (2011) going as far as demonstrating that sentiments expressed on Twitter can predict stock market movements. (Pak and Paroubek, 2010) introduced a method for building a corpus of tagged tweets without manual effort by collecting posts that included emoticons. Depending on the included emoticon, the tweet was classified as either positive, negative, or neutral. The use of emoticons had the further advantage of being language independent while providing a much clearer distinction between positive and negative than pure text data. (Pak and Paroubek, 2010) also identified some features for distinguishing facts from opinion. According to their findings, objective tweets make higher use of common and proper nouns, while subjective texts use more personal pronouns. Statements of facts tend to contain more past participles and third person verbs whereas opinions are often written in simple past tense making use of second person verbs. (Go et al., 2009a) also worked with Tweets containing emoticons. Using Support Vector Machines, Naive Bayes and Maximum Entropy they achieved over 80% classification accuracy.

2.3 Ensemble Learning and Sentiment Classification

Not much work has been done in the application of machine learning ensemble methods to sentiment classification. Wang et al. (2014) have constructed ensembles consisting of Naive Bayes, Maximum Entropy, Decision Tree, Support Vector Machines and K-Nearest Neighbors algorithms and have applied them to 10 public sentiment analysis datasets. Their empirical results show that ensemble methods on average perform better than single learners. Wilson et al. (2006) achieved an improvement of 23 - 96% in accuracy over single learners by using the ensemble technique boosting. Whitehead and Yaeger (2010) compared bagging, boosting, and random subspace to the performance of Support Vector Machines. They found that bagging performs at least equally well while boosting and random subspace often performed significantly better.

Chapter 3

Methodology

3.1 Design of System Framework

The systemic framework for implementing experiments is illustrated in figure 3.1 The

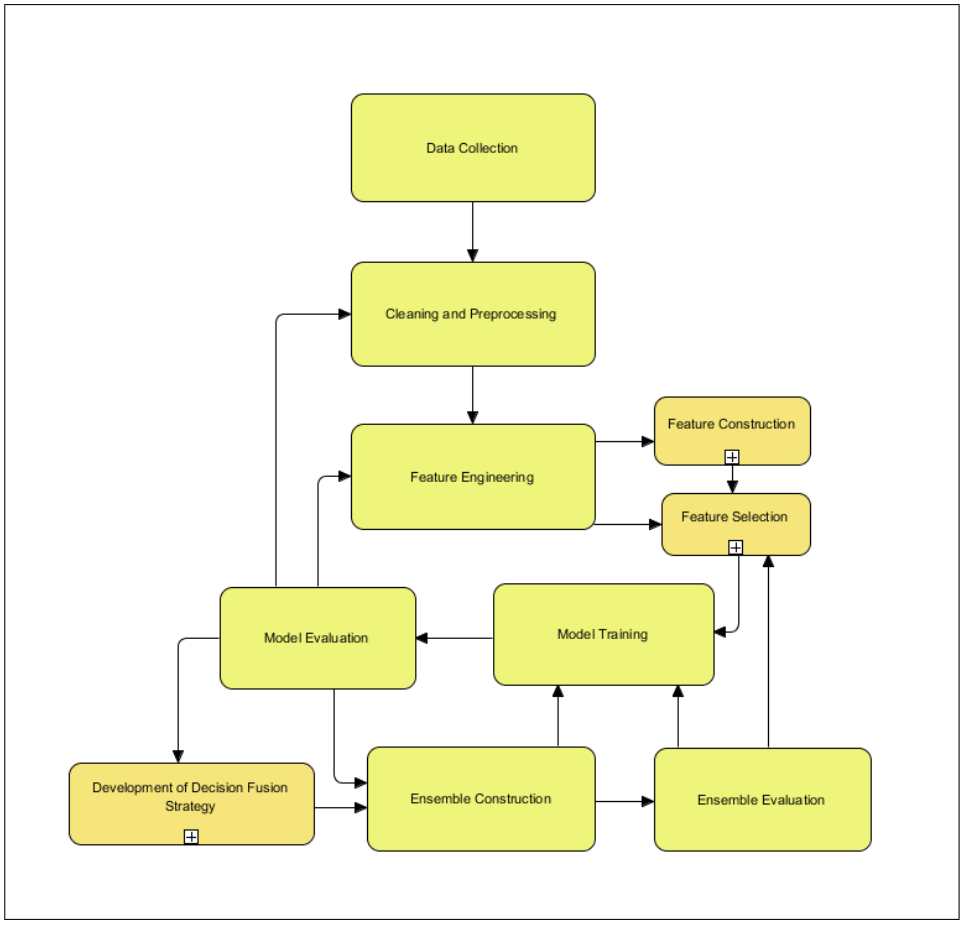

Figure 3.1: Model Building: Systemic Framework

first step consists of acquiring data suitable for model construction. This stage involves obtaining prelabelled sets of Tweets and labelling Tweets manually. Secondly, the data sets need to be cleaned and preprocessed. Datasets are initially brought into a uniform

format consisting of .csv files with two columns. The first column contains the tweets while the second one contains sentiment labels. All features of the data that have no value for the task of sentiment classification such as punctuation and stopwords are removed. Then, the tweets are transformed into a vector format. A document term matrix is created consisting of documents as rows while terms, consisting of between one and three words, that occur throughout the documents constitute columns. Once more the data is cleaned by removing sparse terms. The next stage, feature engineering, consists of two substages. At the feature construction stage features that may boost model performance are created and added to the matrix. Then, features that do not contribute significantly to the classification of sentiment, are identified and removed. Methods for identifying redundant features that do not require fitting a model to the data, such as correlation measurements, exist. They are commonly known as filter methods. Yet, they should not be solely relied upon as feature selectors as they consider predictors in isolation do not deal with issues of collinearity (Guyon and Elisseeff, 2003). Therefore, fitting models on various feature subsets is vital for evaluating predictor importance and finding appropriate feature combinations. Thus, the feature selection stage transitions directly into the model training and evaluation stage. Depending on model performance the process requires iterating back to the previous stages to perform further preprocessing and feature engineering. This procedure may be repeated several times and requires a large amount of experimental work such as testing the impact on the performance of different feature combinations and different preprocessing steps. Once possible performance improvements through feature selection and preprocessing have been sufficiently explored, the ensemble construction stage is entered. At this stage, it should be clear how various model types perform on different subsets and features of the data. Based on this knowledge classifiers for the ensemble are selected along with the data and feature sets on which they are trained. Iterations back to the single model training and evaluation stages are performed to identify adequate candidates among models for ensemble construction. Furthermore, a strategy for fusing the classifications generated by individual models is developed. Model selection and strategy development need to be performed with regard to increasing ensemble performance on metrics such as diversity and complexity.

Figure 3.2 illustrates how the software resources are used to transform the data.

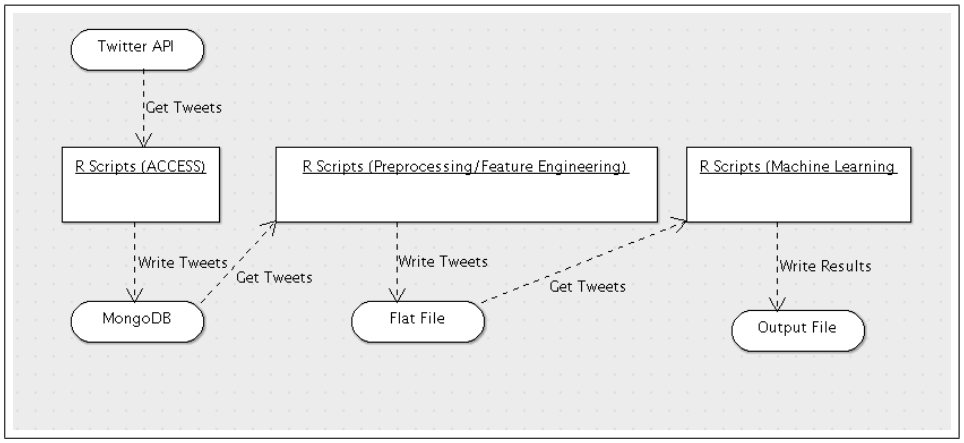

Figure 3.2: Model Building: Process Overview

3.2 Design of Experiments

The primary research question this dissertation aims to answer are:

1. How can ensemble techniques be applied to the problem of Twitter sentiment classification?

2. Can ensemble techniques be found that significantly outperform single learners on the problem of Twitter sentiment classification?

To address these question experiments are designed in consideration of the following requirements:

What kind of data will the ensemble be dealing with? The input consists of tweets thus, the ensemble has to deal with text data. This includes lexical features in the form of single or multi- word expressions, grammatical features such as parts of speech tags, topical features that allocate tweets to certain topics, and simple numeric features such as the length of tweets or single words. Furthermore, text data is characterized by high dimensionality. For experimental design, this implies that learning algorithms need to be selected based on how they respond to high dimensional data, with features of diverse types to which statistical techniques such as normalization are usually not applicable,

What kinds of tasks does the ensemble have to perform? The primary goal is the classification of tweets into one of two sentiment categories. For experimental design, this implies that classifiers rather than regressors or predictors are primarily selected as learning algorithms.

How to determine the appropriate size of the data set? To avoid unnecessarily wasting time and computing resources, the size of the data set is reduced as far as possible as long as it does not impact classification accuracy negatively. The size at which accuracy starts to decline is determined experimentally by iteratively fitting various learners on the training data and classifying the test data, both of which are shrunk by a factor of 10 with every iteration. However, even if no decline in accuracy is observed, the size of the dataset is not reduced to below 2000 observations as small samples increase the vacillation of results making them less reliable.

How to balance and partition the data? The data used has not been pre-partitioned. For all experiments a random split into a training and a test set using a 60:40 ratio is performed. I did not attempt to use other ratios for the splits as it would have taken too much time from the main focus of this work. During the training process, I further use 10-fold cross validation on the training set to achieve more reliable and stable results.

How to measure the performance of classifiers? Contrary to other binary classification problems such as breast cancer screening, where false negatives are costlier than false positives, the problem at hand does not require obtaining better predictions on one class than another. The goal is to achieve an improvement in overall classification accuracy. Therefore, the primary indicator of performance is classification accuracy. Sensitivity and specificity are used as further indicators to detect imbalances in the class predictions and to optimize ensemble diversity. For example, if the majority of models in an ensemble has higher sensitivity than specificity, adding models with higher specificity rates could increase ensemble diversity

How to select classifiers for the ensemble? Classifiers are primarily selected based

on how they perform on the data at hand and whether they complement existing classifiers in attaining higher ensemble diversity and classification accuracy.

How to determine appropriate feature subsets for each classifier? Given the high dimensionality of the data, exhaustively evaluating feature combinations to determine the optimal feature subset for every classifier is not a feasible option. Instead, filter methods such as correlation analysis are applied to filter out features that mainly add noise as determined by statistical metrics. Furthermore, wrapper methods and algorithms that rank features by importance such as random forests are used to extract feature subsets and gain an understanding which features are critical to improving model performance. The obtained information is used to derive heuristics that are applied to determine 3 feature subsets of different sizes. Each algorithm is then trained on the feature subset on which it achieves the highest classification accuracy. As models have been obtained from different feature subsets, it is also likely to increase ensemble diversity.

How to determine appropriate sampling strategies for each classifier? During the pre-processing stage, it is ensured that data in the training and test sets are roughly balanced to contain an equal number of observations of each class. Sensitivity and specificity produced by each classifier on the test data are used to determine imbalances in the class predictions. If the majority of classifiers in the ensemble performs significantly better on observations of one class, then a subset of classifiers may be retrained on a modified training set that contains more samples of the underpredicted class. Randomly selected samples of the underpredicted class are replicated until at least one classifiers attains a rough balance in class predictions.

How to determine hyperparameters for each classifier? Hyperparameters are systematically tuned using grid search over a range of possible hyperparameter values. Every model that allows hyperparameter tuning is refit 10 times using 10-fold cross validation. In every iteration, the tuning parameter value is changed using intervals appropriate for the respective parameter. For example, the weight decay in a neural network is increased in increments ranging from 0 to 0.1. The model

achieving the best area under the ROC curve value is selected.

How many classifiers to include? Classifiers are added to the ensemble as long as adding more models improves ensemble performance. However, limitations in time and computing power curtail the amount of single classifiers that can be trained and evaluated. Furthermore, some ensemble strategies may involve an exhaustive combination of outputs of all classifiers in the ensemble. Since the space of possible model combinations grows exponentially with every additional model, a high number of single classifiers is not feasible. Some ensemble methods like random forests may generate hundreds of single tree learners. However, in this work, the focus is on manually assembling hybrid ensembles of different types of algorithms. For these reasons, the number of models combined in an ensemble will not exceed 15 base learners.

How to select an ensemble strategy? In addition to previously addressed steps to improve single model performance, ensemble strategies need to be selected with the goal of further increasing overall classification accuracy. Whether an ensemble strategy is suitable depends on whether it improves overall prediction accuracy compared to single learners, how diverse predictions of the base learners are, and how well the ensemble generalizes to unseen data. The first approach to finding an appropriate ensemble consists of systematically applying established ensemble methods such as Bagging, Boosting and Stacking to the problem at hand. However, merely applying existing methods reduces the likelihood of achieving performance superior to single classifiers and restricts the room for innovation. Accordingly, I may attempt to develop new ensemble strategies or alter existing ones to be better suited for the research problem at hand.

3.3 Methods and Metrics for Evaluation

The evaluation of results requires

1. Suitable metrics
2. Statistically valid procedures and tests to determine the significance in differences between classifiers and the validity of the obtained results.
3. A benchmark against which to compare results

As explained in section 3.2 classification accuracy, sensitivity and specificity are the main metrics used for performance evaluation. Since statistical assumptions such as the normal distribution of observations do not apply to some variables, only nonparametric tests are considered for evaluating differences between classifiers. I ultimately chose to use the McNemar test to assess between-classifier difference for the following reasons:

1. The problem at hand is a binary classification task.
2. Since outputs are obtained on the same dataset they can be assumed to be related
3. We want to evaluate whether the responses of two classifiers to the same problem are consistent or not. If they are inconsistent, the predictions are significantly different according to the McNemar test.

To further evaluate the total difference between classifier outputs, I also calculated the number of samples classified differently between each model pair. Since experimental outputs are obtained on a random training/test split, results may partly be due to chance. To limit the impact of randomness on the classification output, I verify the results generated by each single learner and ensemble classifiers, by repeating the process of model training 10 times, every time partitioning the data into random training/test splits using a proportion of 60/40. The mean of each classifiers predictions across the 10 iterations is obtained as the final value. The system attempts to improve performance through the following stages

1. Data Preprocessing
2. Feature Engineering

3. Hyperparameter Tuning

4. Ensemble Learning

To evaluate relative progress, results at each stage are compared to the results obtained at the previous one starting with simple untuned models fitted on the bag of words models and ending with outputs generated by advanced ensembles on feature enhanced data sets. The final outputs are further measured against the results of comparable work such as Saif et al. (2012) who used the same data, or Wang et al. (2014) who applied ensemble learning to the problem of sentiment classification.

3.3.1 Terminology

In medical statistics, sensitivity describes the correct identification of positive cases, while specificity refers to the correct identification of negative cases. In this work the negative sentiment cases are the reference class thus sensitivity in this context describes the correct identification of negative sentiments whereas specificity refers to the correct identification of positive sentiments.

3.4 Tools and Practical Implementation

3.4.1 Tools

For obtaining tweets from the Twitter API I wrote programs in R and Java and stored them in the NoSQL database MongoDB. Data cleaning and preprocessing, feature engineering, the machine learning tasks, as well as statistical evaluation have been performed in R.

3.4.2 Implementation of Experiments

Implementation of experiments in R consisted of 4 stages.

1. Data Collection

2. Data Cleaning and Preprocessing

3. Data Transformation

4. Feature Engineering

5. Machine Learning

For some of the tasks R packages existed while for others, especially those specific to my problem, I had to write my own functions. I decided to divide my files into a "function" level and an "execution" level as illustrated in 3.3. The "function" level

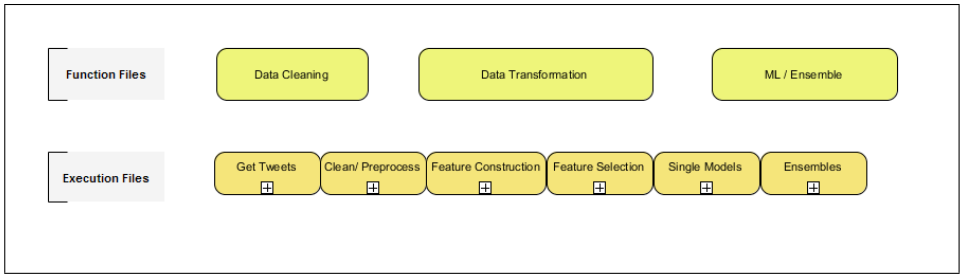

Figure 3.3: Structure of R files

contains all my custom functions while the execution level implements the experiments calling functions from packages or the function level.

For obtaining data I used the TwitteR package (Gentry, 2015) which provides functions for accessing the Twitter API and downloading tweets into the R environment by date, topic, language and geographic region. To clean and pre-process data I have written several functions that iterate over the raw tweets and remove textual features such as punctuation marks or hyperlinks. Some of these functions made use of the "Stringr" (Wickham, 2015b), and "plyr" (Wickham, 2011) packages for cleaning and processing raw text while "readxl" (Wickham, 2015a) was used for writing to and reading from .xl type files. The functions were called from a file at the execution level to bring raw tweets into a flat file format. To transform the data I created another function file that made use of "TM" (Feinerer and Hornik, 2015), "slam" (Hornik et al., 2014) "SnowballC" (Bouchet-Valat, 2014) and "RTextTools" (Jurka et al., 2014) to convert tweets stored in .csv, .xls and .txt files into a text corpus format and further into a document-term matrix format. Then, I wrote several functions to perform preprocessing on the text vectors such as removing sparse terms, tokenization, and parts of speech tagging which was achieved with the help of the "NLP" (Hornik, 2015a),

openNLP (Hornik, 2015b), and "RWeka" (Hornik et al., 2009) packages. Again, these functions were called from an execution level file to implement the data transformation. At the feature engineering stage, Latent Dirichlet Allocation was performed with the support of the (Grün and Hornik, 2011) package and unsupervised k-means clustering and hierarchical clustering were performed with the "cluster" (Maechler et al., 2014) package. This essentially required looping over the tweets and applying functions from aforementioned packages which I implemented in 2 files at the execution level. Therefore, no custom file at the function level was necessary at this stage. Model fitting, hyperparameter tuning and prediction were performed across 4 execution level files using the "caret" (from Jed Wing et al., 2015), "RTextTools" (Jurka et al., 2014), "e1071" (Meyer et al., 2014) , "kernlab" (Karatzoglou et al., 2004) and "randomForest" (Liaw and Wiener, 2002) packages.

At the ensemble construction stage I conducted some initial experiments with the caretEnsemble (Mayer and Knowles, 2015) package and ultimately used it to implement stacking. Since ensemble construction was the main focus of this dissertation I wrote several functions for combining the predictions of classifiers via majority voting, weighted majority voting, generating all possible sub-combinations of predictors and performing exhaustive subset search. For Bagging and my custom ensemble methods Hybrid Boost and Bayesian Hybrid Boost I wrote my own implementations as R functions which I then called from the execution level to conduct my experiments. Lastly, I created 3 evaluation files that call all function files and several execution scripts to automate the whole process from reading files to ensemble construction. These files also allow the user to execute the process in a loop with different random data splits in each iteration.

For graphics and plotting the "ggplot2" (Wickham, 2009) and "pROC" (Robin et al., 2011) packages were used.

Chapter 4

Data Collection and Preprocessing

4.1 Datasets

One of the biggest challenges in the initial stages of the dissertation was obtaining enough labeled tweets. The difficulty was compounded by the goal of performing sentiment analysis on data in multiple languages. Due to time constraints manually labeling a sufficient number of Tweets in multiple languages was not a realistic option. I finally decided for using a pre-labeled data set compiled by the NLP group at Stanford University for English data (Go et al., 2009b) and another one compiled by researchers at Cairo University for Arabic (Nabil et al., 2015a). Furthermore, I annotated 2000 German tweets manually and hired native Mandarin speakers to annotate another 10000 Weibo (the Chinese equivalent to Twitter) entries by sentiment. I brought all datasets in uniform formats as csv files with 2 columns. The first column contained Tweets while the second contained sentiment values ranging from 1 to 2 (negative, positive).

4.2 Cleaning

To minimize the potential for distorting the output at the modeling stage all features that were not indicative of sentiment needed to be removed. These included:

- Hyperlinks
- Usernames
- Punctuation

- Features peculiar to tweets such as "RT" or "@".

This was achieved by writing a function in R that iterated over all tweets and removed the aforementioned features. Furthermore, I removed tweets that consisted of less than two words since expressing an opinion as well and the subject of that opinion requires at least two words.

4.3 Data Transformation

4.3.1 From Text to Vector Format

For building a predictive model the data needs to be transformed into a format where words and fragments of text can be treated as features. This is achieved by creating a term document matrix the rows of which constitute tweets and the columns words or groups of words occurring throughout the tweets. Each column contains the number of times that particular word occurred in a given tweet. In R I implemented the transformation by using the custom functions in the "RTextTools" package.

4.3.2 Data Reduction and Sparsity

As every word is treated as a column, transforming a collection of Tweets into a term document matrix initially results in an extremely high dimensional dataset. For example transforming a 10.000 Tweet subset of the Stanford dataset resulted in a matrix of 9564 dimensions. to reduce computing time I tested the performance of support vector machines, random forests and decision trees on subsets of 200000, 20000 and 2000 tweets of the Stanford dataset. No decrease in accuracy was detected due to shrinking the dataset. Thus, I decided to use the smallest set of 2000 tweets. Dimensionality reduction was achieved by removing stop words such as "is", "or", "and" which occur in high frequency and usually do not convey opinions. Furthermore, I removed sparse term occurring with a frequency below 0.1% across the corpus which reduced dimensionality by almost 90%.

4.3.3 Text Processing

Tokenization refers to the process of dividing a text into smaller linguistic units such as words or groups of words. Single words were already extracted from tweets in the process of creating the document term matrix. From the perspective of sentiment analysis, units consisting of two (bigrams), three (trigrams) or more words such as frequent expressions may be of interest. Wilson et al. (2005) determined that using groups of more than three words does not yield significant improvements for predicting sentiment orientation. Accordingly, I used the tokenizer function from the RWeka package to extract all bigrams and trigrams that occurred with a frequency above 0.1% throughout the text. Parts of Speech Tagging is the process of identifying the grammatical roles of words in a text such as adjectives, nouns, or verbs. The RTextTools package implements the Maxent POS Tagger from Apache OenNLP (OpenNLP, 2011) which I used to perform POS tagging. Furthermore, I used Porter's Stemming algorithm (Porter, 1980) on the Stanford dataset to remove common English word endings resulting from inflection.

Chapter 5

Feature Engineering

5.1 Feature Construction

5.1.1 Semantic Features

To improve classifier accuracy I constructed the following semantic features:

POS Tags The previously generated POS tags were used to identify the number of different parts of speech in each Tweet. In the dataset this resulted in one column per part of speech containing the number of its occurrences in every tweet.

Lexicon Scores I used the sentiment lexicon of common positive and negative words developed by Liu et al. (2005) to assign a score to each tweet. For each negative word identified by the lexicon, one was deducted while for each positive word one was added. The procedure resulted in a score ranging from -5 to 5 on the Stanford dataset.

Adjective Noun Distances In their work on miming sentiments in product reviews Ding et al. (2008) implemented an algorithm which calculates distances between opinion bearing words and product features. Assuming that nouns identify objects while adjectives and adverbs indicate sentiment I calculated distances between nouns identified by the POS tagger and adjectives whose sentiment orientation was determined by the previously used opinion lexicon. I measured distance in words located between the object and the opinion bearing word. Starting from a score of one for opinion words located adjacent to objects I divided by the number of words located between the two. Thus, greater distances resulted in lower

scores. The rationale behind this approach is that the further an opinion bearing word is located from an object, the less likely it is to refer to that particular object.

Adjective/Personal Pronoun to Noun Proportion According to the findings of Pak and Paroubek (2010) objective tweets make higher use of common and proper nouns, while subjective texts use more personal pronouns. Thus, I implemented a subjectivity score which sums up the number of nouns and subtracts the number of adjectives and personal pronouns in every tweet. Tweets with higher subjectivity scores are likely to bear stronger sentiment orientation.

Length of Tweet A column containing the number of words in each Tweet. Whether and how the length of a tweet has an impact on sentiment probably depends to a large extent on context and the topic. Accordingly, it needs to be determined through experimentation.

Word Length A column containing the average length words in a Tweet. As with Tweet length, the value of this feature regarding the prediction of sentiment needs to be evaluated through experimentation.

5.1.2 Topical Features

Saif et al. (2012) argue that topical features are far superior to semantic features for predicting document sentiment. The most common method for extracting topics from documents is Latent Dirichlet Allocation (Blei et al., 2003). As LDA requires the specification of a certain number of topics to which it assigns documents, I initially performed hierarchical clustering on the document vectors to investigate whether a pattern exists by which tweets naturally separate. As this was not the case I first calculated the term frequency inverse document frequency on the data and removed terms with frequencies below 0.1 as well as tweets that consisted exclusively of infrequent terms. In the next step, I performed LDA with 200 iterations on a 10.000 and with 50 iterations on a 2000 tweet subset of the Stanford data incrementing the number of topics by one with each iteration. For 10.000 tweets and 200 iterations, this calculation took approximately 38 hours to complete while for 2000 tweets and 50

iterations it terminated in just under 1 hour. As a metric for determining the ideal number of topics, I estimated the log likelihood of each number of topics in relation to the data.

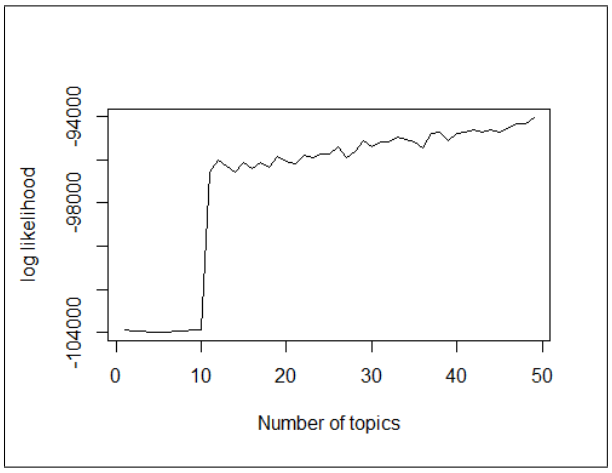

Figure 5.1: LDA: Log Likelihood for different topics

On the 2000 Tweet Stanford subset the log likelihood spiked when the number of topics increased from 10 to 11 and then kept increasing slowly. As no significant increase occured after the spike I decided to allocate documents to 11 topics.

5.1.3 Evaluation of Manually Constructed Features

I investigated the manually constructed features to determine their effects on accuracy, sensitivity, and specificity by adding each feature individually to the basic dataset (the document term matrix) and fitting multiple models. Since the purpose at this stage was to investigate the impact of features on model performance, I did not tune the models but only fit them with default parameters. As no model tuning was performed, no validation set was required. The results reported were obtained on the test set. The highest accuracy was consistently achieved by Random Forests and Support Vector Machines with a Linear Kernel and a cost parameter of 1. Using only unigrams, bigrams and trigrams these models achieved 72.23% and 73% accuracy respectively. As can be seen in figure 5.2 the topic models resulted in by far the largest improvement across most models yielding 76.75% accuracy for Random Forests. The second most valuable feature was the lexicon score achieving 73.64% accuracy on Linear Kernel SVMs and 73.91% with Random Forests. The distance score also resulted in an improvement on

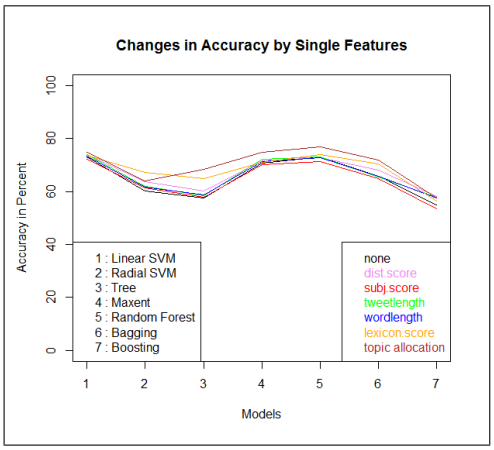

Figure 5.2: Change in accuracy by feature and model

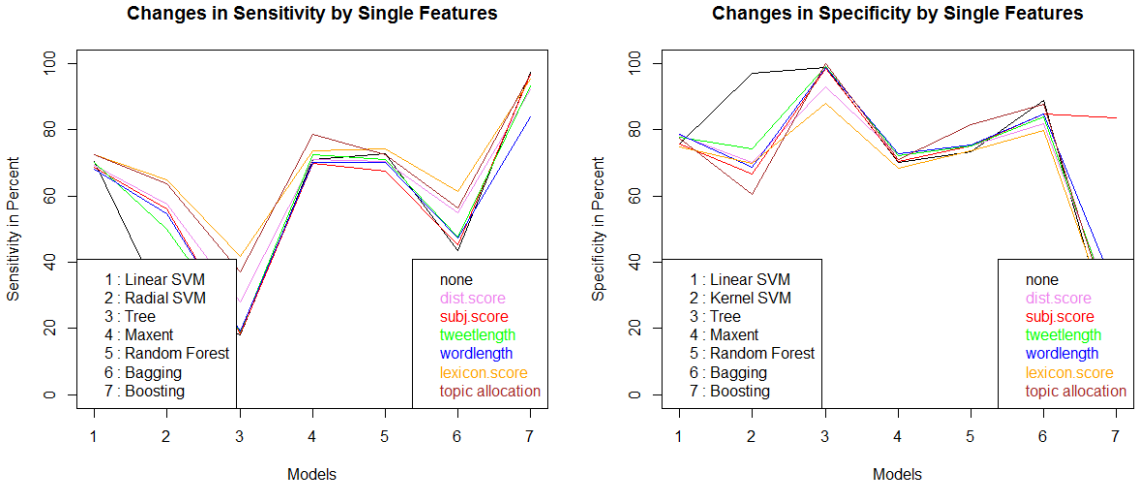

Figure 5.3: Changes in sensitivity and specificity by feature and model

some of the models yielding 73.88% accuracy for Linear Kernel SVMs while there was no remarkable change in the Random Forest model. The other features did not produce significant improvements over the basic document term matrix while the subjectivity score led to overfitting as it resulted in even lower accuracy. Still these results need to be treated with caution as 2000 tweets is not a large subset, but fitting models on larger subsets was too time-consuming while using larger sets did not yield improvements in model performance. The results were obtained by fitting models included in the RTextTools package and only served to gain an initial idea of the impact new features had on performance.

5.2 Feature Selection

At the feature selection stage, the features present in the dataset consisted of

- unigrams
- bigrams
- trigrams
- The constructed sets of features described in section 5.1.

On the 2000 tweet Stanford subset all the features combined amounted to 1270.

5.2.1 Model Based Selection

Given the large amount of initial features, applying a computationally expensive wrapper method like a genetic algorithm or recursive feature elimination on the full set of features would waste time and computing resources. The Random Forest model internally measures the decrease in node impurity achieved by each feature across all trees. Powerful features are better at separating data into classes resulting in higher node purity. The performance of Random Forest models proved to be relatively stable across multiple features and subsets of the data. Thus, ranking features by their achieved decrease in node impurity on the random forest model appeared to be a good way to quickly detect features that did not contribute significantly to sentiment classification. To evaluate the results I fitted multiple models reducing the number of features in each iteration.

First iteration 1219 Features (Mean Gini decrease >0)

Second Iteration 658 Features (Mean Gini decrease >0.1)

Third Iteration 174 Features (Mean Gini decrease >0.5)

Fourth Iteration 104 Features (Mean Gini decrease >0.75)

Fifth Iteration 79 Features (Mean Gini decrease >1)

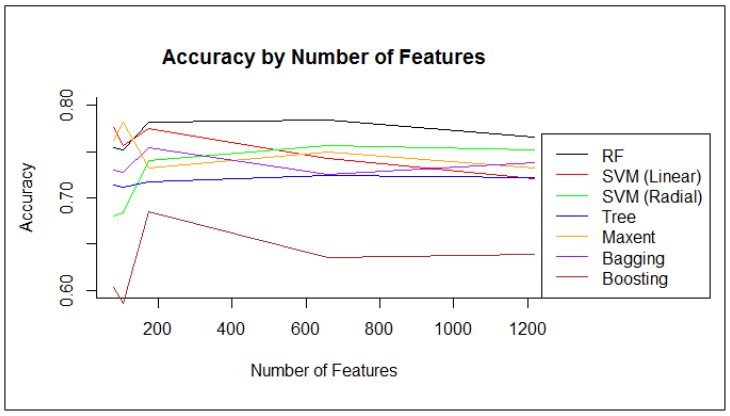

Figure 5.4: Change in accuracy by number of features

For every set of features I fitted several different classifiers to investigate which features would yield the best results.

The highest accuracy was achieved by Random Forests on the 658 feature subset. In terms of sensitivity, Boosting performed best on the smallest feature subset (79 features) attaining 95%. The single tree model achieved the best specificity score of 89.65% also on the smallest feature subset. As can be seen from the plot 5.4 accuracy increased significantly up to the 174 features used in the third iteration. The only exception was Linear Kernel SVMs whose accuracy peaked much earlier on the 79 feature subset. Up to 658, a slight increase was noticeable with the exception of Boosting whose accuracy declined. Beyond 658 up to the full feature set accuracy either slowly declined or remained constant for all classifiers. Additional features beyond 658 seemed to only cause overfitting. This result mirrors the findings of Saif et al. (2012) who found a number of approximately 500 features to be ideal for sentiment classification. The Mean Gini Decrease of the Random Forest classifier was effectively able to filter out roughly 50% redundant features. Testing the selected feature subsets on other classifiers confirmed their redundancy. Support Vector Machines and Random Forests performed best. On the data set enhanced by feature construction accuracy rates of up to 77% were achieved by those classifiers compared to 73% to 74% on the original data set.

5.2.2 Feature Correlation

The previous section described how initially a bulk of redundant features has been removed to reduce dimensionality. As has been demonstrated no model has seen improved performance beyond 658 features. Yet, as will be demonstrated in chapter 6 different models perform best on different feature sets (Kuhn, 2013) , thus fitting all models to the 658 feature subset would likely result in a deterioration of performance among some models. The Chi Square test is a popular statistical technique for measuring the goodness of fit between two variables (Greenwood, 1996). It can be used for feature selection by measuring the fit between the response variable and individual predictors. As it does not necessitate fitting a model for evaluation, it belongs to the category of filter methods. I used the Chi Square test to determine the likelihood that the tested predictor is not independent of the response using a standard p-value of 0.05. 135 out of the remaining 658 variables displayed a statistically significant relationship with the response variable. Still, it would be premature to simply discard the insignificant predictors. Firstly, evaluating all features separately results in 658 Chi Square tests. The large number of tests makes the occurrence of an error in at least one of them increasingly likely. To mitigate the error I performed another iteration of Chi Square tests with a Bonferroni adjusted p-value which resulted in 104 significant predictors. The results obtained by fitting the previously used algorithms on this subset did not differ significantly from the 127 feature subset only performing slightly worse. Accordingly, the 104 feature subset was discarded. Secondly, the Chi Square test evaluates all predictors in isolation. Therefore, it does not account for correlation among the predictor possibly giving rise to issues with multicollinearity later in the modeling process (Guyon and Elisseeff, 2003). For those reasons, the Chi Square test should not be solely relied upon for feature selection.

5.2.3 Recursive Selection

To further reduce dimensionality and find the best predictors I performed recursive feature selection on the 658 feature subset determined in section 5.2. Recursive or backward feature selection operates by ranking features from most to least important

and then iteratively performing multiple model fits. In each iteration, the least significant features are removed and the importance of the remaining features is recalculated (Kuhn, 2013). As a reference model, I used Random Forests again as it proved to be the most stable algorithm delivering very consistent results across different samples and data sizes.

By applying the recursive method it was possible to reduce the feature set down to just 5 predictors without resulting in a significant deterioration in model performance **"lexicon.score", "farrah", "sad", "lol", "fawcett"** could be identified as the most influential features: One of them, lexicon score, is a manually constructed feature described in 5.1.1. The other four are terms extracted from the Tweets. It makes intuitive sense that the word "sad" is a strong indicator of negative sentiment. It occured in 158 of the analyzed tweets and only 2 of them had been labelled as positive. "LOL" short for "Laughing out Loud" is a colloquial way of expressing amusement and is often used sarcastically. Thus, using it for sentiment prediction is not as straightforward as the word "sad". Out of a total of 67 tweets in which the word occurred 15 were labeled as positive while 52 were negative. This result is most likely due to the sarcastic use of the word. Farah Fawcett is the name of an American actress whose death coincided with the collection of Tweets for the Stanford dataset. Accordingly, a large amount of Tweets were related to her passing. The word "Farrah" occurred in 289 tweets while "Fawcett" was found in 212 tweets. All of them had been labeled negatively. The subsets of 658, 127, and 5 features determined by model-based methods, filter methods, and recursive selection respectively serve as the basis for model construction described in the next chapter.

Chapter 6

Single Algorithms and Model Tuning

6.1 Performance Evaluation Metrics

In previous chapters, model performance was measured in classification accuracy attained on the test set. While accuracy can give a first impression of the model's potential, it is not enough for comprehensively evaluating model performance. This is due to several reasons. Firstly, the standard for judging accuracy is relative and highly dependent on the distribution of the data. A model trained on a dataset consisting of 90% observations of class A and only 10% of class B could achieve 90% accuracy by simply predicting class A all the time. A prediction accuracy of 90% could be considered outstanding on a balanced dataset, while it is essentially useless on a highly skewed set. Although the data sets used for the analysis are balanced, the Kappa statistic is being used to evaluate the obtained results against the expected accuracy. Furthermore, accuracy does not provide any information on how the model performed on different classes. A classifier could possibly learn the structures underlying observations of one class much better than those of another. This phenomenon has already been discussed in section 5.2 where some models performed very well in predicting negative sentiments while yielding inferior results on positive cases. To alleviate this problem sensitivity and specificity are measured as "area under the curve values" in a ROC curve.

6.2 Classifiers

The caret package was used for fitting and tuning most of the models. To evaluate classifier performance the data was partitioned into a training and a test set using a 60 (training)/40 (testing) split. When fitting classifiers on the training set 10 fold cross-validation was applied. Some models such as neural networks or support vector machines require parameter tuning to optimize their performance. Hyperparameter tuning was undertaken using grid search over 10 values for each parameter. For example, for neural networks, hidden layers were increased in increments of 2 from 1 to 19 while weight decay varied from 0 to 0.1 roughly doubling with each iteration. The combination yielding the best "area under the ROC curve" value was selected for the final model.

6.2.1 Decision Trees

As has been illustrated in section 5.2 in terms of classification accuracy a decision tree is inferior to most other classifiers. Additionally, there was an extreme imbalance between error rates on positive and negative predictions. Trees did very well on correctly classifying positive sentiment achieving rates of up to 89.65% compared to negative sentiment (58%). These results were investigated more in depth by training C5.0 trees with 10 fold cross validation on the training set and evaluating their performance on the test set. The C5.0 tree is an updated version of the C4.5 decision tree algorithm developed by Ross Quinlan (Salzberg, 1994). It goes beyond the abilities of a basic decision tree as it is able to perform classification on multiclass data, can generate rule models and supports boosting. As can be seen, predictive accuracy improves as the

Table 6.1: Decision Tree performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
5 features	0.7262	0.4559	0.5844	0.8747
127 features	0.7425	0.4872	0.6455	0.844
658 features	0.755	0.5109	0.7115	0.8005

number of features is increased while sensitivity and specificity become more balanced. The C5.0 algorithm automatically performs feature selection, thus using a larger feature set gives the algorithm more choice to assemble a good feature combination. The

C5.0 tree used 310 out of the 658 features to perform classification. Particularly on smaller feature sets the tree has learned the features very well that explain positive sentiment while it performed poorly on negative cases. At this point it should be recalled that the reference class is negative sentiment, thus sensitivity describes the correct classification of negative classes. The McNemar test for comparing the performance of the three models yielded the following p - values:

5 vs 127 p-value = 0.23

5 vs 658 p-value = 0.035

127 vs 658 p-value = 0.34

Thus, only the the 5 feature and the 658 feature model have generated significantly different results from each other.

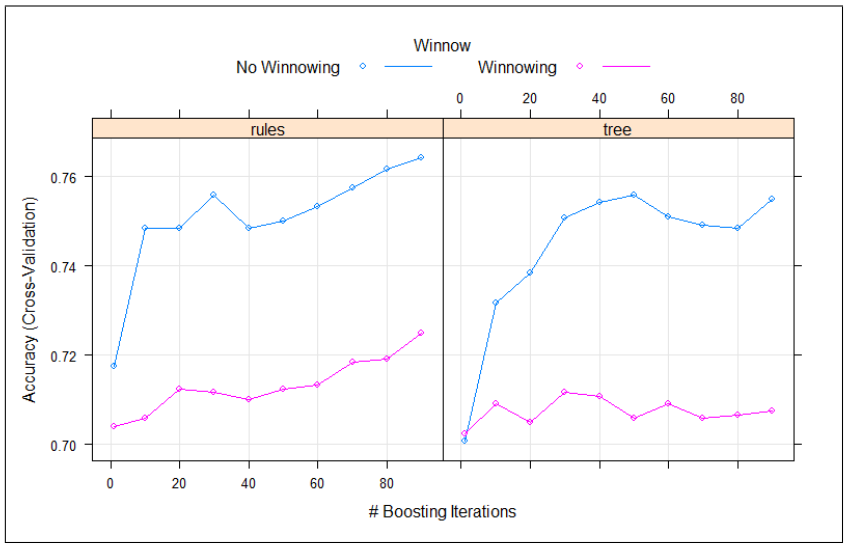

Figure 6.1: C5.0 tree and rule model performance with different numbers of trees and boosting iterations on the cross validated training set

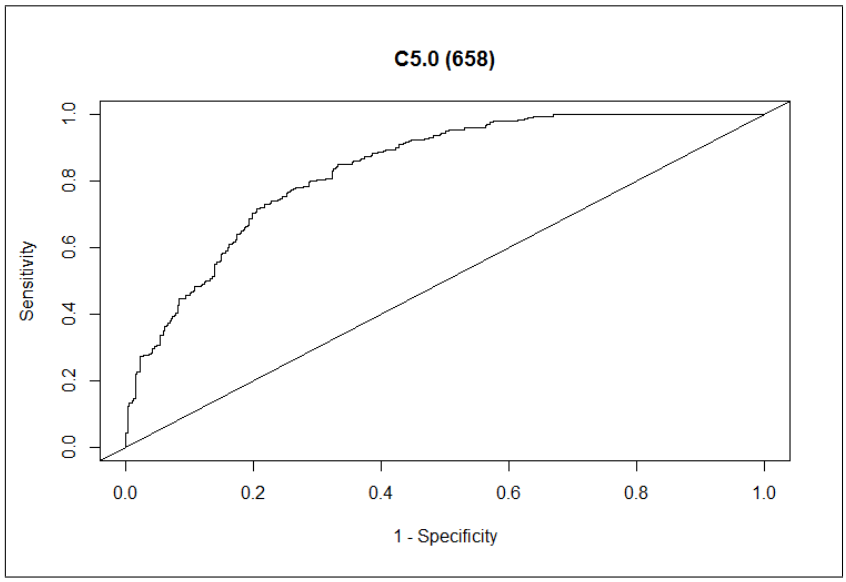

Figure 6.2: C5.0 ROC Curve

6.2.2 Support Vector Machines

Among single algorithms, support vector machines constantly performed very well in terms of classification accuracy. Initially, I investigated the difference between radial kernel and linear kernel SVMs by splitting the data into a training and a validation set and fitting multiple versions of these models with different cost parameters. The following models were trained on the training set:

SVM (Linear Kernel) Incrementing the cost parameter by 1 in every iteration from 1 to 20

SVM (Radial Kernel) Incrementing the cost parameter by 20 in every iteration from 20 to 400.

Then, the performance of those models was observed on the validation set. Figures 6.3a and 6.3b illustrate the performance of support vector machines on the 800 tweet validation set with various cost parameters. Linear Kernel SVMs performed best with a low-cost parameter. Thus, a linear kernel performs best when maximizing the classification margin even at the cost of misclassifying more observations. Radial kernels tended to perform best with a cost parameter between 30 and 50 while higher costs led to a decline in accuracy due to overfitting the training data.

The more complex decision boundaries produced by radial SVMs with higher cost values slightly outperform low-cost Linear kernel SVMs in terms of accuracy, specificity,

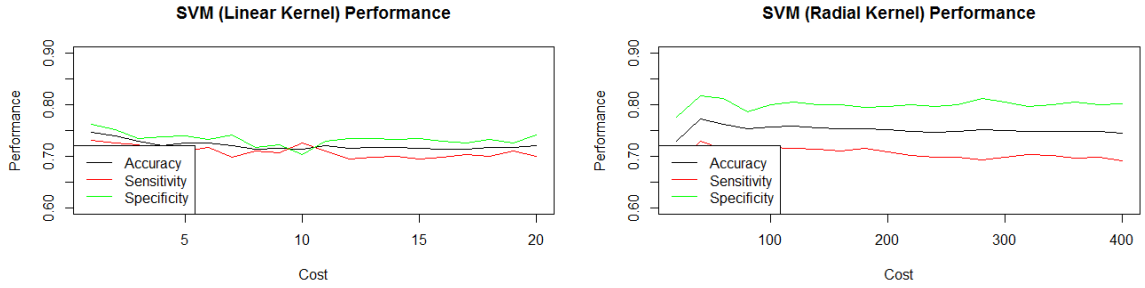

Figure 6.3: SVM Performance with different cost parameters on the test set

and sensitivity. For the purpose of ensemble construction, a linear kernel SVM was used. I decided that the small improvement did not merit the increased model complexity and cost in terms of computing resources. Furthermore, linear kernels provided relatively stable performance at a constant cost of one.

Table 6.2: Linear kernel SVM performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
5 features	0.7262	0.4556	0.5966	0.8619
127 features	0.7538	0.5097	0.6528	0.8593
658 features	0.7475	0.4963	0.687	0.8107

The best performance was achieved on a subset of 127 features. As decision trees, SVMs performed much better in terms of specificity than sensitivity.

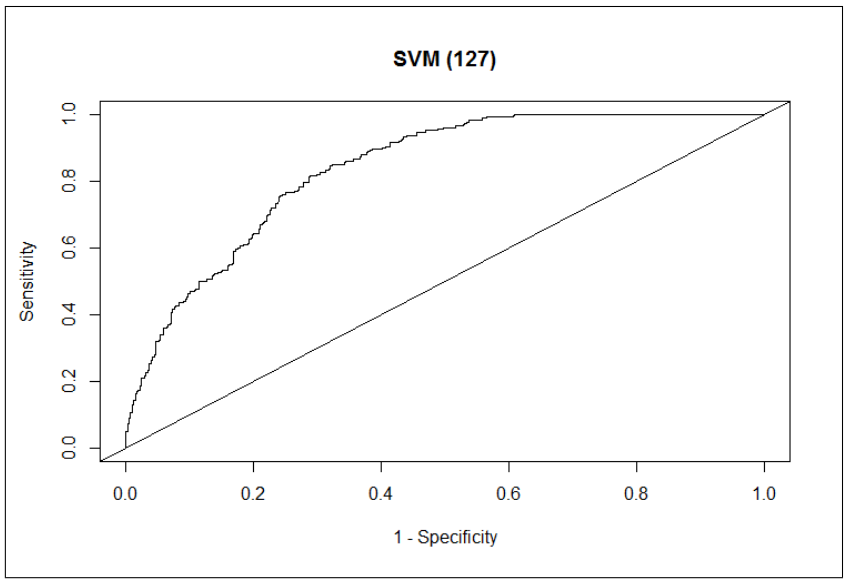

Figure 6.4: ROC curve of the linear SVM model with cost one

5 vs 127 p-value = 0.063

5 vs 658 p-value = 0.2

127 vs 658 p-value = 0.71

6.2.3 Random Forests

As Hastie (2009) explain decision trees are prone to overfitting and generally display high variance. random forests is an ensemble method which grows several decorrelated deep decision trees and averages their results. This has the benefit of reducing variance while making use of decision trees' ability to capture complex patterns in the data. Along with neural networks, random forests overall performed best among all tested classifiers achieving up to 77.13% classification accuracy while they also exhibited a large degree of stability. Their accuracy, sensitivity and specificity rates remained relatively constant when testing them on different subsamples of the same data and on different numbers of trees. For random forests using more features generally tended to result in higher performance. This is partly due to the fact that averaging of multiple tree learners effectively prevents the algorithm from overfitting. Furthermore, the random forest algorithm internally performs feature selection.

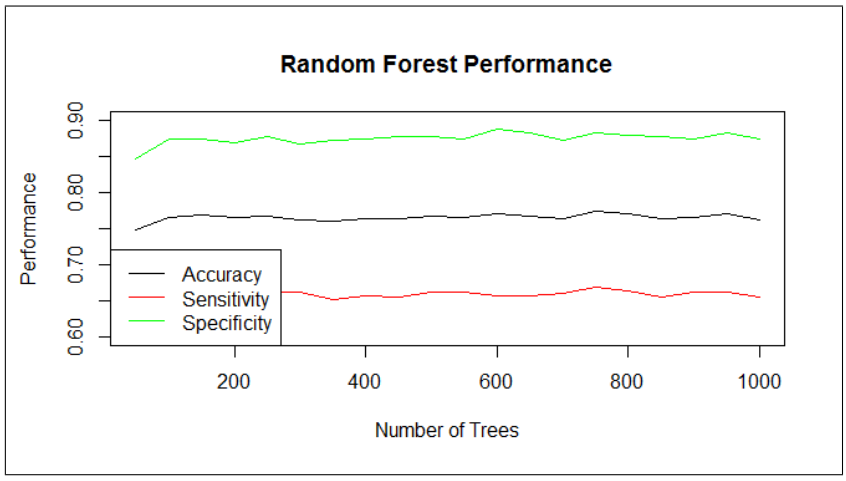

Figure 6.5: Random forest performance with different numbers of trees on the test set

Table 6.3: Random forest performance summary over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
features_5	0.7288	0.4608	0.5892	0.8747
features_127	0.705	0.4168	0.4474	0.9744
features_658	0.7538	0.5094	0.6626	0.8491

5 vs 127 p-value = 0.088

5 vs 658 p-value = 0.025

127 vs 658 p-value = 0.0014

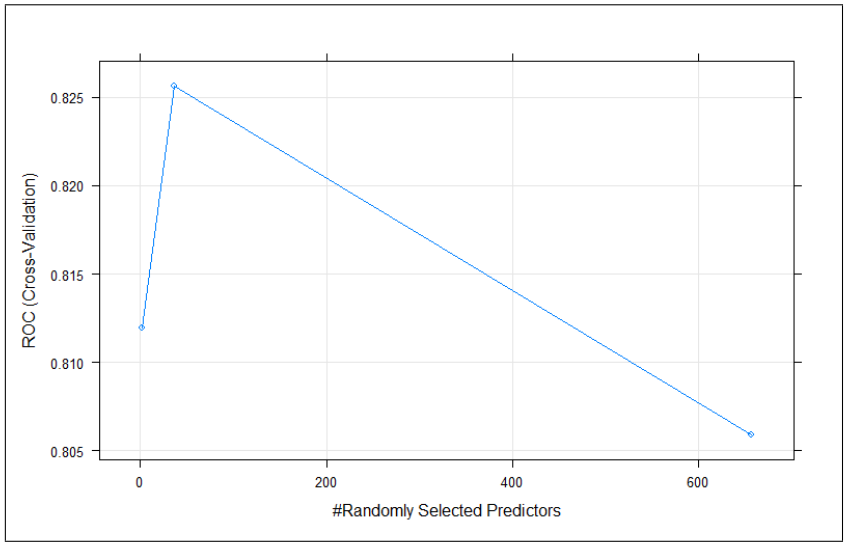

Figure 6.6: Change in Random Forest performance on the cross validated training set depending on the number of variables randomly selected for each split

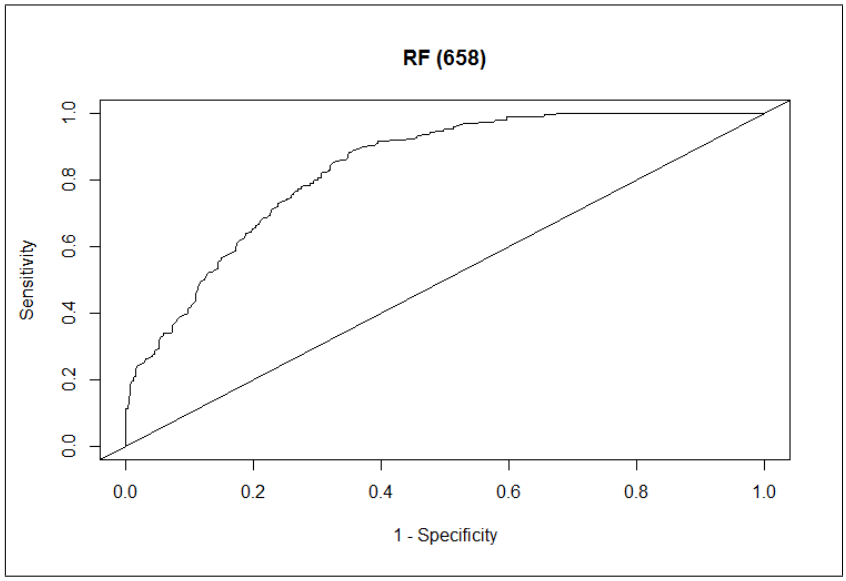

Figure 6.7: Random Forest ROC curve

6.2.4 Naive Bayes

Among all tested classifiers Naive Bayes performed worst. As can be seen in figure 6.4 the classifier achieved an accuracy of 69.25% on the 5 feature subset. In higher dimensions, the algorithm simply reverted to always predicting one class.

Table 6.4: Naive Bayes performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
features_5	0.6925	0.3926	0.4181	0.9795
features_127	0.5112	0.0	1.0	0.0
features_658	0.5112	0.0	1.0	0.0

Since the problem at hand requires classification rather than regression, a nonparametric naive Bayes classifier has been used. Nonparametric naive Bayes determines results simply based on the frequencies with which a certain class value occurs given the presence of certain feature values without consideration for possible dependencies between features. Rennie et al. (2003) explain that especially in text classification settings the independence assumption negatively affects the performance because every word is weighted equally. The weights of strongly dependent words will add up increasing the magnitude of evidence for one class under the assumption that all those words independently provided evidence for one class. Using just 5 features the between feature correlation was very low, thus the independence assumption was largely justified. When using higher dimensional data many features were closely correlated. Thus, the assumptions underlying Naive Bayes were inadequate.

From the previous explanations, it is evident that the algorithm applied to 127 and 658 features produced exactly the same results which in turn differed from the class labels attained on 5 features. Still, for completeness, the McNemar test was performed.

5 vs 127 p-value = $7.5e^{-}9$

5 vs 658 p-value = $7.5e^{-}9$

127 vs 658 p-value = 1

6.2.5 Neural Networks

Neural networks can be extremely powerful in complex classification settings (Kuhn, 2013) as they have a large range of parameter values such as initial weights of single neurons, weight decay over time, and layers of hidden neurons that can be adjusted to model almost arbitrarily complex patterns. Accordingly, they are well suited for high dimensional and high volume datasets. But these characteristics, as Hastie (2009) point out, make neural nets prone to overfitting and often overparametrized. In my

Table 6.5: Neural Network performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
features_5	0.7275	0.4582	0.5941	0.867
features_127	0.765	0.5312	0.7042	0.8286
features_658	0.74	0.4801	0.7311	0.7494

experiments neural nets achieved the highest classification accuracy besides random forests and the highest sensitivity among all classifiers. The highest test accuracy was achieved by a neural net trained on 127 features. When using larger feature sets test accuracy declined while training accuracy still remained relatively high. Using more features evidently led the neural net to overfit the training data. On the other hand test sensitivity was highest when using the largest feature set. Most models performed considerably worse in terms of sensitivity than specificity with neural nets on high dimensional data achieving the best results.

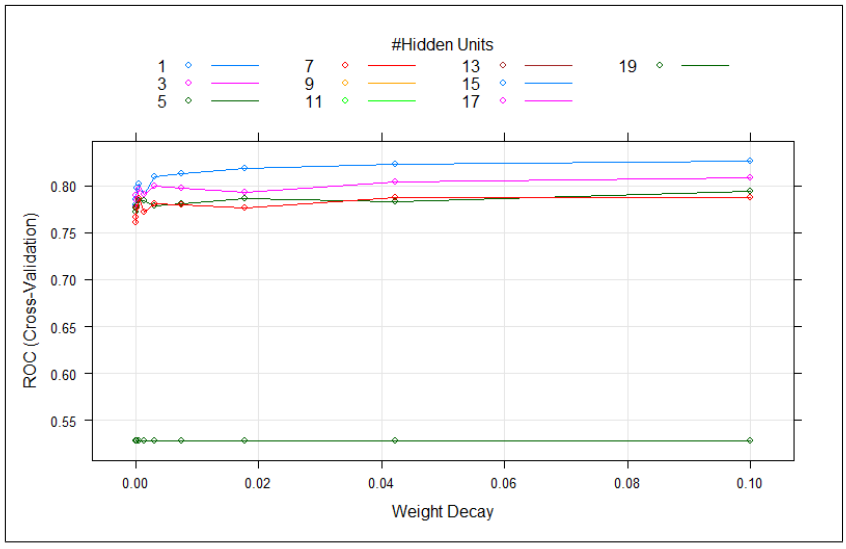

Figure 6.8: Change in performance of feed forward neural nets on the cross validated training set depending on the number of hidden layers and the rate of weight decay

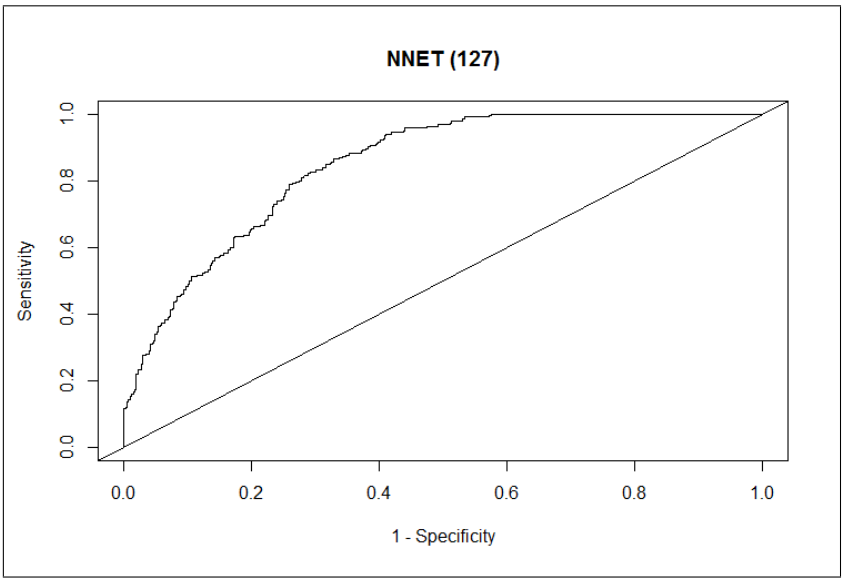

Figure 6.9: Neural net ROC curve

The results of the McNemar test indicate that using 127 features neural nets perform significantly better than on the 5 feature subset while the difference between 127 and 658 features can be considered marginally significant. A clearly nonsignificant p-value indicates that the output generated by neural nets trained on very high dimensional data does not differ significantly from results achieved on low dimensional data.

5 vs 127 p-value = 0.0076

5 vs 658 p-value = 0.49

127 vs 658 p-value = 0.13

6.2.6 Bagged Trees

Performing tree bagging across several feature subsets yielded results that were inferior to those of most other classifiers. Random forests, which in contrast to bagging splits nodes only over a random subset of the available features, clearly performs better across the board. Thus, tree bagging is not further considered as an ensemble candidate. The

Table 6.6: Tree bagging performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
features_5	0.7288	0.4608	0.5917	0.8721
features_127	0.7113	0.4237	0.6626	0.7621
features_658	0.7175	0.4366	0.6504	0.7877

results generated by tree bagging on different subsets all did not differ significantly from each other.

5 vs 127 p-value = 0.26

5 vs 658 p-value = 0.41

127 vs 658 p-value = 0.73

6.2.7 Generalized Boosting Machines

As tree bagging, generalized boosting machines are ensemble methods that mainly make use of tree learners, but in this context, I am going to treat them as a single learner to be used as members of an ensemble. GBMs extend simple boosting by introducing a loss function. Gradient descent is applied by iteratively adding new learners one at a time which are highly correlated with the gradient of the loss function. This process continues until a local minimum is found (Friedman, 2001). Thus, boosting essentially turns into a mathematical optimization problem. As the following figures indicate, GBMs do reasonably well, especially on the 127 feature subset. In terms of parameter tuning an approximate number of 150 Boosting iterations and a restriction of the maximum tree depth to no more than two nodes achieved the best performance in terms of ROC.

Table 6.7: Tree boosting performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
features_5	0.7275	0.4584	0.5868	0.8747
features_127	0.7512	0.5047	0.6504	0.8568
features_658	0.7362	0.4743	0.6553	0.821

5 vs 127 p-value = 0.017

5 vs 658 p-value = 0.044

127 vs 658 p-value = 0.13

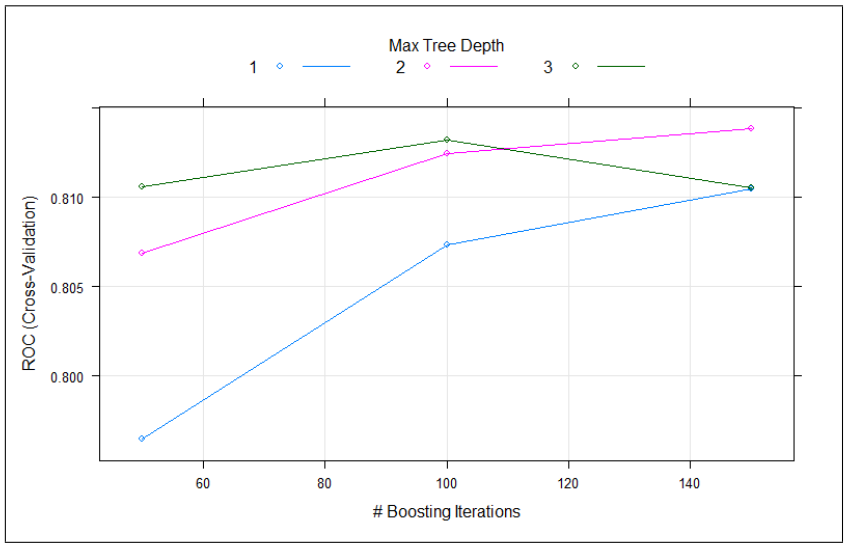

Figure 6.10: Change in performance of general boosting machines on the cross validated training set depending on the number of boosting iterations

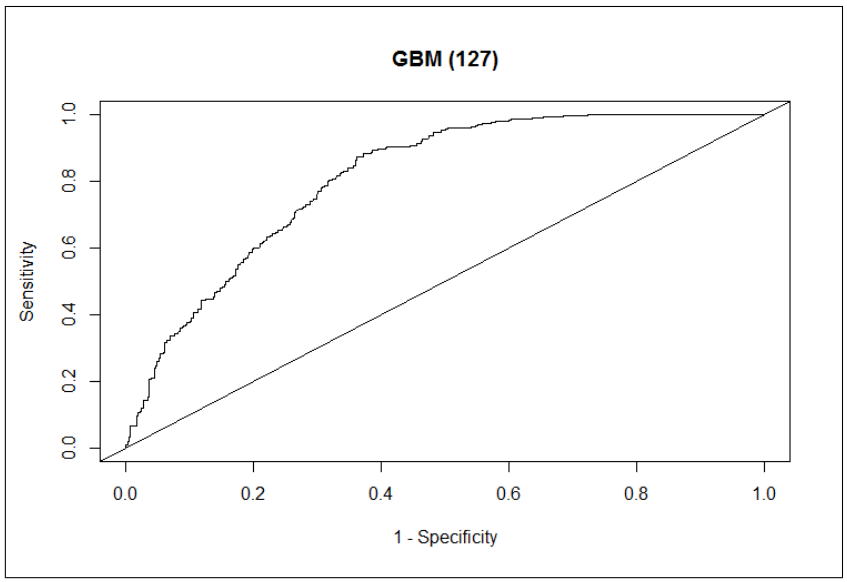

Figure 6.11: GBM ROC

6.2.8 Penalized Linear Models

Hastie et al. (1995) explain that generalized linear models such as LDA do not deal well with large numbers of correlated predictors and nonlinear decision boundaries. In such contexts penalized linear models can improve performance by introducing shrinkage parameters that either shrink unimportant parameters or discard them completely Hastie (2009) which may result in lower prediction error. Accordingly, I decided to fit penalized linear models to see how it would impact performance. In case the penalty term has no impact at all, the penalized method equals the normal linear method.

Penalized Linear Discriminant Analysis

The authors proposed a range of models including penalized discriminant analysis which adds a parameter that penalizes high covariance to the standard LDA model and in extension reduces the feature space Hastie et al. (1995).

Table 6.8: Penalized Discriminant Analysis performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
5 features	0.7212	0.4459	0.5844	0.8645
127 features	0.7762	0.5543	0.6822	0.8747
658 features	0.7212	0.4433	0.6846	0.7596

5 vs 127 p-value = 0.00099

5 vs 658 p-value = 1.0

127 vs 658 p-value = 0.0017

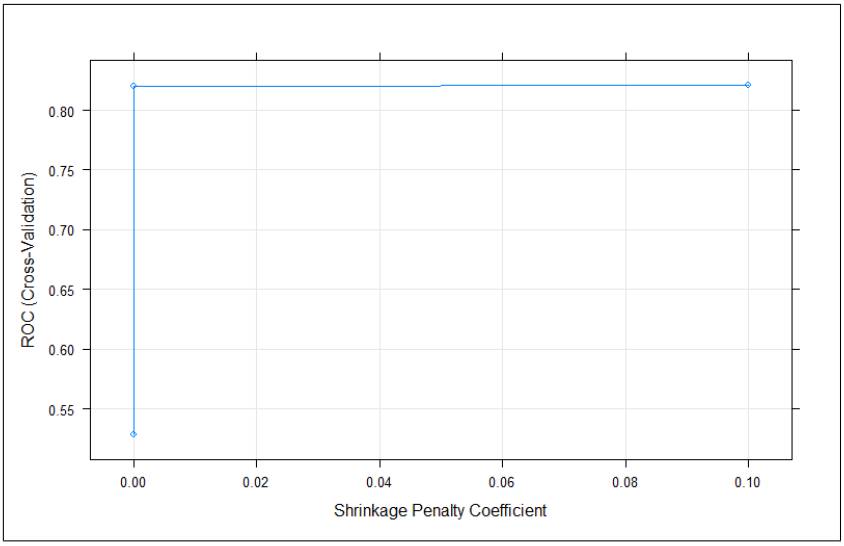

Figure 6.12: Change in performance of Penalized Discriminant Analysis on the cross validated training set depending on shrinkage penalty factor

As the plot indicates, the shrinkage parameter did not have any influence at all. Accordingly, "PDA" equals simple linear discriminant analysis.

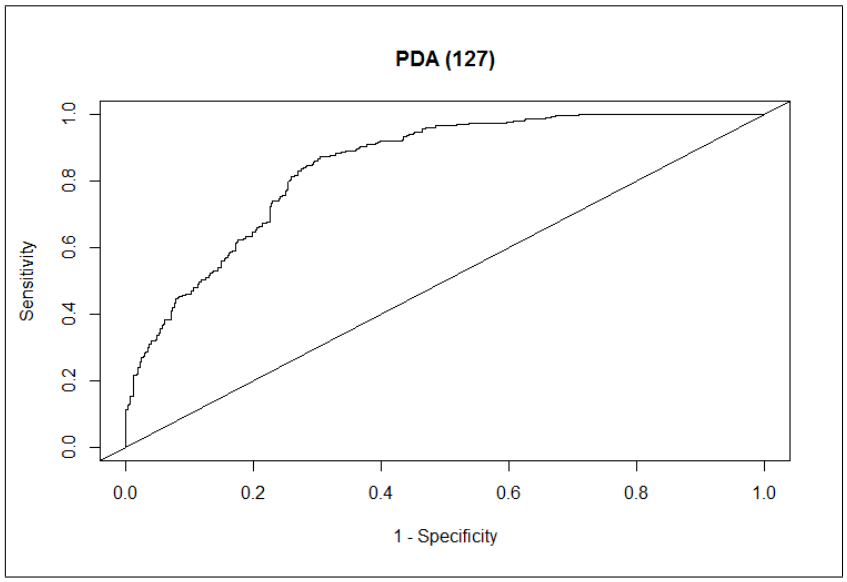

Figure 6.13: ROC curve of a Penalized Discriminant model with the best determined tuning parameters on the 127 feature subset

Elastic Net

Further notable linear models with shrinkage parameters are ridge regression, which extends linear regression by adding an L2 regularization parameter, and the lasso, which performs L1 regularization on the linear regression model (Hastie, 2009). Since L2 regularization penalizes the residual sum of squares, ridge regression is unable to shrink parameters to zero. Consequently, it cannot remove features, but improved performance over the OLS model is achieved by a better bias-variance trade-off. The Lasso suffers from the opposite problem. Penalizing the sum of the absolute values allows it to attain a parsimonious feature subset, but results in high variance (Zou and Hastie, 2005). To address these problems the elastic net, a mixture model of ridge regression and the Lasso has been proposed (Zou and Hastie, 2005).

Table 6.9: Elastic Net performance over different features on the test set

	Accuracy	Kappa	Sensitivity	Specificity
5 features	0.7262	0.4557	0.5941	0.8645
127 features	0.7775	0.5566	0.6944	0.8645
658 features	0.7412	0.4852	0.6235	0.8645

5 vs 127 p-value = 0.00013

5 vs 658 p-value = 0.17

127 vs 658 p-value = 0.0019

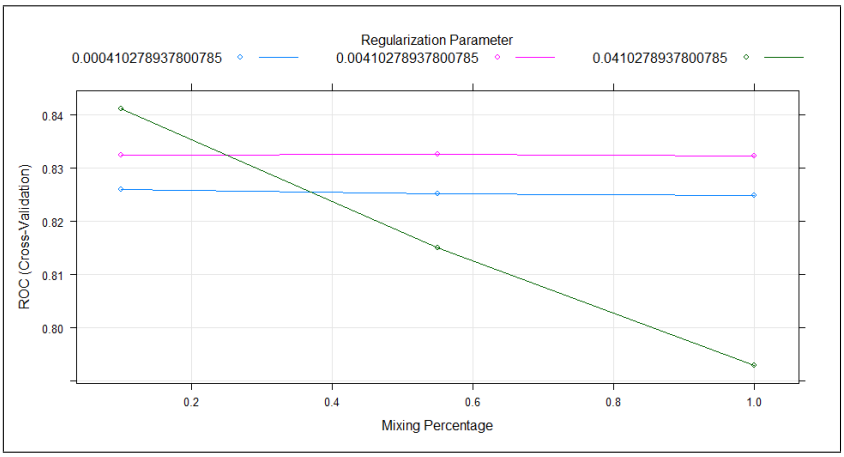

Figure 6.14: Change in performance of Elastic Net models depending the regularization factor and the mixing percentage between the Lasso and Ridge Regression

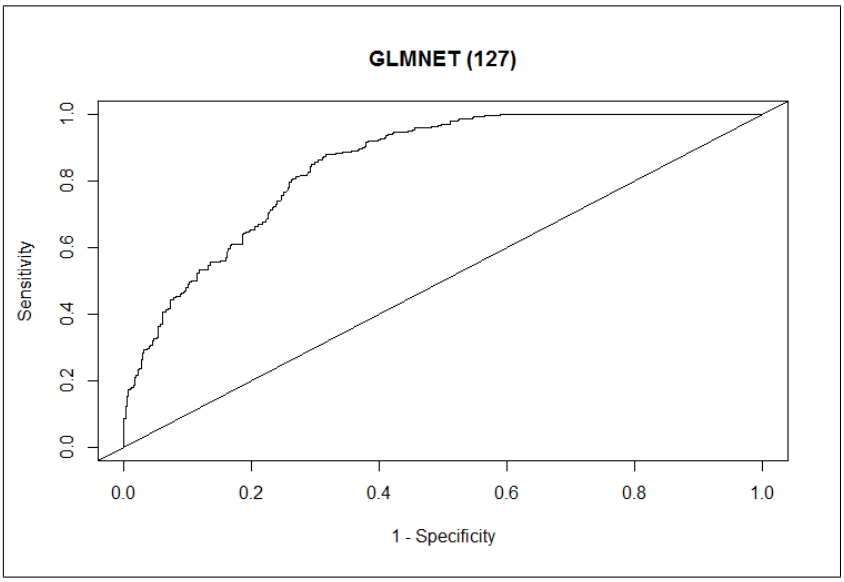

Figure 6.15: Elastic Net ROC curve using the best determined tuning parameters and the 127 feature subset

6.3 Final Model Selection

In Chapter 6 several machine learning algorithms were evaluated with regard to their performance. To enhance performance, achieve a better balance between class predictions, and increase model diversity I trained models on three different feature subsets. For every algorithm, I selected the feature subset on which the algorithm attained the highest accuracy resulting in the following models

- C5.0 trees fitted on 658 features
- General Boosting Machine (GBM) on 127 features
- Elastic Net (GLMNET) on 127 features
- Linear Kernel Support Vector Machine (SVM) on 127 features
- Naive Bayes (NB) on 5 features
- Neural Net (NNET) on 127 features
- Partial Discriminant Analysis (PDA) on 127 features
- Random Forest (RF) on 658 features

To increase diversity I selected further models on the basis of their class predictions. I added **Random Forests on 127 features** as they achieved almost perfect specificity scores. As has been demonstrated in 6.2 all models scored much higher on specificity than on sensitivity. Accordingly, I added **Neural Networks on 658 features** which did best in terms of sensitivity and created To further alleviate the imbalance in class predictions I performed oversampling of classes by replicating positively labeled samples and repeatedly training several algorithms until sensitivity almost equaled specificity for predictions generated on the test set. PDA and NNET trained on 127 features first achieved class balance while retaining relatively high classification accuracies after replicating roughly 400 positive samples in the training set. Accordingly, I added **PDA on 127 features with oversampling** and **NNET on 127 features with oversampling** to the candidates for ensemble construction resulting in a final set of 11 models.

Chapter 7

Established Ensemble Strategies

7.1 Simple Majority Voting

The combination of the 11 base models introduced in section 6.3 by simple majority vote achieved an average of 81.25% accuracy, a sensitivity of 76% and a specificity of 86,7%. This marks a significant improvement over single models testifying to their diversity.

7.2 Bagging

Bagging is an ensemble technique where multiple models are trained on subsamples of the same data. Samples are drawn with replacement from the original dataset which is known as bootstrapping. The predictions of the models trained on the bootstrap samples are either averaged in a regression setting or used for majority voting in a classification setting. Since all the samples are drawn from the same data, bagging primarily serves to improve stability and reduce variance compared to single models. When the size of the bootstrap sample equals that of the original dataset it can be expected to have an average of 63.2% of the unique samples contained in the original dataset (Efron and Tibshirani, 1994).

7.2.1 Implementation

I bagged C5.0 trees, support vector machines, neural nets and the elastic net and penalized discriminant analysis models. Especially C5.0 trees and neural nets proved to be relatively unstable since their results varied considerably. I was primarily interested in whether I could improve the stability of those models by bagging them. I did not bag random forests and general boosting machines since these are already ensemble

methods. Furthermore, random forests implemenent a variation of bagging and display great stability. I trained every algorithm on 10 bootstrap samples each of size equal to the original training data. Then I implemented majority voting for all predictions generated by one algorithm. Finally, I also aggregated the results of all bootstrap predictions across all models.

7.2.2 Experiments and Results

The best results were achieved by the linear models with the elastic net achieving 76.5$ and Partial Discriminant Analysis attaining 77.25% classification accuracy. Still, bagging did not improve accuracy significantly compared to using single models for any of the algorithms. Furthermore, the individual class predictions, as measured by sensitivity and specificity, did not change significantly. From these results, it appears that performing bootstrap aggregation does not yield any significant benefits compared to fitting single models. This preliminary conclusion will be investigated more thoroughly in section 9.1.

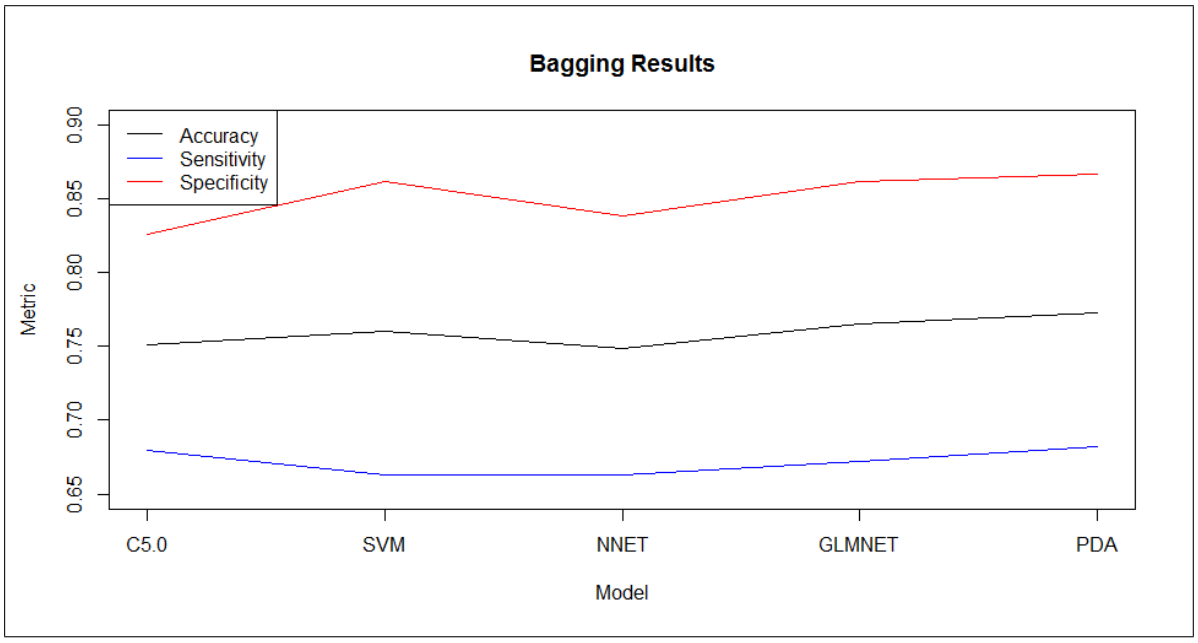

Figure 7.1: Model performance across multiple Boosting iterations

7.3 Stacking

Stacked generalization or Stacking uses 2 or more layers of models whereby models in higher layers are trained on the outputs generated at lower levels (Wolpert, 1992).

Writing in 1992 Wolpert (1992) described choosing an algorithm and features for deriving higher level models as black art since no metrics existed for making these choices. Since 1992 several approaches to making these choices have been proposed (Ting and Witten, 1999).

7.3.1 Implementation

For the purpose of my investigation, I decided to use the training predictions generated by the 11 models outlined in section 6.3 as inputs for level 1 models. The training predictions were obtained using 5-fold cross validation. Thus, for every fold 4/5 of the data were used to train the model and predictions were generated on the remaining 1/5. As level 1 classifiers I fitted all algorithms I have used as level 0 classifiers (PDA, NNET, GBM, LINEAR SVM, RF, C5.0, ELASTIC NET) on the 11 predictions generated at level 0. Regarding the question which features to use as input for the meta model, several options were under consideration:

1. Training the meta model solely on the predictions generated at level 0
2. Adding the predictions generated by level 0 classifiers to the training set and training the meta model on the enhanced training set
3. Exhaustively fitting meta models on all possible combinations of level 0 predictions

Contrary to majority voting, where predictions are simply added up, stacking requires fitting a meta model on each of the possible subsets. Exhaustively evaluating all combinations of predictions from level 0 would have resulted in $2^1 1 = 2048$ different meta models. In terms of time and computing power, this was clearly not a feasible option. Similarly, the second option would have given rise to the question which predictions to add to the training set and how they would interact with already existing predictors. Thus, given the constraints in computing power and time, I opted for the first option.

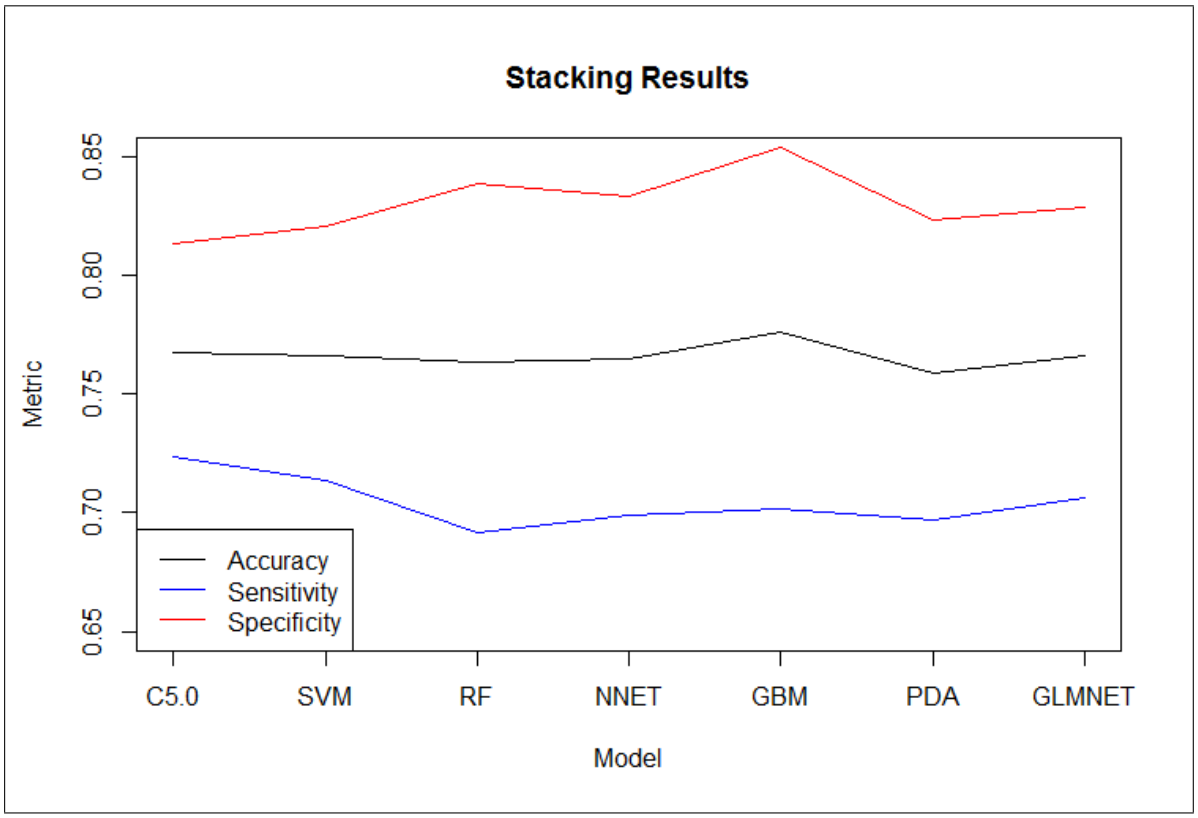

Figure 7.2: Model performance using different level 1 classifiers

7.3.2 Experiments and Results

As figure 7.2 illustrates, the performance of most stacked ensembles roughly equals that of the best single classifiers with stochastic gradient boosting as a meta model achieving the highest classification accuracy of 77.63%. The distribution of sensitivity and specificity is more balanced than in the case of single models. All models achieved sensitivity scores of roughly 70% and specificity scores of between 81% and 86%. Although Stacking does not achieve performance superior to single models in terms of classification accuracy, it does seem to improve model stability. Yet, without evaluating the performance of meta models on subsets of the level 0 predictions, the number of experiments performed is clearly insufficient to establish a final conclusion.

Chapter 8

New Ensemble Strategies

The previous chapter described my implementation of the established ensemble methods majority voting, bagging, and stacking, In this chapter, I turn to Boosting and Bayesian methods, albeit in a different way. Instead of simply implementing established methods and observing their performance on my data, I attempt to create my own ensemble methods primarily by altering and combining ideas from boosting, bayesian learning and majority voting.

8.1 A Hybrid Approach to Boosting

Boosting is one of the most popular advanced ensemble techniques. The general idea is to train many learners of the same algorithm on a dataset and forcing models to focus on those cases that have been misclassified in previous iterations. Finally, one ends up with multiple models that have focused on different parts of the data. Performing majority voting or averaging on the generated results leads to improved performance in many cases(Hastie, 2009).

8.1.1 Adaboost: Background and Problems

AdaBoost developed by Freund et al. (1999) served as a basis for my experiments to construct an ensemble based on boosting. In my experiments, the original Adaboost did not lead to substantial improvements in model performance. This partly stems from the fact that Boosting usually works well on weak classifiers while the performance of my single algorithms was already significantly above the baseline of random guessing. When applying Adaboost I could force my algorithms to correctly classify most of those samples that had been misclassified in previous iterations, but this came at the cost

of misclassifying many observations that had been correct previously. Generally, this wouldn't constitute a large problem since the final result is obtained by a majority vote of all classifiers in which weaker performers have less influence. Yet, the exponential divergence of weights had the effect that models generated in later iterations which misclassified a few heavily weighted samples had less weight in the final vote than previous models which overall performed worse but had more evenly distributed case weights.

8.1.2 Hybrid Boost: A Novel Approach to Boosting

I wanted to investigate whether I could use AdaBoost's powerful ability to force classifiers to focus on previously misclassified samples as a method for increasing the diversity of my existing ensemble while mitigating the described drawbacks. To adapt Adaboost to my needs I needed to transform it from a standalone ensemble algorithm that uses weak classifiers of one type, into a procedure that supplements an existing hybrid ensemble. Accordingly, I developed the following three procedures which differ marginally from each other.

Standalone Hybrid Boost.
D: the dataset split into a training set D_t and a test set D_v
N: number of observations in D_t
ϕ: the ensemble of classifiers
i: index of instances
k: index of boosting iterations
j: index of learners

Select a dataset D and a set of available machine learning algorithms T.

$$T = t_1, t_2, ..., t_j$$

1. Pick any algorithm t from T.

2. Assign equal case weights to every observation in the training data:

$$w_i = \frac{1}{N}, i = 1, 2..., N$$

3. Fit the selected classifier t on the observations with the current weights to obtain the hypothesis $h_t(x_i), i = 1, 2, ...N$

4. Calculate the classification error ϵ_t for the classifier t as follows

$$\epsilon_t = \frac{\sum_{i=1}^{N} w_i \theta_1}{\sum_{i=1}^{N} w_i} - \sum_{i=1}^{N} \theta_2 \tag{8.1.1}$$

where

$$\theta_1 = \begin{cases} 1, & \text{if } h_t(x_i) \neq y_i \\ 0, & \text{otherwise} \end{cases}$$

and

$$\theta_2 = \begin{cases} 1, & \text{if } h_t(x_i) = y_i \wedge y_i \neq h_{t-1}(x_i) \\ 0, & \text{otherwise} \end{cases}$$

5. Calculate the model weight α_t

$$\alpha_t = \log \frac{1 - \epsilon_t}{\epsilon_t} \tag{8.1.2}$$

6. Update the data instance weights for the next iteration

$$w_i = w_i e^{\alpha_t \theta_1} \tag{8.1.3}$$

7. Repeat steps 3-6 k times to generate k boosted models $t_1, t_2, ...t_k$ from algorithm t.

8. Add the boosted models $t_1, t_2, ...t_k$ to the ensemble ϕ.

9. Repeat steps 1-8 for all algorithms in $T = t_1, t_2, ...t_j$.

10. Perform majority voting with all learners in the ensemble ϕ to obtain the final ensemble classifier.

Majority Hybrid Boost.

D: the dataset split into a training set D_t and a test set D_v

N: number of observations in D_t

ϕ: the ensemble of classifiers

i: index of instances

k: index of boosting iterations

j: index of learners

Choose a dataset D and a set of different machine learning algorithms C.

1. Select a subset T of C.

$$T = t_1, t_2, ..., t_j$$

$$\text{where } T \in C$$

2. Follow steps 1 - 9 from "Standalone Hybrid Boost".

3. Pick the algorithms not used for boosting T'

$$\text{where } T' \notin T \in C$$

and train them on the training set D_t to obtain the classifiers $L = l_1, l_2, ..., l_{T'}$.
and add them to the ensemble ϕ.

4. Obtain the final classifier by majority vote of all classifiers in ϕ.

Exhaustive Hybrid Boost.

D: the dataset split into a training set D_t and a test set D_v

N: number of observations in D_t

ϕ: the ensemble of classifiers

i: index of instances

k: index of boosting iterations

j: index of learners

Choose a dataset D and a set of different machine learning algorithms C.

1. Follow steps 1 - 3 from "Majority Hybrid Boost".

2. Generate all possible sub-combinations of classifier predictions and evaluate them on the validation set D_v.

3. Pick the subset that achieves the highest classification accuracy on the validation set.

To establish guidelines for selecting algorithms I attempted to boost all of the 11 models mentioned in section 6.3. I finally settled on C5.0 trees, neural nets and penalized discriminant analysis for the following reasons Firstly, the models generated from these algorithms were relatively different from each other as determined in section 6.3, which ensured greater diversity. Secondly, the models achieved relatively high individual classification accuracy. Furthermore, it is advisable to select unstable algorithms such as neural networks or decision trees for the subset T as boosting on unstable models is likely to achieve greater differences between boosted models and base models. 8.1 indicates that boosting achieved much greater differences between models on neural networks and decision trees as compared to partial discriminant analysis. My approach differed from traditional AdaBoost in three respects. Firstly, the number of boosting iterations was much lower. While Adaboost typically involves up to 1000 iterations (Mease and Wyner, 2008) to achieve model convergence, I used less than 10 iterations on all tests. Secondly, I did not use final model weights in generating the final classification. Thirdly, I added a shrinkage parameter to the weight calculation procedure in step 7 to prevent weights from declining too quickly. The reason for implementing these modifications is that model convergence was not the goal. Rather, I wanted to apply AdaBoost's idea of case weights to generate diverse models from the same algorithm and the same data and in turn increase overall ensemble diversity.

8.1.3 Experiments and Results

In accordance with Freund et al. (1999) Adaboost drove the training error for C5.0 trees down to less than 1 % after just 2 iterations. Yet, it should be noted that the C5.0 tree model already achieved near perfect accuracy on the training set. The plot "Corrected Observations" illustrates how many observations, that had been misclassified in the previous iteration, were corrected. Corrections ranged between 29 and 50 for C5.0 trees while only amounting to 10 in the second iteration for partial discriminant

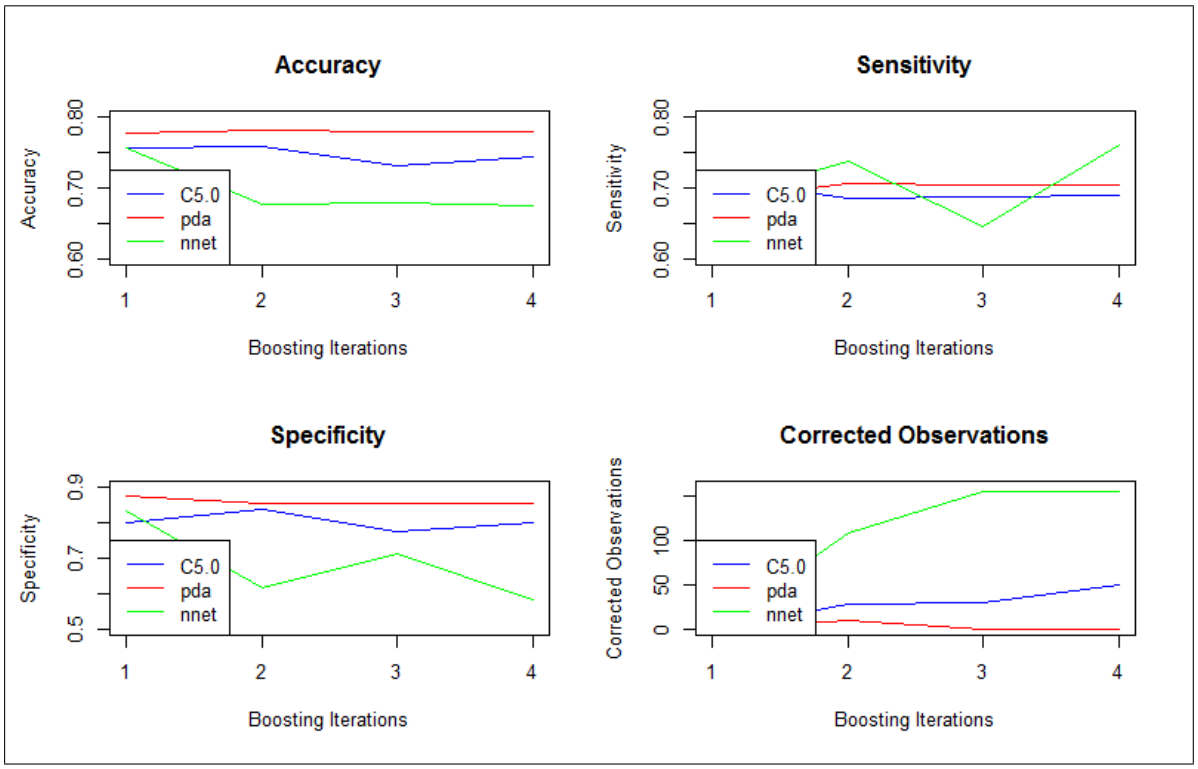

Figure 8.1: Model performance across multiple Boosting iterations

analysis. Neural nets on the other hand corrected up to 155 observations in one boosting iteration resulting in highly diverse models. Yet, these corrections did not result in significant increases in accuracy between models as they came at the cost of misclassifying other observations. On the test set classification accuracy for C5.0 models increased slightly between the first and the second boosting iteration and marginally declined in further iterations. The difference achieved was not significant compared to the single model. For partial discriminant analysis accuracy almost did not change at all whereas for neural networks it went down after the first iteration and remained constant over further iterations. Neural nets are the most flexible, but also the most unstable among the observed algorithms as the weighting produced by boosting resulted in alternations between predictions of high specificity and high sensitivity. Performing further boosting iterations did not result in significant changes. An unweighted majority vote of all the all the boosted models (**Standalone Hybrid Boost**) resulted in roughly 77.25% accuracy which equals the best prediction achieved by the single member Partial Discriminant Analysis. Weighting the classifiers by the weights generated during the boosting procedure resulted in an inferior accuracy of 75.5%. Then I

applied (**Majority Hybrid Boost**) on my established ensemble of 11 models. I generated boosted versions of "PDA" "C5.0" and "NNET" and added the models generated by the first boosting iteration to the ensemble. Thus, the entire ensemble consisted out of 14 classifiers. A simple majority vote of the 14 classification vectors yielded a total accuracy of 83.25% and thus outperformed all previously tried methods. Furthermore, creating all possible combinations of the 14 models (**Exhaustive Hybrid Boost + Majority Vote**) yields $2^{14} = 16384$ possible ensembles. Since this amount of combinations can be generated in reasonable time, I wrote a function that performs an exhaustive search on the ensemble space by using every possible model combination to generate predictions on the test set. The best performance was achieved by 7 base models **GBM (127), GLMNET (127), SVM(127), NB (5), NNET OVERSAMPLED (127), RF (658), RF (127)** and two of the boosted models **Boosted C5.0 (658) and Boosted PDA (127)** yielding 84.12% accuracy.

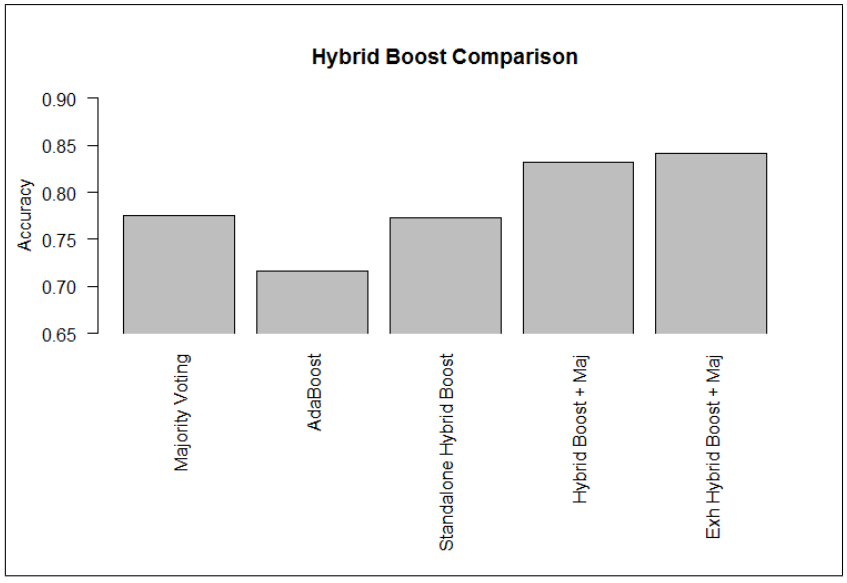

Figure 8.2: Classification accuracies achieved by various boosting strategies

8.2 Bayesian Ensemble Approaches

In previously created ensembles every classifier had equal weight in the final prediction. In this chapter, I explain how I attempted to use a Bayesian approach to weight classifiers in the ensemble which did achieve an increase in final classification accuracy. The fundamental idea underlying Bayesian ensemble methods is to assign weights to

every model proportionate to the probability that the model is the data generating model. The Bayes optimal classifier is a complete ensemble of all possible hypotheses in the model space all weighted by their respective probabilities to be the data generating model (Mitchell, 1997). In theory, this would constitute the global optimal ensemble. Yet, due to computational and time constraints. It is rarely possible to capture all hypotheses in a model space and to determine their respective probabilities in practice.

8.2.1 Background: Bayesian Averaging and Model Combination

To mitigate aforementioned problems Bayesian Averaging samples a limited number of models from the hypothesis space. The distribution of the predicted variable is calculated by averaging the models weighted by their posterior distributions given the data (Madigan et al., 1996).

$$p(y|D) = \sum_{k=1}^{K} p(y|h_k, D)p(h_k|D)$$

where y is the predicted variable, D the data, K the number of models, and h denotes single models. Performing Bayesian Averaging first requires calculating the probabilities of the data given each model at hand $p(h_k|D)$ which is equivalent to the marginal likelihood of the model. When dealing with continuous variables model parameters are marginalized by integrating them out while in a discrete setting the quantities of all parameters across a single model are summed up. The individual posteriors for each model are calculated using Bayes' law

$$p(h_k|D) = \frac{p(d|h_k)p(h_k)}{\sum_{l=1}^{K} p(D|h_l)p(h_l)}$$

where $p(h)$ is the prior probability of the hypothesis which is equal for all generated models H as they have not yet seen the data. As mentioned in the beginning of this chapter Bayesian methods assign probabilities to every model that it is the data generating model (DGM) implying that such a model is assumed to exist in the model space. Even if the data generating model existed in the model space, BMA is unlikely to find it as it only samples a subsection of the hypothesis space. Domingos (2000) argued that the assumption of the existence of a DGM leads BMA to assign disproportionately

higher weights to models with small superiority in accuracy. This in turn often results in the dominance of one model. Accordingly, other ensemble methods like Bagging or Boosting often perform better while being less complicated to implement. For similar reasons Monteith et al. (2011) assert that BMA is actually not an ensemble technique but a model selection method. As an alternative, they proposed a slight modification. Instead of calculating a Bayesian average of single models they proposed a method for calculating the Bayesian average of ensembles of models.

8.2.2 Bayesian Hybrid Boost

In this section, I explain how I adapted the Bayesian model combination approach developed by Monteith et al. (2011) to develop a weighting scheme for Hybrid Boost. Since the task at hand was classification and not regression, averaging was not a suitable option to obtain ensemble predictions as well as the final posterior. Instead, I used a weighted voting approach where $x = \{1, -1\}$ for positive and negative classes. To obtain the posterior probability of a model a uniform class noise model can be assumed (Kearns and Vazirani, 1994) which makes it possible to calculate the Bayesian posteriors as follows. For every observation, a misclassification probability ϵ equal to the overall classification error of the model is assumed. As a result the equation for obtaining the posterior

$$p(h|D) = \frac{p(h)}{p(D)} \prod_{i=1}^{I} p(d_i|h)$$

turns into a statement of proportionality

$$p(h|D) \propto p(h)(1-\epsilon)^r(\epsilon)^{I-r}$$

where r is equal to the number of correctly classified observations(Domingos, 2000) (Monteith et al., 2011). The terms $p(h)$ and $p(D)$ have been ignored as they are equal for every model. The procedure is executed as follows:

Bayesian Hybrid Boost.

D: the dataset split into a training set D_t and a test set D_v

ϕ: the ensemble of classifiers

i: index of instances

e: number of learners in ensemble ϕ

j: index of learners

1. Through the hybrid boosting procedure designed in section 8.1.2 obtain an ensemble ϕ containing e classifiers. Each model in the ensemble ϕ represents a hypothesis h

2. Initialize a vector α of length e so that each element in α equals 1.

3. Draw k vectors $M = m_1, m_2, ..., m_k$ each of length e from a random dirichlet distribution parameterized by α. The observations $W = w_1, w_2, ..., w_e$ in each vector m represent Bayesian priors for each corresponding hypothesis h in the ensemble ϕ.

4. Obtain k ensemble classifications on the training data D_t by applying each vector of priors in M as weights to the ensemble ϕ resulting in k ensembles

$$\phi_1, \phi_2, ..., \phi_k$$

5. For each of the k ensembles obtain predictions on the validation set D_v as follows

$$p(y_i|x_i, \phi) = \begin{cases} 1, & \text{if } \sum_{j=1}^{e} p(y_i|x_i, h_j)w_j > 0 \\ -1, & \text{otherwise} \end{cases} \tag{8.2.1}$$

6. Obtain the classification error ϵ_k and the number of correctly classified samples r_k for each of the k ensembles.

7. Calculate the posterior probability for each of the k ensembles as follows

$$p(\phi|D_v) \propto (1-\epsilon)^r(\epsilon)^{I-r} \tag{8.2.2}$$

8. Update the parameter vector α by adding the vector of priors m, which resulted in the lowest prediction error, to the current priors.

9. Repeat steps 3 - 6 $\frac{k^e}{k}$ times updating α with every iteration resulting in $E = k^e$ differently weighted ensembles

$$\phi_1, \phi_2, ..., \phi_E$$

with posteriors

$$p(\phi_1|D_v), p(\phi_2|D_v), ..., p(\phi_E|D_v)$$

10. Obtain the final prediction across all ensembles weighted by their respective posteriors

$$p(y_i|x_i, E) = \begin{cases} 1, & \text{if } \sum_{n=1}^{E} p(y_i|x_i, \phi_n)p(\phi_n|D_v) > 0 \\ -1, & \text{otherwise} \end{cases} \tag{8.2.3}$$

k should not be larger than 3 as linear growth in k results in exponential growth of the ensemble space E.

8.2.3 Experiments and Results

To implement Bayesian Hybrid Boost I first assembled an ensemble of the aforementioned 11 base models. Then I used the hybrid boosting procedure to obtain boosted versions of the neural net, C5.0 tree classifiers. As the ensemble space grows exponentially with linear increases in the number of classifiers, I only used the models resulting from the first boosting iteration. Accordingly, the hybrid boosting procedure added 2 boosted models to the base ensemble of 11 classifiers. The initial results displayed 84.38% classification accuracy on the test set 79.46% sensitivity and 89.51% specificity. This constitutes a slight improvement over the best accuracy so far achieved by exhaustive Hybrid Boost.

In summary, it turned out that an ensemble of base classifiers and hybrid-boosted models weighted by the Bayesian posterior probability of constituting the data generating model achieved the best performance. Exhaustively searching the space of possible model combinations consisting of base classifiers and boosted models achieved the second place with simply performing majority vote on the boosted models and the base models ranked third. In total, all implementations of hybrid boost outperformed traditional Adaboost on a tree model.

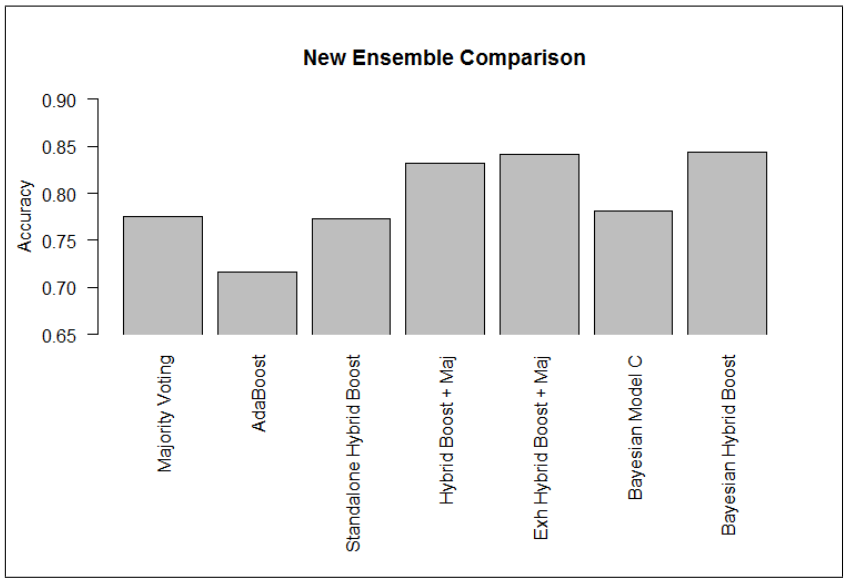

Figure 8.3: Summary of the performance of new ensembles

Chapter 9

Evaluation and Discussion

9.1 Evaluation of Results

The results discussed in the previous two chapters were obtained using the same training and test sets. These were generated by randomly partitioning the Stanford dataset of 2000 tweets into 1200 training- and 800 testing observations. Since the number of observations was comparatively small, and as one training and one test set were used, a large potential for variation in the results exists. Accordingly, performing some steps for evaluating the obtained results was necessary. To investigate the differences between model outputs and to validate the results I took the following steps

1. Testing the statistical significance of model differences.

2. Calculating the number of samples classified differently by different models.

3. Repeating the process of single model and ensemble generation 10 times with different training and test splits.

4. Comparing the performance of models after parameter tuning, feature engineering and applying different sampling strategies to the performance of the untuned base models on the bag of words data set.

5. Testing single models and ensemble strategies on a different data set.

To speed up the process I automated the procedures of feature selection, model tuning, and ensemble construction in one program allowing the user to perform the whole process multiple times in a loop.

9.1.1 Evaluation of Model Differences

In Chapter 6 I trained models using different algorithms, different feature subsets, and oversampling to increase the diversity of the model population. To measure the effectiveness of these steps I used the MCNemar test, a nonparametric test for evaluating the statistical significance of differences in two class predictions. Table 9.1 contains the p-values obtained by applying the McNemar test for comparing two model outputs to each other As is evident from 9.1 the results of the significance test are relatively

	C5.0.preds658	gbm.preds127	glmn.preds127	linsvm.preds127	nb.preds5	nnet.preds127	nnet.preds127.overs	pda.preds127	pda.preds127.overs	rf.preds658	rf.preds127
C5.0.preds658	#N/A	0,9219	0,1814	0,4879	0,0092	0,1508	0,0852	0,0170	0,0620	0,1846	0,3593
gbm.preds127	0,9219	#N/A	0,1561	0,5338	0,0017	0,1489	0,0790	0,0084	0,0554	0,1757	0,2274
glmn.preds127	0,1814	0,1561	#N/A	0,3914	0,0001	0,8955	0,5596	0,0973	0,4764	0,9240	0,0179
linsvm.preds127	0,4879	0,5338	0,3914	#N/A	0,0002	0,3619	0,2301	0,0087	0,1800	0,7016	0,0829
nb.preds5	0,0092	0,0017	0,0001	0,0002	#N/A	0,0000	0,0001	0,0000	0,0000	0,0000	0,0653
nnet.preds127	0,1508	0,1489	0,8955	0,3619	0,0000	#N/A	0,8241	0,2530	0,6750	0,7806	0,0116
nnet.preds127.overs	0,0852	0,0790	0,5596	0,2301	0,0001	0,8241	#N/A	0,4941	0,8875	0,5944	0,0083
pda.preds127	0,0170	0,0084	0,0973	0,0087	0,0000	0,2530	0,4941	#N/A	0,6025	0,2152	0,0006
pda.preds127.overs	0,0620	0,0554	0,4764	0,1800	0,0000	0,6750	0,8875	0,6025	#N/A	0,4879	0,0045
rf.preds658	0,1846	0,1757	0,9240	0,7016	0,0000	0,7806	0,5944	0,2152	0,4879	#N/A	0,0214
rf.preds127	0,3593	0,2274	0,0179	0,0829	0,0653	0,0116	0,0083	0,0006	0,0045	0,0214	#N/A

Figure 9.1: Change in accuracy by feature and model

diverse. Overall, operating at a 0.05% significance level, most classifiers have not been evaluated as being statistically significantly different measured by their predictions on the test set.

Yet, the ability of statistical tests to quantify differences is limited since they only use class totals in the calculation. Thus, the difference of two prediction vectors with roughly equal class frequencies would be considered insignificant, although the underlying models may have generated their predictions on different samples. Accordingly, I evaluated the total number of samples classified differently d by every pair of models as

$$d = \sum_{n=1}^{N} \phi \tag{9.1.1}$$

where

$$\phi = \begin{cases} 1, & \text{if } x_n \neq z_n \\ 0, & \text{otherwise} \end{cases}$$

where x and z denote the outputs of two different classifiers and N is the total number of observations. The results are summarized in table 9.2.

The observations in table 9.2 testify to the limitations of statistical tests in comparing classification outputs. For example the comparison between "C5.0preds658" and

	C5.0.preds658	gbm.preds127	glmn.preds127	linsvm.preds127	nb.preds5	nnet.preds127	nnet.preds127.overs	pda.preds127	pda.preds127.overs	rf.preds658	rf.preds127
C5.0.preds6	0	104	126	133	213	140	135	128	139	96	144
gbm.preds1	104	0	84	93	163	108	105	90	109	66	116
glmn.preds	126	84	0	49	181	58	47	44	71	110	130
linsvm.prec	133	93	49	0	154	77	84	47	94	109	133
nb.preds5	213	163	181	154	0	173	220	161	216	153	199
nnet.preds	140	108	58	77	173	0	81	62	91	116	132
nnet.preds	135	105	47	84	220	81	0	77	50	127	147
pda.preds1	128	90	44	47	161	62	77	0	59	110	130
pda.preds1	139	109	71	94	216	91	50	59	0	133	143
rf.preds658	96	66	110	109	153	116	127	110	133	0	118
rf.preds127	144	116	130	133	199	132	147	130	143	118	0

Figure 9.2: Number of Observations classified differently per model pair

"gbm.preds127" via the McNemar test produced a p-value of 0.92. Thus, the difference was judged to be highly insignificant, yet on 104 out of 800 test samples did the classifiers produce different predictions. Although the statistical test did not indicate a significant difference, the absolute difference in predicted samples as well as the ensemble construction results obtained in chapter 7 indicate that models were sufficiently diverse from each other to justify combining their predictions in an ensemble.

9.1.2 Evaluation of Single Models

To conclusively determine whether the steps taken in all previous chapter did, in fact, yield better results, I had to compare the results of single tuned models and the constructed ensembles against a benchmark. For establishing a benchmark of base performance I fitted the algorithms I used during ensemble construction on the simple bag of words data set. No feature selection, feature construction or model parameter tuning was performed. Then I added my constructed features and fitted the base models without parameter tuning on the feature enhanced data. To minimize the potential that results were due to chance, I generated every model 10 times using different random training/test splits with every iteration.

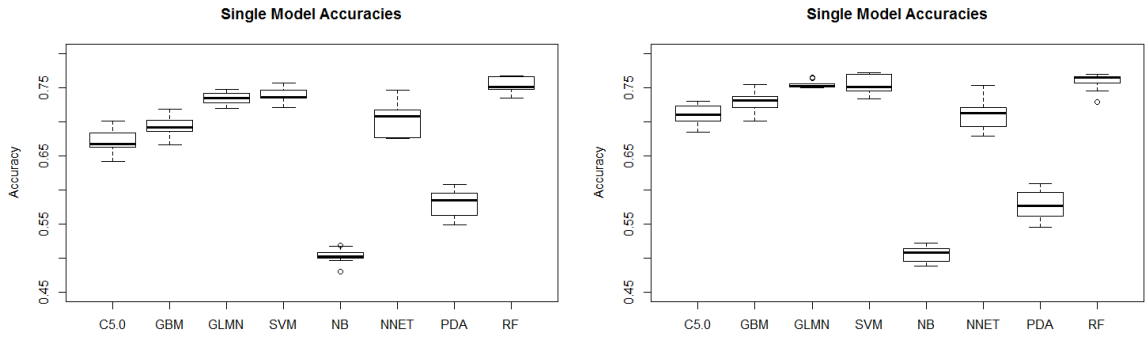

Figure 9.3: Distribution of base accuracies of default models fitted on the simple bag of words dataset (left) and on the feature enhanced dataset (right) for 10 different training/test splits

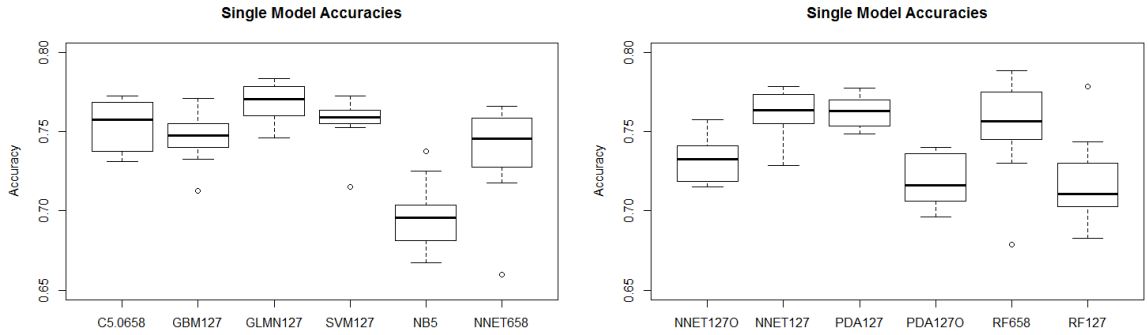

Figure 9.4: Distribution of single tuned algorithm accuracies on the feature enhanced data set using 10 different training/test splits

Judging from the summary of the results displayed in the plots it depends largely on the model how much and if at all adding the manually constructed features had an impact. For example, the performance of C5.0 trees or general boosting machines changed markedly while that of support vector machines was only marginally affected. Lastly, I evaluated the fully tuned models on the feature enhanced dataset.

Comparing the distribution of accuracies attained by base models against those of tuned and feature enhanced models, it is evident that the latter consistently performed better. However, different algorithms reacted differently to the steps taken to enhance their performance. C5.0 trees, partial discriminant analysis, elastic nets and neural networks benefitted greatly from parameter tuning. General boosting machines and linear support vector machines saw slight improvements in response to parameter selection. For random forests the main factor appears to be the number of features. They seem to benefit the most from high dimensionality while also dealing well with

feature correlation. Similar observations have already been made in section 5.2. Most other algorithms require smaller subsets with lower correlations and a higher emphasis on parameter tuning. Naive Bayes, as has been observed in section 6.2.4, reverted to simply predicting one class when confronted with high dimensionality.

9.1.3 Evaluation of Ensemble Strategies

Next, it needs to be investigated whether the application of the various ensemble strategies developed in chapter 7 achieved significant improvements over single algorithms. When I evaluated the results of my experiments with bagging in section 7.2 I tried to run the procedure 10 times in a loop as I had done it with previous strategies. Unfortunately, the program regularly crashed both on the HPC and on a Windows 10. Thus, I finally decided to run every iteration manually. Due to the time and computing constraints involved, I trained every algorithm only on 4 bootstrap samples instead of 10. The results confirmed the conclusion drawn in chapter 7.2 that in the case of

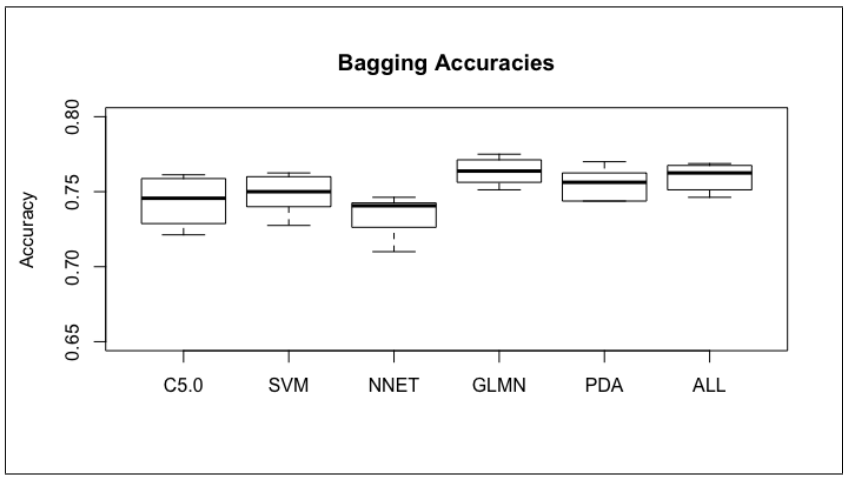

Figure 9.5: Distribution bagged model accuracies on the feature enhanced data set using 10 different training/test splits

this project bagging did not lead to improvements in model performance. Similar to bagging, stacking did not achieve any significant improvements over single algorithms. Yet, the validity of this conclusion is limited, as only a small number of experiments has been performed. Training meta models on exhaustive combinations of sub-models and potentially adding the predictions generated by layer zero models as features to the original training data would be necessary to obtain conclusive evidence.

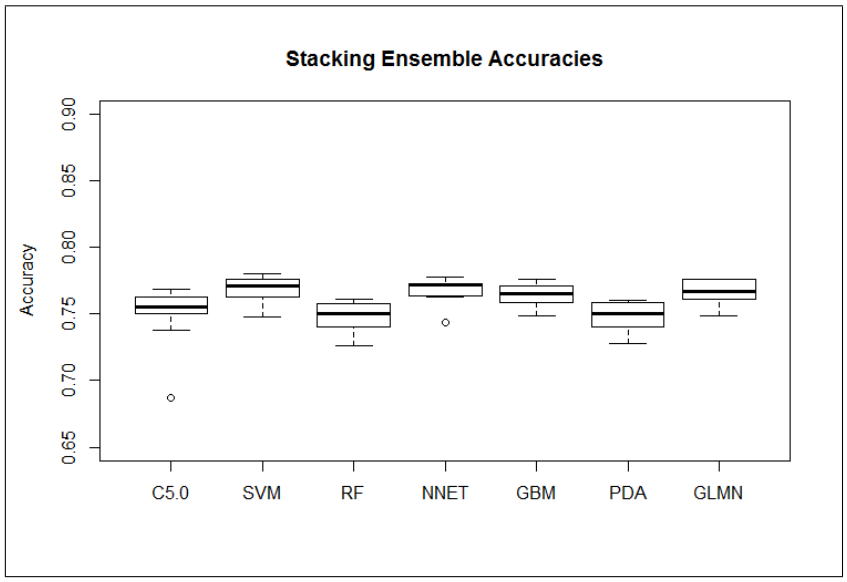

Figure 9.6: Distribution of stacking accuracies on the feature enhanced data set using 10 different training/test splits

Figure 9.7 shows the distribution of the accuracies achieved by majority voting, AdaBoost, Standalone Hybrid Boost, Majority Hybrid Boost, Bayesian model combination and Bayesian Hybrid Boost over 10 different training/test splits.

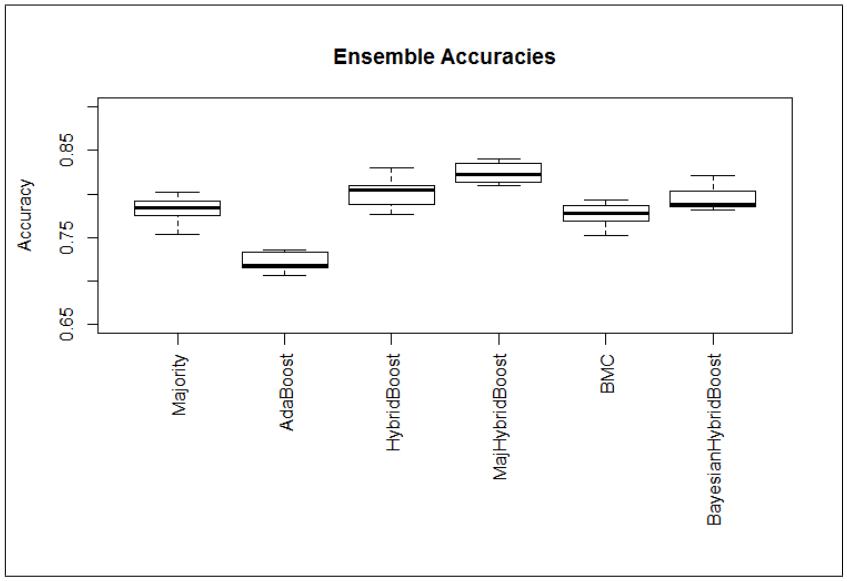

Figure 9.7: Distribution of accuracies obtained through different ensemble strategies on the feature enhanced data set using 10 different training/test splits

It turns out that the initial training test split was favorable to majority voting and Bayesian Hybrid Boost which achieved much higher accuracies on the initial data set than the averages over 10 random training/test splits. Standalone Hybrid Boost, on the other hand, was underestimated as the mean accuracy was above 80% compared

to 77.25% on the initial split. Overall it is clear that the methods that made use of the boosting procedure developed in 8.1 **Hybrid Boost**, **Majority Hybrid Boost** and **Bayesian Hybrid Boost Boost** performed better in terms of overall accuracy than the established methods **Majority** (Voting) and **Bayesian** (Model Combination).

9.1.4 Performance on Different Data

To ensure that results were purely due to a bias in data selection, I repeated most of the previously described experiments on the Arabic language tweet collection by Nabil et al. (2015b). The lack of suitable algorithms for parts of speech tagging and dealing with the encoding of Non-Western characters significantly slowed down my progress compared to English. I wrote separate programs in R that process simplified Chinese characters and Arabic letters and processed the set of Arabic tweets into a document term matrix. However, I did not create additional features due to the lack of appropriate packages. Since I performed feature selection and trained models on a bag of words using the 63 most important terms as predictors base performance was much lower compared to the Stanford dataset I trained models on 2000 tweets applying a 60/40 split using C5.0 trees, elastic net, general boosting machines, partial discriminant analysis, support vector machines, random forests, and neural networks. Furthermore, I used those models to create ensembles applying previously described strategies.

The results over ten random training/test splits are summarized in figure 9.8.

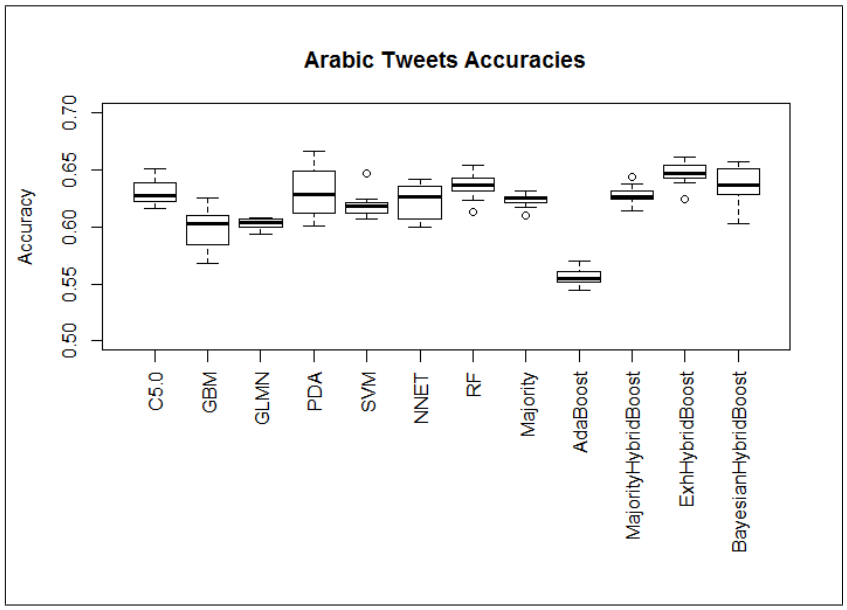

Figure 9.8: Distribution of accuracies obtained through different ensemble strategies on the feature enhanced data set using 10 different training/test splits

Most single algorithms achieved classification accuracies between 60% and 63%. Random forest performed best attaining an average of 63.8%. It was outperformed by Bayesian Hybrid Boost with a mean of 64% and Exhaustive Hybrid Boost which attained a mean of 65.10% classification accuracy. Thus, except for majority hybrid boost, the ensembles based on hybrid boost outperformed all single methods, albeit by a much smaller margin than on the Stanford dataset.

9.2 Discussion of Results

Although the scope of research dealing with the application of ensemble methods to sentiment classification is limited, some works do exist, which merit comparison. Wang et al. (2014) evaluated the performance of Bagging, Boosting and Random Subspace on 10 public datasets using naive Bayes, decision trees, maximum entropy, KNN and support vector machines. Their approach differed from mine as they constructed homogeneous ensembles for each individual classifier. The application of ensemble techniques resulted in an average improvement of 5-8% in classification accuracy, with random subspace support vector machine ensembles achieving the best results on average. Accordingly, their performance improvements approximately equal those achieved

by exhaustive hybrid boost and majority hybrid boost. But given the use of different datasets and different ensemble strategies the approaches can hardly be compared. Wilson et al. (2006) applied boosting to a sentiment analysis task. They used short documents of several sentences about current events instead of tweets. Their baseline for comparison was a classifier that always predicted the most frequent class. Using boosting they achieved 23-96% improvements in accuracy. As my data was balanced between two classes, a baseline classifier would achieve 50% accuracy. The best results attained by exhaustive hybrid boost constitute an improvement of roughly 66% over that baseline. It needs to be mentioned though that Wilson et al. (2005) used multi-class sentiment datasets, thus their baseline was lower than 50%. Among the examined previous research, the one that most closely resembles my work has been conducted by Whitehead and Yaeger (2010). They used bagging, boosting and random subspace ensembles of support vector machines on 5 datasets and compared the obtained results against those achieved by single SVMs. Depending on the data and the model chosen they found that ensembles obtained improvements between 1-3% over single models. The results obtained are consistent with my findings as traditional AdaBoost achieved lower accuracy on some datasets than single models. Notably, most of the previous papers experienced their greatest success in applying random subspace ensembles, which I haven't used. In response to the research question formulated in section 1.3 the following answers can be provided. The results obtained throughout the dissertation and evaluated in the last section testify that the application of ensemble methods to sentiment classification problems can indeed significantly increase classification accuracy compared to single algorithms. However, significant improvements were almost exclusively achieved by the ensemble procedures I developed in chapter 8. The application of established methods, as the results obtained in chapter 7 illustrate, did not produce improvements when applied to sentiment classification.

One possible explanation is that many ensemble procedures such as bagging achieve their success by reducing variance (Breiman, 1996). Hansen and Salamon (1990) demonstrated that generalization error can be reduced by constructing ensembles of similar neural networks. In those cases the success of ensembles derived primarily from the number of learners. This approach seems especially promising in regression settings

where ensemble strategies such as bagging can have a significant impact on the final result by reducing the variance produced by single learners. In binary classification the result can only vary between two outcomes, thus the purpose of ensembling as a means for variance reduction is more limited. Algorithms used in my custom ensembles were carefully trained and adjusted through feature engineering, sampling techniques and parameter tuning using the Stanford dataset. The improvements in performance are the result of a long process taking account of the specific challenges presented by the data. On the set of Arabic language tweets the results, while still favoring the newly developed ensemble strategies, were not as explicit. A data bias can therefore not be entirely excluded. However, it appears that due to the complexity and ambiguity presented by human language, the success of experiments involving human language depends to a great extent on the design of experiments, the data at hand and the development of appropriate and customized strategies. Furthermore, my own research as well as that of Wilson et al. (2006) and Whitehead and Yaeger (2010) demonstrates that boosting has a high potential for achieving significant performance improvements, although just implementing standard procedures may not be able to unlock this potential. With regard to research questions 2 and 3 formulated in section 1.3, it can thus be concluded that a large potential for improving the performance of ensemble techniques exists either through the adaption of existing strategies or through the development of new ones.

Chapter 10

Conclusion

10.1 Conclusion

In this work I demonstrated my experiments with machine learning ensemble techniques to extract sentiments from tweets. I finally developed two novel ensemble strategies which outperformed established methods on the problem at hand. The obtained results motivated me to argue in section 9.2 that the success of ensemble learning in the field of sentiment classification requires more customization to individual data requirements than other machine learning problems which may explain their lack of popularity in this domain. Nevertheless, the potential of ensemble learning to enhance the accuracy of sentiment classifiers exists. Coming years are likely to witness major changes in the practice of machine learning. On the one hand, these changes derive from the rise of technologies such as deep learning and the increase of computing power particularly in connection with quantum computing. However, there is also an ongoing trend towards automatizing tasks such as feature construction or the selection of appropriate modeling strategies which currently still require human judgment. Nowadays, ensemble learning for sentiment classification may not be very popular outside of academia due to the large amount of human effort required. But the prospect of automation coupled with the new possibilities in natural language processing presented by deep learning, promise to make ensemble techniques for processing human language more popular among industry practitioners in the near future.

10.2 Suggestion for Further Work

For the experiments discussed in this dissertation, I always used 11 base models and up to 3 boosted models generated through the hybrid boosting procedure. In my evaluation, I concluded that Exhaustive Hybrid Boost overall performed best. However, in experimental results with smaller ensemble employing only 8-10 learners, Bayesian Hybrid Boost consistently achieved the lowest classification error. Due to time constraints, I did not test these smaller ensembles rigorously enough to include the results in the dissertation. An interesting direction for further work would be to investigate the impact of the number of learners in an ensemble on the performance of the developed strategies. Furthermore, using the newly developed strategies in regression settings and on other datasets outside the sentiment classification domain would provide interesting avenues for research. Lastly, Hybrid Boost could be further enhanced by various means such as investigating different weighting strategies for classifiers.

Bibliography

Abbasi, A., Chen, H., and Salem, A. (2008). Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. *ACM Transactions on Information Systems (TOIS)*, 26(3):12.

Abdul-Mageed, M., Diab, M. T., and Korayem, M. (2011). Subjectivity and sentiment analysis of modern standard arabic. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 587–591. Association for Computational Linguistics.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bollen, J., Mao, H., and Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8.

Bouchet-Valat, M. (2014). *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. R package version 0.5.1.

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

Davidov, D., Tsur, O., and Rappoport, A. (2010). Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd international conference on computational linguistics: posters*, pages 241–249. Association for Computational Linguistics.

Dietterich, T. G. et al. (2000). Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15.

Ding, X. and Liu, B. (2010). Resolving object and attribute coreference in opinion mining. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 268–276. Association for Computational Linguistics.

Ding, X., Liu, B., and Yu, P. S. (2008). A holistic lexicon-based approach to opinion mining. In *Proceedings of the 2008 international conference on web search and data mining*, pages 231–240. ACM.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. In *ICML*, volume 2000, pages 223–230.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Feinerer, I. and Hornik, K. (2015). *tm: Text Mining Package*. R package version 0.6-1.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

from Jed Wing, M. K. C., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., and Scrucca., L. (2015). *caret: Classification and Regression Training*. R package version 6.0-47.

Gentry, J. (2015). *twitteR: R Based Twitter Client*. R package version 1.1.8.

Go, A., Bhayani, R., and Huang, L. (2009a). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12).

Go, A., Bhayani, R., and Huang, L. (2009b). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12).

Greenwood, P. E. (1996). *A guide to chi-squared testing*. Wiley, New York.

Grün, B. and Hornik, K. (2011). topicmodels: An R package for fitting topic models. *Journal of Statistical Software*, 40(13):1–30.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.

Hastie, T. (2009). *The elements of statistical learning : data mining, inference, and prediction*. Springer, New York.

Hastie, T., Buja, A., and Tibshirani, R. (1995). Penalized discriminant analysis. *The Annals of Statistics*, pages 73–102.

Hatzivassiloglou, V. and Wiebe, J. M. (2000). Effects of adjective orientation and gradability on sentence subjectivity. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 299–305. Association for Computational Linguistics.

Hornik, K. (2015a). *NLP: Natural Language Processing Infrastructure*. R package version 0.1-7.

Hornik, K. (2015b). *openNLP: Apache OpenNLP Tools Interface*. R package version 0.2-5.

Hornik, K., Buchta, C., and Zeileis, A. (2009). Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232.

Hornik, K., Meyer, D., and Buchta, C. (2014). *slam: Sparse Lightweight Arrays and Matrices*. R package version 0.1-32.

Jurka, T. P., Collingwood, L., Boydstun, A. E., Grossman, E., and van Atteveldt, W. (2014). *RTextTools: Automatic Text Classification via Supervised Learning*. R package version 1.4.2.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20.

Kearns, M. J. and Vazirani, U. V. (1994). *An introduction to computational learning theory*. MIT press.

Kim, S.-M. and Hovy, E. (2006). Extracting opinions, opinion holders, and topics expressed in online news media text. In *Proceedings of the Workshop on Sentiment and Subjectivity in Text*, pages 1–8. Association for Computational Linguistics.

Kuhn, M. (2013). *Applied predictive modeling*. Springer, New York.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

Liu, B., Hu, M., and Cheng, J. (2005). Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, pages 342–351. ACM.

Madigan, D., Raftery, A. E., Volinsky, C., and Hoeting, J. (1996). Bayesian model averaging. In *Proceedings of the AAAI Workshop on Integrating Multiple Learned Models, Portland, OR*, pages 77–83.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., and Hornik, K. (2014). *cluster: Cluster Analysis Basics and Extensions*. R package version 1.15.2 — For new features, see the 'Changelog' file (in the package source).

Mayer, Z. A. and Knowles, J. E. (2015). *caretEnsemble: Ensembles of Caret Models*. R package version 1.0.0.

Mease, D. and Wyner, A. (2008). Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, 9(Feb):131–156.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2014). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-4.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York.

Monteith, K., Carroll, J. L., Seppi, K., and Martinez, T. (2011). Turning bayesian model averaging into bayesian model combination. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2657–2663. IEEE.

Nabil, M., Aly, M. A., and Atiya, A. F. (2015a). Astd: Arabic sentiment tweets dataset. In *EMNLP*, pages 2515–2519.

Nabil, M., Aly, M. A., and Atiya, A. F. (2015b). Astd: Arabic sentiment tweets dataset. In *EMNLP*, pages 2515–2519.

OpenNLP, A. (2011). Apache software foundation. *URL http://opennlp. apache. org*.

Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10.

Pang, B., Lee, L., et al. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Rennie, J. D., Shih, L., Teevan, J., and Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 616–623.

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.-C., and Mueller, M. (2011). proc: an open-source package for r and s+ to analyze and compare roc curves. *BMC Bioinformatics*, 12:77.

Saif, H., He, Y., and Alani, H. (2012). Alleviating data sparsity for twitter sentiment analysis. CEUR Workshop Proceedings (CEUR-WS. org).

Salzberg, S. L. (1994). C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240.

Shoukry, A. and Rafea, A. (2012). Sentence-level arabic sentiment analysis. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 546–550. IEEE.

Ting, K. M. and Witten, I. H. (1999). Issues in stacked generalization. *J. Artif. Intell. Res.(JAIR)*, 10:271–289.

Wang, G., Sun, J., Ma, J., Xu, K., and Gu, J. (2014). Sentiment classification: The contribution of ensemble learning. *Decision support systems*, 57:77–93.

Wang, H., Lu, Y., and Zhai, C. (2010). Latent aspect rating analysis on review text data: a rating regression approach. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 783–792. ACM.

Whitehead, M. and Yaeger, L. (2010). Sentiment mining using ensemble classification models. In *Innovations and advances in computer sciences and engineering*, pages 509–514. Springer.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wickham, H. (2011). The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29.

Wickham, H. (2015a). *readxl: Read Excel Files*. R package version 0.1.0.

Wickham, H. (2015b). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.0.0.

Wilson, T., Wiebe, J., and Hoffmann, P. (2005). Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics.

Wilson, T., Wiebe, J., and Hwa, R. (2006). Recognizing strong and weak opinion clauses. *Computational Intelligence*, 22(2):73–99.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.