A Framework for Ensemble Predictive Modeling

by

Tarek Abdunabi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2016

Abstract

Ensemble systems have been successfully applied in many fields, such as finance, bioinformatics, medicine, cheminformatics, manufacturing, geography, information security, information retrieval, image retrieval, and recommender systems. The ultimate objective of an ensemble system is to produce better predictions by combining the approximations of different classifiers/models. However, the ensemble performance depends on three main design features. Firstly, the diversity/independence of the base models/classifiers. If all models/classifiers produce similar/correlated predictions, then combining those predictions will not provide any improvement. Diversity is considered to be a key design feature of any successful ensemble system. Secondly, the fusion topology, namely, the selection of a representative topology. Thirdly, the fusion function, namely, the selection of a suitable function. Accordingly, building an effective ensemble system is a complex and challenging process, which requires intuition and deep knowledge of the problem context, and a well-defined predictive modeling process.

Although several taxonomies have been reported in the literature, which aim to categorize ensemble systems from the system's designer point of view, there are still important research gaps need to be addressed. First, a comprehensive framework for developing ensemble systems is not yet available. Second, several strategies have been proposed to inject model diversity in the ensemble; however, there is a shortage of empirical studies that compare the effectiveness of these strategies. Third, most of the ensemble systems research has concentrated on simple problems, and relatively small/low-dimensional data sets. Further experimental research is required to investigate the application of ensemble systems to large and/or high-dimensional data sets, with a variety of data types.

This research attempts to fill these gaps. First, the thesis proposes a framework for ensemble predictive modeling. It coins the term "ensemble predictive modeling" to refer to the process of developing ensemble systems. Second, the thesis empirically compares several diversity injection strategies. Third, the thesis validates the proposed framework using two real-world, large/high-dimensional, regression and classification case studies. The empirical results indicate the effectiveness of the proposed framework.

Acknowledgements

After a long and intense period, today is the day to put the finishing touch on my thesis. It has been a rewarding and enlightening experience for me, and I would like to reflect on the people who have supported me throughout this journey.

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Otman Basir for his wisdom, patience, motivation, enthusiasm, immense knowledge, and continuous support. I express my deepest appreciation for granting me the freedom to explore new ideas. His invaluable guidance, keen interest, and continuous encouragement helped me during my research.

I would like to thank the members of my doctoral committee: Prof. Mahmoud El-Sakka, Prof. Keith Hipel, Prof. Fakhri Karray, and Prof. Ladan Tahvildari for their encouragement, and insightful comments.

My sincere thanks also go to Prof. Roger D. Peng, Prof. Brian Caffo, and Prof. Jeff Leek from Johns Hopkins University, USA, for their valuable "*Specialization Certificate in Data Science*", which has equipped me with the hands-on skills to perform my research. Special thanks to Prof. Andrew Ng from Stanford University, USA, for his excellent machine learning online course. Needless to say, I owe many thanks to the open source communities and Programming forums, especially the *R* and *Python* communities, *Kaggle.com*, *stackoverflow.com*, and *stackexchange.com*. They have taught me many industry-level skills.

I am grateful to Prof. Mike Holcombe and Prof. Jon Barker from the University of Sheffield, UK, for their continuous support and encouragement. Also, I would like to thank my colleagues at the Pattern Analysis and Machine Intelligence (PAMI) Center, University of Waterloo, for the stimulating discussions, and excellent environment. Many thanks go to my fellow scuba divers at the Tri-city Scuba Center for the memorable diving trips, which helped me take time out and recharge.

I would like to extend my sincere thanks to the Libyan people for fully sponsoring my graduate studies, and providing me this opportunity.

Last but not least, I would like to express my deep sense of gratitude to my family, and friends, in Canada and back home, for their unconditional support, encouragement, and best wishes.

Dedication

To the memory of my beloved mother, for her prayers for me. May her soul rest in peace.

To my father, for teaching me the value of hard work, and being self-reliant.

To my brothers and sisters, for their unconditional love and support.

To my close friends, for the wonderful company.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

We live in the age of data, where many things around us are connected to data sources, and many aspects of our daily lives are captured and stored digitally. We are surrounded by an ever-expanding sea of data fed from multitudes of sources, including: social networks, video, news, smartphones data, customer transactions, stock markets data, economic data, weather data, geospatial data, machine-generated data, healthcare records, scientific publications, government records, and the list goes on.

We often make decisions based on the information. In some cases, we have factual, objective data, such as our current financial situation. Other times we rely on intuition and experience such as the experience we use in making decisions in avoiding specific routes during icy weather; or that we use in making a decision to perform a medical test based on our parents' medical history. In either case, we are making decisions based on the predicted future events given the information and experience we currently have. As the available data/information is growing exponentially with each passing day, the opportunities to make better decisions to improve all aspects of our lives is also growing exponentially. Given the inability of the human brain to process huge, diverse, and/or high-dimensional data, we turn to computer systems to aid in our decision making. The process of developing these kinds of systems has evolved in fields such as machine learning, statistical learning, artificial intelligence, data mining, pattern recognition, chemistry, biology, knowledge discovery, predictive analytics, and more recently, data science. Although each field approaches the problem from different perspectives and using different tools, they all share the same ultimate objective of making accurate predictions [45].

Predictive modeling is commonly used to refer to the process of developing a mathematical model, using a learning algorithm, to approximate the relationship between a target, response, or dependent variable and various predictors or independent variables. The developed model is then used to predict future, unknown, values of the target variable [19, 45]. Depending on the type of the target variable, numerical/continuous or discrete/categorical, the problem is, respectively, called a regression or classification problem. However, systems designers are faced with two limitations, which often affect the proper approximation of the relationship. Firstly, real-world data sets often contain a substantial quantity of noise (e.g. errors, uninformative or highly correlated predictions), which can mislead the learning algorithm and produce non-optimal or wrong approximations. Secondly, most learning algorithms have limitations of their operations. So, it is possible that the model space considered by the learning algorithm for the problem does not contain the optimal model. As a result of these limitations, the building of a perfect model/classifier for any given problem is often impossible. On the other hand, different learning algorithms vary in their interpretations of the data and noise, which may lead to different approximations of the relationship between the target and its predictors. This diversity between learning algorithms had resulted in the development of *ensemble systems* [68].

In the literature, the term *ensemble* is often used to refer to a collection of classifiers/models built using the same learning algorithm (e.g., random forest and stochastic gradient boosting machines). On the other hand, the term *multi-classifier system* is used to refer to a collection of classifiers built using different learning algorithms [73]. Nevertheless, this research uses the term *ensemble* to refer to both kinds of systems.

The ultimate objective of an ensemble system is to produce better predictions by fusing/combining the approximations of different classifiers/models. Ensemble systems have been successfully applied in many fields such as: finance [51], bioinformatics [88], medicine [56], cheminformatics [59], manufacturing [55, 72, 74], geography [15], information security [58, 61], information retrieval [23, 25, 75, 76], image retrieval [52, 89], and recommender systems [79]. Intuitively, the combination of different models/classifiers should produce better results than a single model/classifier. However, this depends on three main design features. Firstly, the diversity/independence of the base models/classifiers. If all models/classifiers produce similar/correlated predictions, then combining those predictions will not provide any improvement. Diversity is considered to be a key design feature of any successful ensemble system. Secondly, the fusion topology, namely, the selection of a representative topology for model fusion. Thirdly, the fusion function, namely, the selection of a suitable function [68]. Accordingly, building an effective ensemble system is a complex and challenging process, which requires intuition and deep knowledge of the problem context, and well defined predictive modeling process [45].

Although several taxonomies have been reported in the literature [14, 22, 38, 42, 46, 73, 81, 90], which aim to categorize ensemble systems from the system's designer point of view, there are still research gaps need to be addressed. First, a comprehensive framework for developing ensemble systems is not yet available [77]. Moreover, some of the proposed design methods, such as "overproduce and choose" (also called "test and select") [84], become unfeasible for large and/or high-dimensional data sets. Second, several strategies have been proposed to inject model diversity in the ensemble; however, there is a shortage of empirical studies that compare the effectiveness of these strategies. Third, most of the ensemble systems research has concentrated on simple problems and relatively small/low-dimensional data sets. Further experimental research is required to investigate the application of ensemble systems to large and/or high-dimensional data sets, with a variety of data types, such as time-series, text, multimedia, etc [73].

1.2 Objective

The primary objective of this thesis is to propose a comprehensive framework for ensemble predictive modeling, which can be used to develop ensemble systems for large and/or high-dimensional data and a wide range of applications. The thesis coins the term "ensemble predictive modeling" to refer to the process of developing ensemble systems. The effectiveness of the proposed framework is validated by two real-world, high-dimensional, regression and classification case studies. The proposed framework will address the following main design issues:

- **Data preparation**: data preparation has a significant impact on the predictive ability of a model, especially for regression applications [45]. A typical data preparation phase involves exploratory data analysis, feature engineering, data pre-processing, and data partition. Despite its importance, data preparation is overlooked by the ensemble design methods reported in the literature.

- **Injecting model diversity**: for an ensemble system to be successful, the errors made by its classifiers/models should not be highly correlated. Accordingly, model diversity is considered to be a key design feature of any ensemble system [30, 46, 68, 73, 77]. Model diversity can be systematically injected at three levels, namely, at data, feature, and learning algorithm levels [46].

- **Model selection**: in the context of ensemble predictive modeling, model selection may be classified into three types. First, the selection of the ensemble size, how many base models/classifiers should be trained. Second, the selection of the optimal hyper-parameters for the learning algorithms over a set of candidate values. Third, the selection of the learning algorithms to use over others.

- **Model evaluation**: model evaluation is of paramount importance in any predictive modeling task. It becomes even more important in ensemble predictive modeling, where the relative performance and diversity of models/classifiers must be thoroughly evaluated. Typically, some measure of accuracy is used to assess the effectiveness of a model/classifier. However, accuracy can be measured using different ways, each with its subtle difference. Relying solely on a single metric to evaluate models/classifiers, and understand their strengths and weaknesses, is problematic [45].
- **Model fusion**: once a set of diverse and accurate models/classifiers are built, an appropriate fusion strategy should be selected to obtain the optimal ensemble performance. Model fusion involves two main design decisions, namely, the selection of the fusion topology, and the selection of fusion function.
- **Fusion evaluation**: similar to the model evaluation, model fusion should be thoroughly evaluated. In addition to the ensemble performance, several aspects should be assessed. Such aspects include: required computing resources, model selection and training time (needed to build all base models/classifiers and the fusion model/classifier), and prediction time (required to produce a final prediction by the ensemble for a new sample) [73].
- **Experimental design**: during the model selection phase, several models/classifiers are experimentally compared, based on their performance. In order to have an accurate comparison, it is imperative to account for sources of variation. Such sources may include: the choice of the testing/training datasets, the internal randomness of the training algorithm, and the random classification error [20, 46]. In addition to these sources of variation, it is also important to account for the randomness in parallel computing, if the model selection is performed on a cluster of computers/cores.

As secondary objectives, this thesis aims to empirically investigate the following issues:

- **Evaluating the effectiveness of model diversity strategies**: several strategies have been proposed in the literature to inject model diversity in the ensemble, at data, feature, and learning algorithm levels; however, it is not clear which approach(s) leads to the optimal model diversity, and consequently, to the optimal ensemble performance [46]. The thesis aspires to provide empirical evidences on the relative effectiveness of these strategies.

- **The application of ensemble systems to large/high-dimensional data applications**: most of the ensemble systems research has concentrated on simple applications, and relatively small/low-dimensional data sets [73]. This research aims to validate the proposed framework using two real-world, large/high-dimensional, regression and classification case studies.

- **Investigating the relationship between diversity metrics and ensembles performance**: several diversity metrics have been proposed in the literature to measure the diversity of the ensembles classifiers. This research aims to empirically assess the relationship between these metrics and the performance of the ensembles.

- **Evaluating the scalability of learning algorithms**: scalability is one of the critical issues in large-scale data analysis and modeling. For example, if a learning algorithm works well for a data set with 30 variables and 10,000 samples, is it still going to perform well for a data set with 300 or 3000 variables and 100,000 samples?. Using two real-world, high-dimensional data sets, and Amazon EC2 instance with 32 cores, this research attempts to experimentally evaluate the effect of "the curse of dimensionality" and sample size on model selection, model performance, and computational cost of several learning algorithms.

1.3 Organization

This thesis is organized as follows: Chapter 2 provides a comprehensive background to ensemble systems from the system's designer point of view. The proposed framework for ensemble predictive modeling is introduced in Chapter 3. To validate its effectiveness, the proposed framework is applied to develop ensemble systems for two, regression and classification, case studies. The regression case study, to predict the stock market's short-term behavior following liquidity shocks, is presented in Chapter 4. Chapter 5 describes the classification case study, to predict the biological response of molecules from their chemical properties. Finally, the conclusion and future work are presented in Chapter 6.

Chapter 2

Background and Literature Review

2.1 Introduction

In supervised learning, a learning algorithm attempts to build a mathematical model to represent the relationship between the target variable and the independent variables. The developed model is then used to predict future, unknown, values of the target variable [19, 45]. Depending on the type of the target variable, numerical/continuous or discrete/categorical, the problem is, respectively, called a regression or classification problem. Several factors affect the proper approximation of the relationship; such factors may include: a substantial quantity of noise in the training data set (e.g. errors, uninformative or highly correlated predictions), and the limitations of the used learning algorithm. Given that different learning algorithms vary in their interpretations of the data and noise, various approximations of the relationship between the target and its predictors may be obtained.

The main idea of ensemble systems is to combine a set of models/classifiers, in order to obtain a better approximation of the relationship, and hence, a better, and reliable, predictive performance than any single model/classifier [73]. This chapter provides a comprehensive background to ensemble systems from the system's designer point of view.

2.2 Philosophy of Ensemble Systems

Several theoretical and empirical studies have demonstrated that the performance of an ensemble of classifiers/models outperforms the performance of a single classifier/model.

Kuncheva [46] states that the common understanding in the research community is that ensemble systems work for three general reasons: statistical, computational, and representational [21, 65].

2.2.1 Statistical

The goal of a learning algorithm is to search a space $\mathcal{H}$ of hypotheses to find the best hypothesis in the space. If the size of the available training data is too small compared to the size of the hypotheses space, the statistical problem emerges. The learning algorithm, without sufficient training data, may find several hypotheses in the space $\mathcal{H}$, all of which have the same performance on the training data, but may have different generalization performances. If we pick one of these classifiers/models as the solution, we may run into the risk of choosing the wrong classifier/model for the problem. By constructing an ensemble to combine the output of these classifiers/models, this risk is reduced [21, 46, 65].

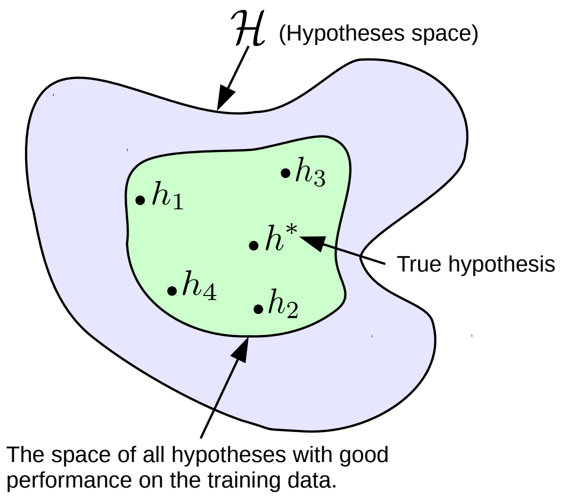

$\mathcal{H}$ (Hypotheses space)
h_3
h_1
h^*
True hypothesis
h_4
h_2
The space of all hypotheses with good performance on the training data.

Figure 2.1: Statistical reason for using ensemble systems (reproduced from [21])

Dietterich [21] graphically illustrates this argument in Figure 2.1. The outer curve denotes the hypotheses space $\mathcal{H}$, while the inner curve denotes the set of hypotheses that all have a good performance on the training data. The true hypothesis is denoted by h^*. By combining all accurate hypotheses, we may find a better approximation to h^* than any single hypothesis [46]. Although it is not guaranteed that the ensemble performance outperforms the best single classifier/model, the risk of making a poor selection is certainly reduced [65].

2.2.2 Computational

Several computational reasons justify the use of ensemble systems, which include imperfect learning algorithms, too much data, small sample size, divide and conquer, and data fusion [21, 46, 65].

Some learning algorithms, such as the back-propagation algorithm for training neural networks, are only guaranteed to converge to a local optima. Even with enough training data, it is computationally difficult for the learning algorithm to find the best hypothesis. Therefore, constructing an ensemble of several classifiers/models with different starting points, as shown in Figure 2.2, may result in a better approximation of the true hypothesis than any individual classifier/model [21, 46].

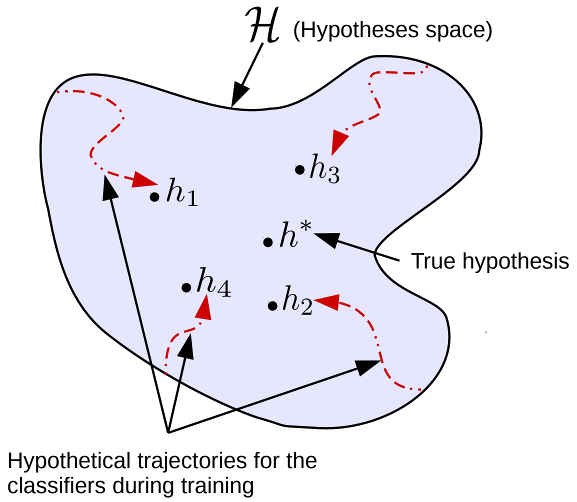

$\mathcal{H}$ (Hypotheses space)
h_3
h_1
h^*
True hypothesis
h_4
h_2
Hypothetical trajectories for the classifiers during training

Figure 2.2: Computational reason for using ensemble systems (reproduced from [21])

In the age of big data, training a single classifier/model using too much data becomes unfeasible. A better choice is to construct an ensemble of many classifiers/models trained on different, manageable, parts of the available data. On the other hand, if the sample size is small, the ensemble base classifiers/models can be trained on different data subsets, generated by using a resampling technique [46, 65].

For some problems, a divide and conquer approach is the most efficient one. The problem is split into smaller, easier to handle, sub-problems. Then, a classifier/model is trained for each sub-problem. The overall solution to the problem is achieved by constructing an ensemble of all classifiers/models, where each sub-problem is handled by the specialized classifier/model [46, 65].

Finally, data fusion is another reason for the use of ensemble systems. In many real-world applications, the data comes from a variety of sources, and with different nature of features (e.g., text, images, video, audio, measurements, etc.). Instead of training a single classifier/model on all types of features, it might be better to construct an ensemble of classifiers/models; each one is built on a particular type of features [46, 65].

2.2.3 Representational

The representational reason arises from the fact that for some applications the true hypothesis h^* cannot be represented by any of the hypotheses in the space $\mathcal{H}$, as shown in Figure 2.3. By constructing an ensemble of good hypotheses drawn from the space $\mathcal{H}$, it may be possible to produce a solution close to the true one. For example, an ensemble of linear classifiers can produce a highly nonlinear classification boundary [21, 46].

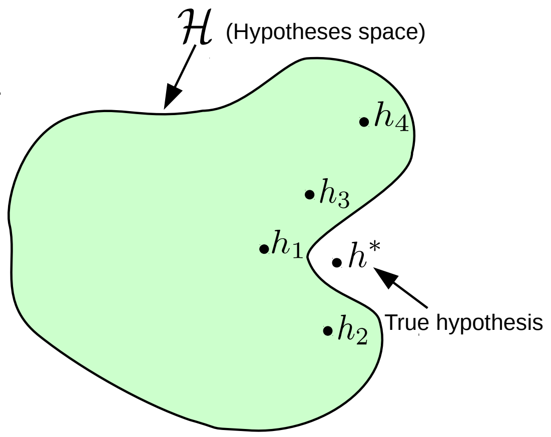


Figure 2.3: Representational reason for using ensemble systems (reproduced from [21])

2.3 Ensemble Methods

Although it is highly probable that the fusion of several models/classifiers would outperform a single model/classifier [78], the fusion of different models might only add to the complexity of the system without any performance improvement. Thus, for an ensemble to be successful, its base models/classifiers should be different from each other [83]. Several methods have been reported in the literature to inject model diversity. In this section, three common methods are reviewed, namely, bagging, boosting, and random subspace. More details on these methods can be found in the provided references.

2.3.1 Bagging

Bagging, which is an acronym for Bootstrap AGGregatING, was introduced by Breiman [11]. The idea of bagging is to inject model diversity at the data level, by training the ensemble classifiers/models on bootstrap replicates (sampling with replacement) of the training data set. Then, the outputs of the classifiers/models are combined using majority vote (in the case of classification), or averaging (in the case of regression). Often, the boosting method is used to build ensembles of decision trees classifiers/models,but it can be applied to ensembles of other learning algorithms. However, bagging is most effective when it is used with unstable non-linear learning algorithms (e.g., decision trees, neural networks, etc.), where small changes in the training data set will lead to significant changes in the classifier/model predictions [30, 46]. The random forests [12], and stochastic gradient boosting machines [29] learning algorithms have built-in bagging algorithms. The two algorithms will be used to build classifiers/models for the regression and classification case studies, Chapter 4 and Chapter 5, respectively. Although random forest and stochastic gradient boosting machines often produce high performance, as the size and dimension of the training data increase, powerful computing resources and long training time are required.

2.3.2 Boosting

The boosting algorithms have been ranked among the top ensemble methods, due to their accuracy, robustness, and broad applicability [46]. Boosting defined by Freund and Schapire [26] as "*general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb*". AdaBoost [27], which is an acronym for ADAptative BOOSTing, is one of the top boosting algorithms. There are two implementations of the AdaBoost algorithm, namely, with resampling and with reweighting. The general idea of AdaBoost algorithm with resampling is to build the ensemble incrementally, adding one classifier/model at a time. The newly added classifier at step k is trained on a dataset selectively sampled from the training dataset Z. The sampling distribution of the training samples starts from uniform and is updated for each new classifier/model, based on the performance of the previous classifier. The likelihood of the misclassified samples at step $k-1$ is increased, so that they have higher probabilities to be selected in the training dataset for the next classifier. In the AdaBoost algorithm with reweighting, no sampling is required. The base classifiers directly use the probabilities on Z as weights [30, 46]. The stochastic gradient boosting machines [29] learning algorithm has, in addition to bagging, a built-in boosting algorithm.

2.3.3 Random Subspace

The random space method [37] injects model diversity at the feature level. The ensemble is constructed from classifiers/models trained on randomly sampled subsets of the features. The final prediction of the ensemble is made by combing the outputs of the classifiers/models using majority vote (for classification tasks), or averaging (for regression tasks) [30, 46]. The random forests [12] learning algorithm employees, in addition to bagging, the random space method.

2.4 Fusion Topology

Several types of topologies have been reported in the literature to combine the output of base models/classifiers. Lu [54] proposes three categories: cascading, parallel, and hierarchical. A more comprehensive categorization is provided by Lam [50], where topologies are classified into conditional, hierarchical (serial), parallel and hybrid. These topologies are reviewed in this section [30, 68].

In a cascading topology, the output of a classifier/model is used as an input to the next classifier/model, as shown in Figure 2.4. The final prediction is obtained by the last classifier/model in the chain. The major drawback of this configuration is the inability of later classifiers/models to correct errors made by the earlier classifiers/models.

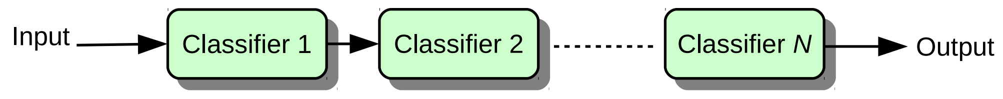


Figure 2.4: Cascading topology (reproduced from [68])

The parallel topology combines the output of all classifiers/models in a single location, as shown in Figure 2.5. The performance of ensembles using this topology largely depends on the selection of proper fusion function. This is particularity important if the base classifiers/models have significantly different performances, where poor classifiers/models can affect the overall performance of the ensemble.

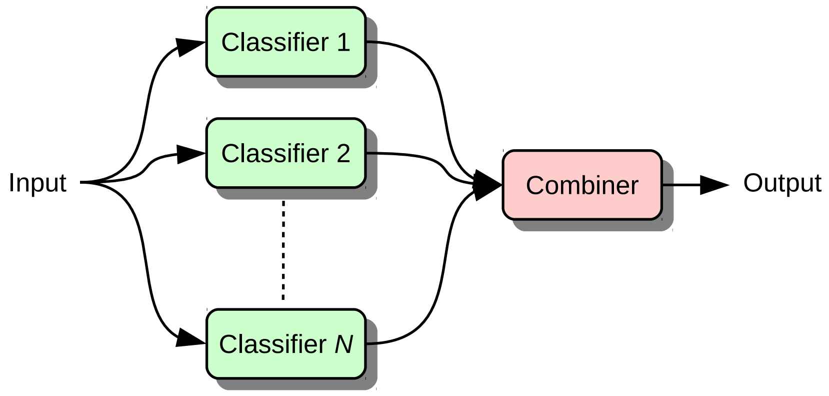


Figure 2.5: Parallel topology (reproduced from [30, 68])

In the hierarchical topology, both cascading and parallel configurations are used to combine the base classifiers/models, as shown in Figure 2.6, to obtain the optimal performance. The use of this topology may compensate the drawbacks of the cascading and parallel topologies.

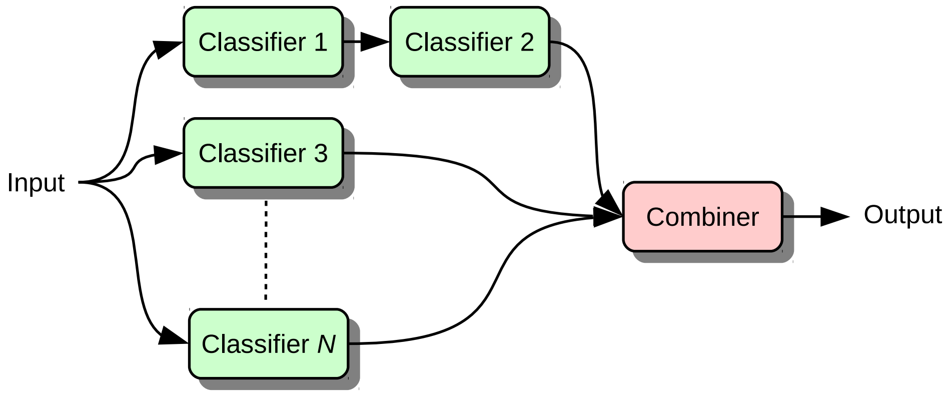


Figure 2.6: Hierarchical topology (reproduced from [68])

Lam [50] defines a slightly different hierarchical (serial) topology, where the base classifiers are applied in succession to reduce the number of possible classes in the input data, as shown in Figure 2.7. As the data travels through the system, the decision of later classifiers becomes more focused.

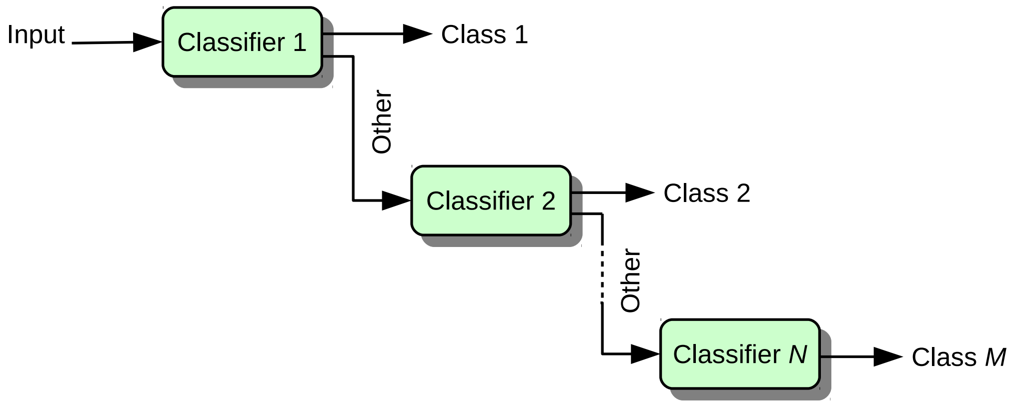


Figure 2.7: Hierarchical (serial) topology (reproduced from [30])

In the conditional topology, a classifier/model is first selected to make a prediction. If it fails to produce an accurate prediction for the presented data, another classifier/model is selected, as shown in Figure 2.8. Often, a primary classifier/model is selected by default as the first one in the classifiers/models queue. The selection of the next classifier/model can be static or dynamic (based on the prediction of the primary classifier/model). The major drawback of this topology is the selection of a criterion by which the failures and success of a classifier/model can be evaluated at prediction time.

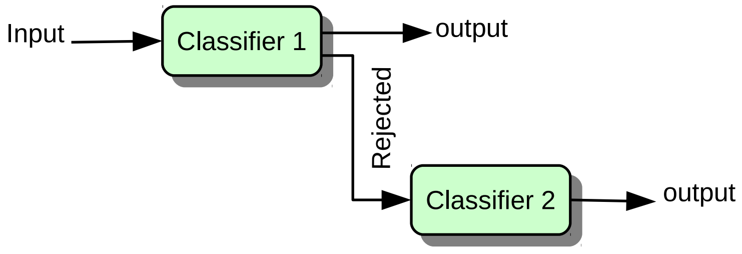


Figure 2.8: Conditional topology (reproduced from [30])

Finally, a hybrid topology, shown in Figure 2.9, employees a selection mechanism to select the best classifier/model for a given input data. This topology could be considered as a trade-off between the parallel and serial topology. Complexity is the main drawback of the hybrid topology.

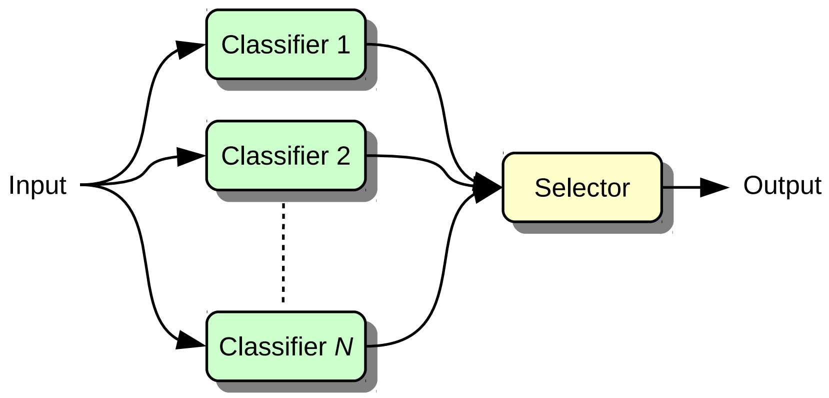


Figure 2.9: Hybrid topology (reproduced from [30])

2.5 Fusion Function

Once a set of classifiers/models have been trained, and a topology is selected, the next step is to combine the output of classifiers/models using an appropriate fusion function.

Generally, the selection of the fusion function depends on the type of the classifiers' output. Kuncheva [46] categorizes classifiers' output into three types: Type 1 - abstract level (the classifier produces a label), Type 2 - rank level (the classifier produces an ordered list of possible classes), and Type 3 - confidence or measurement level (the classifier/model produces numerical values, which represents class probabilities (for a classification task), or target predictions (for a regression task). For example, if the base classifiers' output of Type 1 or Type 2, several fusion functions can be used, such as majority vote, weighted majority vote, and Bayesian combination, while if the base classifiers' output is of Type 3, average rule, max rule, min rule, median rule and the Dempster-Shafer approach can be used among others [30, 68]. Stacked generalization, a more advanced approach, can be applied to both types of output.

Different categorizations of fusion functions have been reported in the literature [68]. Xu et al. [94] categorize fusion functions into two categories: feature-vector-based methods (e.g. neural networks), and syntactic-and-structural methods (e.g., fuzzy rule bases).

Canuto [17] divides fusion functions into four more informative categorizations, namely, linear (e.g., sum, product, etc.), non-linear (e.g., majority vote, weighted majority vote), statistical (e.g., Dempster-Shafer technique, Bayesian combination, etc.), and computationally intelligent (e.g., neural networks, genetic algorithms, etc.). Duin [22] classifies fusion function into fixed, and trainable. The fixed functions, such as majority vote, sum, and average, do not require training. On the other hand, trainable functions, such as stacked generalization approach, require a training phase.

Some of the common fusion functions are briefly discussed in the following sections. More details can be found in the provided references.

2.5.1 Majority Vote and Weighted Majority Vote

The majority vote is one of the oldest strategies for decision making. It can be traced back to the era of ancient Greek city states and Roman Senate. Each classifier in the ensemble classifies a new sample based on its own evaluation; the final class prediction is the class with the greatest number of votes [30, 46]. To avoid a tie, the modeler should choose an odd number of ensemble's classifiers. Majority vote will be used in the classification case study, Chapter 5.

Kuncheva [46] formalizes the majority vote function as follows [30]: let us assume the outputs of the ensemble's classifiers is denoted by a binary vector of size M,

$$[d_{i,1}, ..., d_{i,M}]^T \in \{0,1\}^M, i = \{1, ..., B\}$$

where B is the number of ensemble's classifiers, M is the number of possible classes. If the ith classifier votes the class C_j, for the new sample, $d_{i,j} = 1$, otherwise $d_{i,j} = 0$. The ensemble will predict the class C_k if it has the maximum number of votes:

$$\sum_{i=1}^{B} d_{i,k} = \max_{j=1}^{M} \sum_{i=1}^{B} d_{i,j} \tag{2.1}$$

The weighted majority vote function is a variation of the majority vote, where each classifier is assigned a weight. The classifiers' weights are determined, using some criterion, before the fusion process. Equation 2.1 can be converted to the weighted majority vote, Equation 2.2, by adding the w_i vector, which represents the weights for the ith classifier [30].

$$\sum_{i=1}^{B} w_i d_{i,k} = \max_{j=1}^{M} \sum_{i=1}^{B} w_i d_{i,j} \tag{2.2}$$

2.5.2 Bayesian Combination

The Bayesian approach is based on the posteriori probability, where the conditional terms are represented by the predictions of the ensemble's classifiers. For a given new sample x, the ensemble will assign the class that maximizes such probability. Formerly, for the generic class C_k, the fusion function has to predict the class that maximizes the probability:

$$P\left(C_k|y_1, y_2, ..., y_B\right) \tag{2.3}$$

where y_i represents the class prediction of the ith classifier. However, the conditional probabilities are often unknown. To overcome this problem, the Bayesian rules can be used to approximate Equation 2.3. Common combination rules include product rule, sum rule, max rule, min rule, and median rule. The product rule is explained next [30].

Using the Bayes rule, Equation 2.3 can be rewritten as:

$$P\left(C_k|y_1, y_2, ..., y_B\right) = \frac{P(C_k)P\left(y_1, y_2, ..., y_B|C_k\right)}{P\left(y_1, y_2, ..., y_B\right)} \tag{2.4}$$

The denominator can be rewritten as:

$$P\left(y_1, y_2, ..., y_B\right) = \sum_{l=1}^{M} P\left(y_1, y_2, ..., y_B|C_l\right) P(C_k) \tag{2.5}$$

where M is the number of possible classes. By assuming conditional independence between the predictions of all the classifiers, the conditional probability can be rewritten as:

$$P\left(y_1, y_2, ..., y_B|C_k\right) = \prod_{i=1}^{B} P\left(y_i|C_k\right) \tag{2.6}$$

consequently Equation 2.4 becomes:

$$P\left(C_k|y_1, y_2, ..., y_B\right) = \frac{P(C_k)\prod_{i=1}^{B} P\left(y_i|C_k\right)}{\sum_{l=1}^{M}\prod_{i=1}^{B} P\left(y_i|C_l\right) P(C_k)} \tag{2.7}$$

In order to maximize Equation 2.7, the numerator needs to be maximized with respect to k, that is:

$$\max_k \left\{ P(C_k) \prod_{i=1}^{B} P\left(y_i | C_k\right) \right\} \tag{2.8}$$

Equation 2.8 represents the product rule. One of the major drawbacks of the product rule is that one or more of the ensemble's classifiers may give a probability very close to zero, which leads to an aggregate result very close to zero. In this case, the fusion function could fail [30].

2.5.3 Dempster-Shafer Theory

The Dempster and Shafer (DS) theory [31] has been used to deal with uncertainty management and incomplete reasoning. Unlike the Bayesian approach, DS theory can explicitly model the unknown information. The accumulation of evidence is used to narrow down a set of hypotheses. DS theory allows the representation of the ignorance due to the uncertainty of the evidence. If the value of the ignorance reaches zero, the DS model is reduced to a standard Bayesian model [30].

2.5.4 Stacked Generalization

Wolpert [93] introduced the concept of stacked generalization (or stacking), where the fusion function is another classifier/model, called meta-learner, trained using the outputs of the ensemble's classifiers/models. It is critical that the training of the meta-learner is done on a new training dataset (was not used to train the ensemble's classifiers/models), or on the predictions of the out-of-fold cross-validation [30]. Stacking will be utilized in the classification case study, Chapter 5.

2.6 Related Work

The ensemble methods have received a lot of interest from researchers and practitioners. Specifically, several taxonomies have been reported in the literature, which aim to categorize ensemble systems from the system designer's point of view. This section provides an overview on the most common taxonomies [73].

Ho [38] as well as Valentini and Masulli [90] divide the ensemble methods into two categories:

1. Coverage optimization methods: such as boosting, aim to improve the predictive performance by combining a large set of complementary, generic classifiers/models using a fixed fusion function (e.g., majority vote, average, etc.).

2. Decision optimization methods: such as the mixture of experts, aim to find the optimal combination of a fixed set of highly specialized classifiers/models.

Duin [22] as well as Kamel and Wanas [42] propose two categories of ensemble methods (based on the type of the fusion function):

1. Non-trainable ensembles: a simple fusion function (e.g., majority vote, average, etc.) is used to combine the output of the base classifiers/models, without the need to train the combiner.

2. Trainable ensembles: additional training is necessary to train the combiner to optimally combine the output of the base classifiers/models.

Sharkey [81] developed a taxonomy for neural networks ensembles, with three dimensions:

1. The first dimension indicates if the ensemble's base models/classifiers are competitive or co-operative. In a competitive ensemble, a single base classifier/model is selected to make the final prediction, while in a co-operative ensemble, the final prediction is made by combining all base classifiers/models.

2. The second dimension indicates if the ensembles are created top-down or bottom-up. In top-down ensemble, the combiner does not take into account the output of the base classifiers/models. On the other hand, in bottom-up ensembles, the output of the base classifiers/models is taken into account by the combiner.

3. The third dimension indicates if ensemble has pure, modular, or hybrid components. In pure ensembles, all base classifiers/models are combined to obtain a more accurate and reliable prediction for the same task. On the other hand, modular systems break down a complex problem into several sub-problems, each sub-problem is solved by a specialized classifier/model.

Brown et al. [14] categorizes ensemble methods, based on how model diversity is injected, into two types. Implicit diversity, by using randomization methods, or explicit diversity, via some diversity metric.

Kuncheva [46] groups ensemble methods, based on how model diversity is injected, into four levels, as shown in Figure 2.10:

1. Combination level: different approaches to combine base classifiers/models, are used.
2. Classifier level: various types of learning algorithms are used to construct the ensemble.
3. Feature level: different feature subsets are used to train the base classifiers/models.
4. Data level: different training subsets are used to train the base classifiers/models.

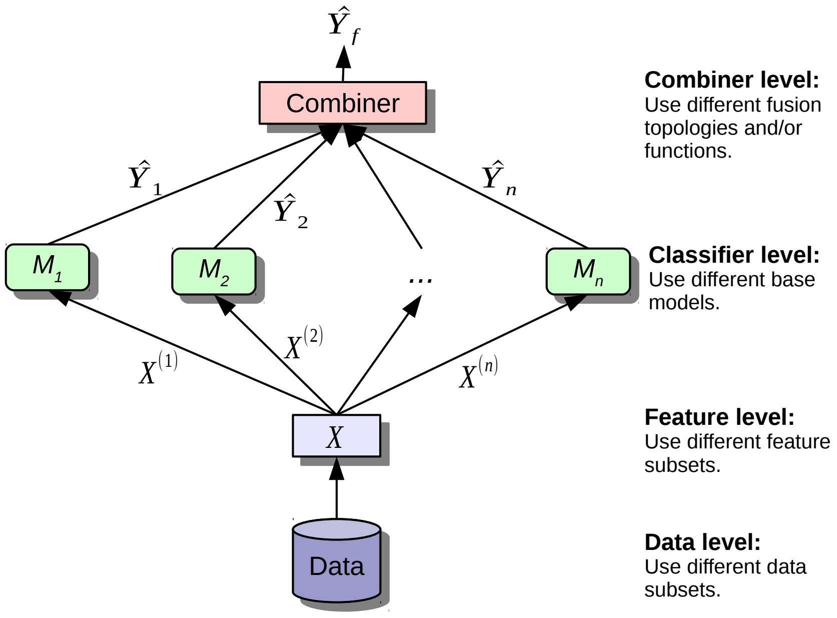


Figure 2.10: An ensemble taxonomy based on four levels (reproduced from [46])

Rokach [73] proposes a taxonomy of five dimensions, as shown in Figure 2.11:

1. Combiner usage: specifies the relationship between the combiner and the ensemble generator.
2. Classifiers dependency: how does each classifier affect the other classifiers during training. Independent classifiers can be trained simultaneously, while dependent classifiers are trained serially.
3. Diversity generator: how model diversity is injected into the ensemble.
4. Ensemble size: how to determine the number of classifiers/models in the ensemble.

5. Cross-inducer (universality): whether the ensemble method can be applied to all common learning algorithms, or specifically designed for a certain algorithm(s).

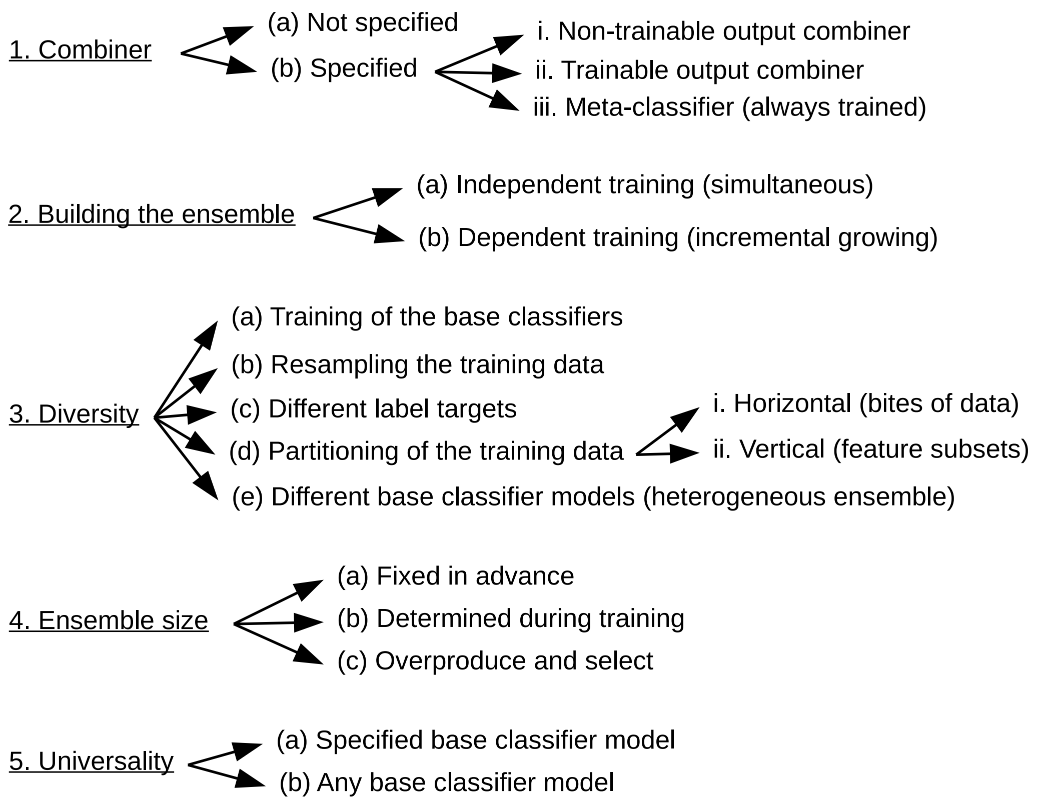

1. Combiner
(a) Not specified
(b) Specified
i. Non-trainable output combiner
ii. Trainable output combiner
iii. Meta-classifier (always trained)
2. Building the ensemble
(a) Independent training (simultaneous)
(b) Dependent training (incremental growing)
3. Diversity
(a) Training of the base classifiers
(b) Resampling the training data
(c) Different label targets
(d) Partitioning of the training data
i. Horizontal (bites of data)
ii. Vertical (feature subsets)
(e) Different base classifier models (heterogeneous ensemble)
4. Ensemble size
(a) Fixed in advance
(b) Determined during training
(c) Overproduce and select
5. Universality
(a) Specified base classifier model
(b) Any base classifier model

Figure 2.11: An ensemble taxonomy based on five dimensions (reproduced from [46])

2.7 Summary

This chapter provided a comprehensive overview of ensemble systems from the system's designer point of view, which included: the philosophy behind the use of ensemble systems, ensemble methods, and types of fusion topology and function.

The chapter ended by presenting the related work to this research. Although several taxonomies have been reported in the literature, there are several research gaps need to be addressed. First, a comprehensive framework for developing ensemble systems is not yet available [77]. Moreover, some of the proposed design methods, such as “overproduce and choose” (also called “test and select”) [84], become unfeasible for large and/or high-dimensional data sets. Second, several strategies have been proposed to inject model diversity in the ensemble; however, there is a shortage of empirical studies that compare the effectiveness of these strategies. Third, most of the ensemble systems research has concentrated on simple problems and relatively small/low-dimensional data sets. Further experimental research is required to investigate the application of ensemble systems to large and/or high-dimensional data sets, with a variety of data types, such as time-series, text, multimedia, etc [73].

In the next chapter, a framework for ensemble predictive modeling is proposed, to address the first research gap. The remaining research gaps will be dealt with in the consecutive chapters.

Chapter 3

Proposed Framework for Ensemble Predictive Modeling

3.1 Introduction

Predictive modeling is typically an iterative process involving problem formulation, exploratory data analysis, data pre-processing, feature selection/engineering, model selection, model validation and interpretation of the results. Kuhn and Johnson [45] state some common reasons why predictive models fail. These may include inadequate data pre-processing, inadequate model validation, unjustified extrapolation (e.g., applying the model to a new data space which was not used during model training), or, over-fitting the model to the training data set. In the context of ensemble systems, the proposed framework incorporates the principles and best practices of predictive modeling outlined by [45] to build optimal subsystem components (base models/classifiers). The thesis coins the term "ensemble predictive modeling" to refer to the process of developing ensemble systems. In addition to subsystem (individual models) predictive modeling, this process includes the systematic introduction of model diversity, the selection of a representative fusion topology, the selection of a suitable fusion function, and adequate fusion evaluation.

The proposed framework for ensemble predictive modeling, shown in Figure 3.1, contains four main phases, namely, phase 0: objective, phase 1: data preparation, phase 2: model building, and phase 3: model fusion. These phases are discussed in the following sections.

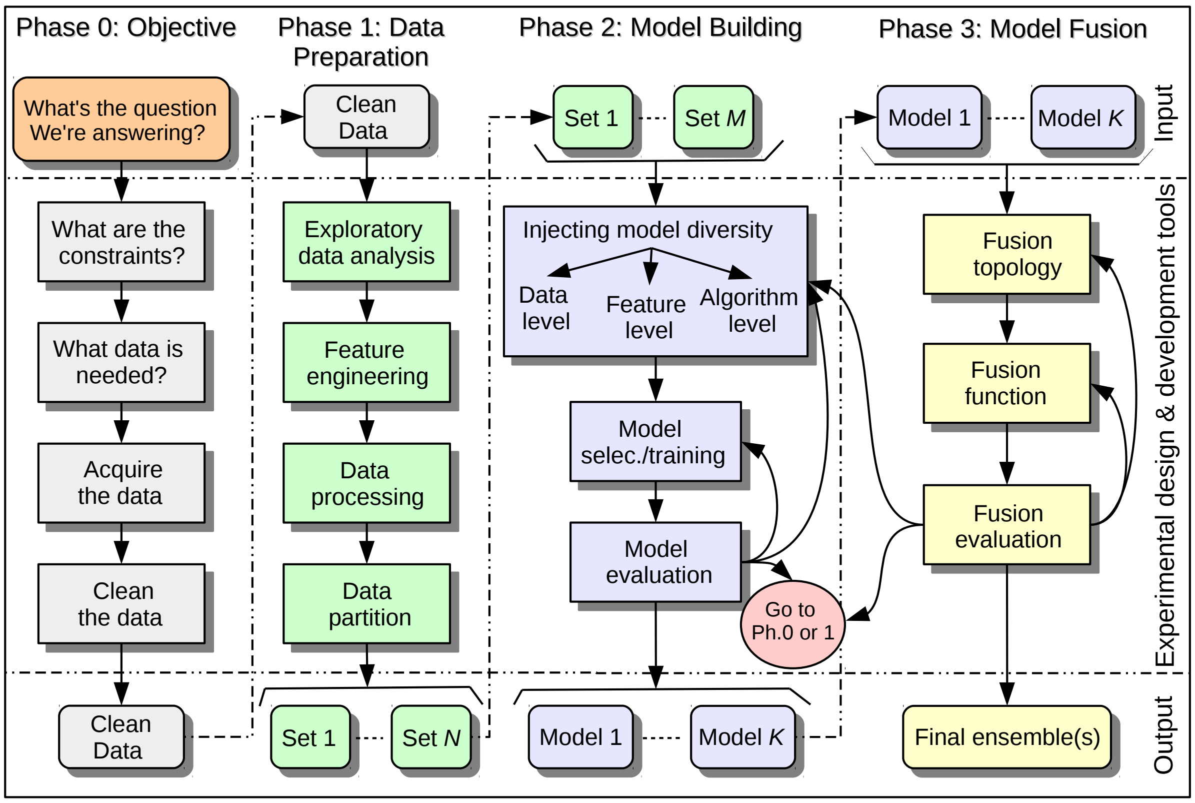


Figure 3.1: Proposed framework for ensemble predictive modeling

3.2 Phase 0: Objective

Kuhn and Johnson [45] argue that building a machine learning model appears to be straightforward: choose a learning algorithm, feed it data, and produce a predictive model. However, the generated model will most likely not be reliable and trustworthy for predicting new data. They state that the first step in any predictive modeling task is to understand the objective and data of the modeling.

Therefore, the problem/question addressed by the ensemble predictive modeling project must be clearly defined, along with the desired outcomes. The modeler (commonly referred to as data scientist in the industry) must take into account any constraints, and ensure that the final results are practical and actionable. Then, the data/tools needed to answer the question should be identified. This process involves the assessment/cleaning of the currently available data, data to be collected, volume/features of the data to develop the models/ensembles, and the appropriate tools/resources.

Although phase 0 has a significant impact on the quality of the models/ensembles (following the famous rule “ garbage in, garbage out”), often in academic research the data collection step is not done by the modeler/data scientist. Thus, the available data to answer the question at hand may not be enough. On the other hand, data scientists in industry settings have to design experiments/procedures to collect, clean, and integrate required data from different sources. The primary outcome of phase 0 is a clean data set, which serves as the input to the next phase (data preparation).

3.3 Phase 1: Data Preparation

Data preparation has a significant effect on the predictive ability of a model, especially for regression applications [45]. This section presents the typical steps performed to pre-process the clean data set, obtained from phase 0 before it is used in phase 2 (model building). In the first case study, Chapter 4, these steps are explained in details using a real data set.

3.3.1 Exploratory Data Analysis

Exploratory data analysis is typically applied before any formal modeling commences. Several statistical summaries and visualizations can be used to understand the data at hand, and eliminate or sharpen potential hypotheses about the world that can be addressed by the available data. Some of the common statistical summaries and plots include histograms and predictor’s statistics. Histograms can be used for class distribution, predictor’s skewness, etc., while predictor’s statistics can be used for missing values, minimum, first quartile, median, mean, third quartile, maximum, and standard deviation. The box-and-whisker plot is typically used to represent most predictor’s statistics graphically. The exploratory data analysis step also identifies what data preparation steps are needed (e.g., data processing to resolve the skewness of predictors).

3.3.2 Feature Engineering

Feature engineering, how the predictors are encoded, is an important step in the data preparation phase, which often has a significant impact on the performance of models.

Some of the feature engineering tasks, such as adding/removing predictors, are informed by the modeler's knowledge of the problem domain at hand. Others, such as binning predictors and converting from categorical to dummy predictors, are standard procedures.

3.3.3 Data Processing

In many of real-world data sets, some of the predictors have no values for a given sample. It is vital to understand why these values are missing and identify if the pattern of missing data is correlated with the outcome, before any strategy to handle the missing values is applied. Such a strategy could be as simple as using the predictor's mean to fill the missing values, or as complex as building models to predict the missing values.

Many of the learning algorithms, such as artificial neural networks and support vector machines, require the predictors to have a common scale. The common scale is achieved by subtracting their average value from all the values. The centered predictors have a zero mean. Then, the predictors are scaled by dividing each value by the predictor's standard deviation. The scaled predictors have a common standard deviation of one. Performing both operations is commonly referred to as normalization. It should be noted that if data transformation is required to resolve skewness (e.g. Box-Cox transformation [10]), it should be performed first, and then, followed by centering and scaling [45].

Resolving predictors' skewness is another issue that should be handled. It should be noted that data transformations should be applied to the training data at hand, and the same values used to transform the training data, are applied to the validation/testing data. For example, during model selection using 10-fold cross-validation, the data transformations are applied on-the-fly to the first nine folds, used as training data, and the same transformation values are used to transform the 10th fold, used as a validation data. The process is repeated for other iterations.

3.3.4 Data Partition

The proper allocation of data to different tasks (e.g., model/feature selection, model training, model fusion, performance evaluation) is one of the important aspects of ensemble predictive modeling. A crucial element of data partition is maintaining a similar population across all samples. For example, in a classification problem, a similar class distribution must be preserved across all data sets (e.g., model selection (including cross-validation folds), training, fusion, and testing data sets) by using stratified sampling.

The number of observations/samples allocated for each modeling task depends on the size of the available data. If the data is small, data partition can have a critical impact on the quality of the final models. For example, if the training data set is small, the learning algorithms, used to build models, may not be able to capture the underlying patterns. On the other hand, a small testing data set would have a limited capacity to evaluate the performance of models/ensembles. In this case, a resampling technique, such as n-fold cross-validation, combined with only a training data set might be a better choice for performance evaluation [45]. The processed data partitions (e.g., model/feature selection, training, fusion, and testing data sets) constitute the input to phase 2 (model building).

3.4 Phase 2: Model Building

The model building phase is the most significant phase in the proposed framework for ensemble predictive modeling. It is highly probable that the fusion of several models/classifiers would outperform a single model/classifier [78]. However, the fusion of different models might only add to the complexity of the system without any performance improvement. Thus, in order for an ensemble to be successful, the errors made by its base models/classifiers should not be highly correlated, and each base classifier/model has an acceptable performance [83]. Accordingly, model diversity is considered one of the key design features within a successful ensemble [30, 46, 68, 73, 77]. Another significant key design feature is the selection of the optimal hyper-parameters (model selection) for the learning algorithms using a quantitative approach, which leads to the optimal performance of the individual models/classifiers. The use of a quantitative approach for model selection is a significant difference between the proposed framework and the common practice reported in the literature [77], where random values are selected for the hyper-parameters by the modeler based on his intuition, to inject model diversity. Proper model evaluation is a vital third step in the model building phase, where the performance and diversity of the base models/classifiers are evaluated before the model fusion phase. If the performance and diversity are not acceptable, some/all of the three steps of the model building phase are repeated. If no improvements are achieved, the modeler needs to go back and start at phase 0 (to collect relevant data), or phase 1 (to investigate engineering new features or/and different data processing approaches). The primary output of the model building phase is a collection of diverse models/classifiers. This section gives an overview on injecting model diversity, model selection/training, and model evaluation. Practical details, using real-world data sets, will be provided in the case studies, Chapter 4 and Chapter 5.

3.4.1 Injecting Model Diversity

At the model building phase, the proposed framework for ensemble predictive modeling incorporates three levels of model diversity, namely, model diversity at the data, feature, and learning algorithm levels [46]. We hypothesize that the optimal performance of an ensemble can be achieved by systematically injecting model diversity at all levels. Conventional methods to incorporate model diversity at these levels are discussed in the following subsections.

Diversity at the Data Level

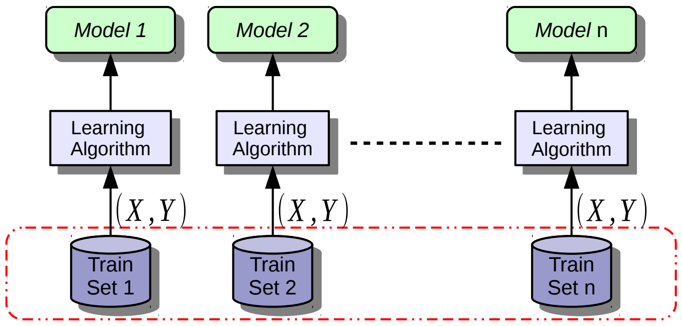


Figure 3.2: Injecting diversity at the data level

Injecting model diversity at the data level is the most used approach to incorporate model diversity within an ensemble. Diversity is injected by training models, using the same learning algorithm, on different subsets of the training data set, as shown in Figure 3.2. The most popular method for the selection of the subsets is the bagging technique, discussed in Section 2.3.1. On each iteration of the learning algorithm, a predefined percentage of the training data set is drawn randomly, with replacement, to train the current model. Boosting, discussed in Section 2.3.2, is another traditional subsampling method, where the error rate of the previous iteration is used to adjust the probability distribution of the training samples. The new probability distribution is then used to randomly draw, with replacement, a subset of the training samples. Stochastic Gradient Boosting Machines (GBM) has a built-in bagging and boosting algorithms to construct diverse base models/classifiers, while the Random Forest learning algorithm has a built-in bagging algorithm. Another simple subsampling method is to manually construct disjoint subsets of the training data set, and train a separate model for each subset. Finally, diversity at the data level can be achieved by training models on copies of the same training data set, but pre-processed using different techniques [30, 46, 68, 73, 77, 82].

In the regression case study, Chapter 4, model diversity at data level is achieved by three methods. First, manually splitting the training data set into two disjoint subsets, and training models on each subset. Second, the built-in bagging and boosting within the GBM learning algorithm add an extra layer of diversity to the GBM models. Third, two different pre-processing techniques are applied to the training subsets. Due to the relatively small data set, model diversity at the data level in the classification case study, Chapter 5, is achieved by the built-in bagging and boosting within the GBM learning algorithm, and bagging within the Random Forest learning algorithm.

Diversity at the Feature Level

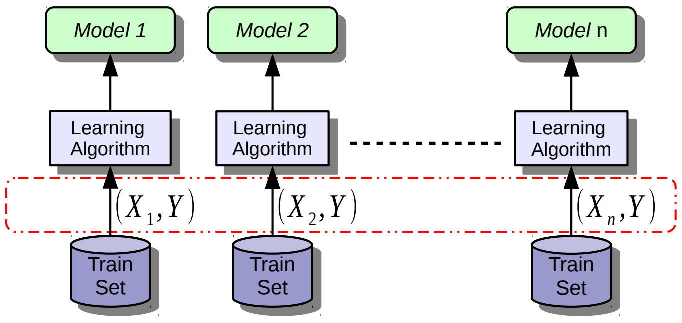


Figure 3.3: Injecting diversity at the feature level

Model diversity at the feature level is achieved by training models, using the same learning algorithm, on copies of the same training dataset, but with different features [46, 68, 73, 82] as shown in Figure 3.3. The features for each training subset can be selected manually or by using different feature selection algorithms. The selected features for the training subsets may overlap or be completely disjoint. Some learning algorithms, such as Random Forest algorithm, have a built-in algorithm to inject model diversity at the feature level.

For the regressing case study, model diversity at feature level is achieved by training models on manually selected subsets of features, while for the classification case study, models are trained on subsets of features selected by five feature selection algorithms. In addition, an extra layer of model diversity at feature level is added to the Random Forest models by its built-in algorithm.

Diversity at the Learning Algorithm Level

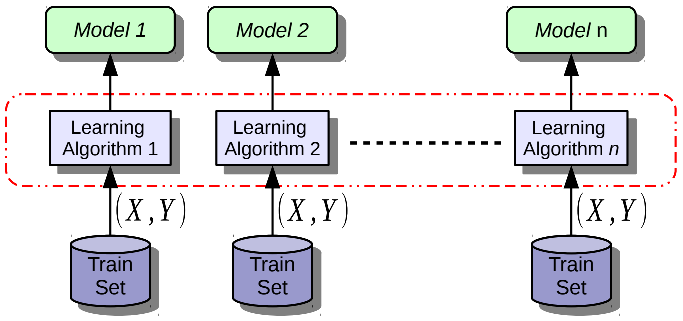


Figure 3.4: Injecting diversity at the learning algorithm level

The main approach to injecting model diversity at the learning algorithm level is to apply different learning algorithms to train models on the same training data set and features, as shown in Figure 3.4. Other methods involve the injection of randomness into the same learning algorithm, such as using different initial weights for several neural networks trained on the same data set [67], or training several models, using the same learning algorithm and data set, with different hyper-parameters values [46, 68, 77, 82].

Model diversity at the learning algorithm level is mainly achieved in the regression and classification case studies by using several learning algorithms. A second layer of diversity is added in the classification case study, where different optimal hyper-parameters values are selected for the same learning algorithm (due to the use of data sets with different features).

3.4.2 Model Selection and Training

In the context of ensemble predictive modeling, model selection may be classified into three types. First, the selection of the ensemble size, how many base models/classifiers should be trained. Second, the selection of the optimal hyper-parameters for the learning algorithms over a set of candidate values. Third, the selection of the learning algorithms to use over others (e.g. based on their performance).

The Selection of Ensemble Size

The ensemble size is determined by several factors, such as the desired accuracy, the nature of the regression/classification problem, computational cost, and the available computing resources [73]. For ensemble learning algorithms, such as the Stochastic Gradient Boosting Machines (GBM), the ensemble size can be treated as a hyper-parameter and tuned using a quantitative approach.

Kuncheva [46] argues that building a large ensemble of strong classifiers is a waste of resources since there is no scope for diversity (e.g., many base classifiers will produce identical predictions, regardless of how different the classifiers may be). He hypothesizes that an ensemble with a small number of strong classifiers may improve on the individual accuracy if their errors are not highly correlated. On the other hand, if the classifiers are weak, a large number of diverse classifiers is required. As discussed before, bagging, boosting, and random forest techniques incorporate this approach.

In the context of big/high-dimensional data, building large ensembles requires powerful computing resources and scalable learning algorithms. Therefore, the proposed framework for predictive modeling suggests building small to medium (e.g., from 3 to 50 classifiers) heterogeneous ensembles of strong base classifiers/models. To maximize model diversity, diversity should be injected at the data, feature, and learning algorithm levels.

The Selection of Optimal Hyper-parameters

Most learning algorithms have one or more hyper-parameter(s). These parameters can not be directly estimated from the data using analytical formulas. The complexity of the model/classifiers is controlled by its hyper-parameters, and hence, poor choices for their values can lead to under- or over-fitting [45]. The proposed framework stress the importance of selecting the optimal hyper-parameters based on a quantitative approach. The use of a quantitative approach for model selection is a significant difference between the proposed framework and the common practice reported in the literature [77], where random values are selected for the hyper-parameters by the modeler based on his intuition, to inject model diversity. As will be shown in the experimental results using real-world datasets, Chapter 4 and Chapter 5, a small change in the value of the hyper-parameter can have a significant impact on the model's performance.

A general approach to search for the optimal hyper-parameters is to define a grid of candidate values, use a proper sampling technique, such as 10-fold cross-validation, to generate reliable estimates of performance for each hyper-parameters combination, as shown in Figure 5.8. The best performing values across all folds are then selected. However, to have a fair comparison, care must be taken regarding three important aspects. First, the similar class distribution must be maintained across all folds, by using stratified sampling. Second, the folds should be presented in the same order, with exactly the same samples, to the learning algorithms (this is particularly important for the selection of learning algorithms). Third, if data pre-processing is required, it should be done on the fly, as shown in Figure 3.5.

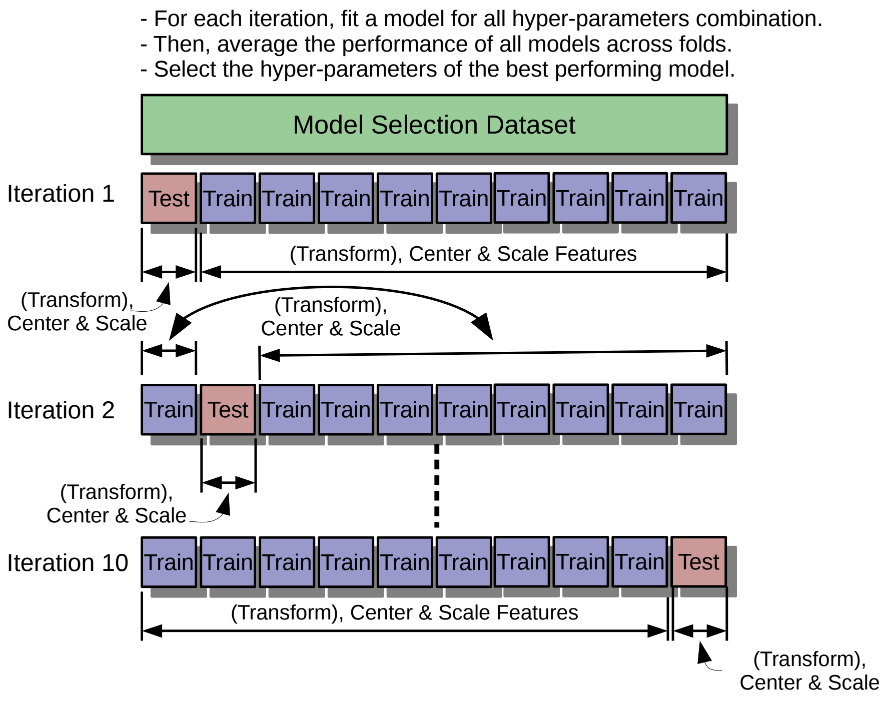
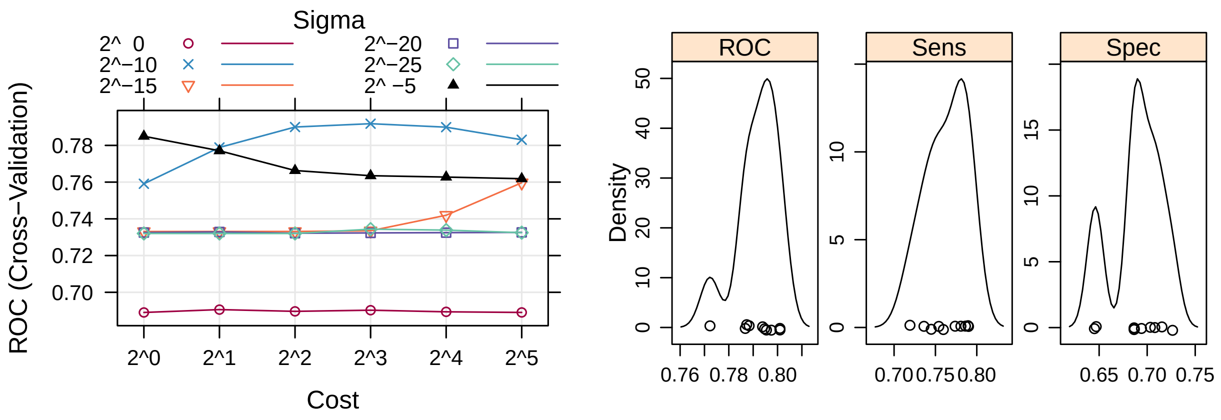

- For each iteration, fit a model for all hyper-parameters combination.
- Then, average the performance of all models across folds.
- Select the hyper-parameters of the best performing model.
Model Selection Dataset
Iteration 1
Iteration 2
Iteration 10
Test
Train
(Transform), Center & Scale Features
(Transform), Center & Scale

Figure 3.5: Model selection using grid-search and 10-fold cross-validation



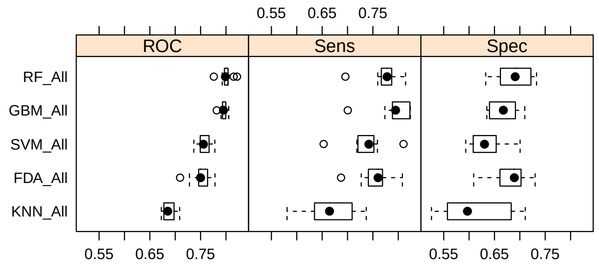
(a) Hyper-parameters grid-search (b) Optimal model resampling accuracy

Figure 3.6: An example of hyper-parameters selection results (SVM classifier)

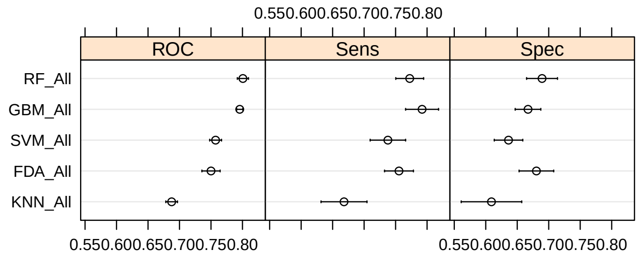
The proposed framework highlights the importance of two aspects during the selection of the optimal hyper-parameters. First, the modeler should understand the effect of different combinations of hyper-parameters values on the classifier/model performance by visualizing the performance of the hyper-parameters grid-search. For example, Figure 3.6a shows the results 10-fold CV and grid-search for SVM classifier Radial Basis Function (RBF) kernel, which has two hyper-parameters (cost and sigma). Based on such a quantitative approach, the modeler may need to go back and refine the hyper-parameters grid, and repeat the process. Second, the modeler should assess the uncertainty of the selected optimal hyper-parameters by visualizing the resampling accuracy's distribution of the optimal model/classifier. Figure 3.6b shows the resampling accuracy (area under ROC, sensitivity, and specificity) distributions of the selected optimal SVM classifier.

The Selection of Learning Algorithms

The selection of the learning algorithms, to build the ensemble models/classifiers, generally depends on the nature of the problem, the modeler's experience and knowledge of algorithms, scalability, computational cost, and performance of algorithms. The proposed framework stresses the importance of selecting the learning algorithms based on a fair quantitative approach. A useful approach is to visualize the results of the selection of the optimal hyper-parameters for each learning algorithm, using the same 10-fold cross-validation data, to assess the relative performance and diversity of the algorithms. Two useful plots are the performance distributions using the box-and-whisker plot, shown in Figure 3.7a, and the performance estimates with 95% Confidence Level (CL), shown in Figure 3.7b.



(a) Performance distributions



(b) Estimates with 95% CL

Figure 3.7: An example of comparing several learning algorithms using 10-fold CV

The selected optimal hyper-parameters are then used to train models/classifiers using the whole training data set(s).

Several diversity measures for classification tasks have been reported in the literature [46, 68]. However, experimental research [8, 33, 47, 49, 85, 92] suggest a weak, and sometimes contradictory, relationship between the diversity measures and ensembles accuracy. This research also experimentally confirms these findings in Section 5.8.1. Therefore, the proposed framework does not use these diversity measures as criteria for selecting the ensemble classifiers. Instead, the framework systematically injects model diversity at all levels (data, feature, and learning algorithm levels), and use the n-fold cross-validation results to assess the relative performance and diversity of the classifiers. As will be validated by experimental results in the classification case study, Chapter 5, this approach leads to the optimal ensemble accuracy without the need for a diversity measure.

3.4.3 Model Evaluation

Model evaluation is of paramount importance in any predictive modeling task. It becomes even more important in ensemble predictive modeling, where the relative performance and diversity of models/classifiers must be thoroughly evaluated. Typically, some measure of accuracy is used to assess the effectiveness of a model/classifier. However, accuracy can be measured using different ways, each with its subtle difference. Relying solely on a single metric to evaluate models/classifiers, and understand their strengths and weaknesses, is problematic [45]. Therefore, the proposed framework for ensemble predictive modeling calls for utilizing several quantitative metrics and visualizations to better assess the performance of the models/classifiers.

A second important aspect of model evaluation is the size of the test data set. If the test dataset is small, it may not be sufficient to make reasonable judgments [45]. Moreover, using a single test dataset can be a poor choice [36, 57, 60]. Therefore, it advisable to use, in addition to a separate test data set, the n-fold cross-validation resampling, used for model selection, as a first step to assess the relative performance of the selected models/classifiers.

Finally, in large-scale/high-dimensional data analysis and modeling, the computational cost of developing several models/classifiers must be considered. The model selection, training, and prediction time for each model should be measured. Prediction time is particularly important for mission-critical applications, such as algorithmic trading systems, where near real-time prediction is essential.

If the performance and/or diversity of models/classifiers are not acceptable, some/all of the three steps of the model building phase are repeated. If no improvements are achieved, the modeler needs to go back and start at phase 0 (to collect relevant data), or phase 1 (to investigate engineering new features or/and different data processing approaches).

More details on using several quantitative metrics and visualizations, to evaluate the performance of regression and classification models, will be given in the regression and classification case studies, Chapter 4 and Chapter 5, respectively.

3.5 Phase 3: Model Fusion

Once a set of diverse and strong models/classifiers are produced by the model building phase, an appropriate fusion strategy should be selected to obtain the optimal ensemble performance. The model fusion phase is an iterative process, where three major tasks are performed sequentially. These tasks are the selection of the fusion topology, the selection of fusion function, and the fusion evaluation.

3.5.1 Fusion Topology

Generally, the selection of a fusion topology depends on the type of the problem at hand, which can be solved by combining ensemble, modular, or hybrid components. Sharkey [80] and Lam [50] distinguish between pure ensemble systems and modular systems. The goal of a pure ensemble system is to combine several models/classifiers, each of which solves the same task (parallel topology), to obtain more accurate and reliable predictions. On the other hand, a modular system breaks down the problem into several sub-tasks, each of which is solved by a specialized model/classifier [68, 73]. A hybrid system combines both approaches.

Another aspect, the system's modeler should investigate, is whether to combine all models/classifiers, produced by the model building phase or combine only a selected subset of them, using a particular criterion [46].

In the regression case study, Chapter 4, hybrid topologies, without selection, are used to combine models, while in the classification case study, Chapter 5, several parallel topologies, with and without selection, are investigated.

3.5.2 Fusion Function

As discussed in Section 2.5, the choice of the fusion function often depends on the type of base models/classifiers' output. For example, if the output is a confidence level (e.g., class probability) or measurement (e.g., regression problem), several simple fusion functions can be used, such as average, weighted average, sum, max, min, etc. If the output is an abstract level (e.g., class label), functions such as majority vote, weighted majority vote, etc., can be used. In both cases, a trained model/classifier, using stacked generalization, can be used to combine the output of the base models/classifier [30, 46, 68]. It should be noted that to avoid over-fitting, the data used to train the base models/classifiers must never be used to train the fusion model/classifier. Instead, a new dataset or out-of-fold cross-validation's predictions should be utilized.

The ensemble size is another important factor in choosing the fusion function. If the ensemble contains a small number of strong models/classifiers, a trained model/classifier might be needed to outperform the best individual model/classifier. On the other hand, high performance can be achieved for large ensembles with a variety of simple fusion functions [46].

In the regression case study, Chapter 4, a simple fusion function, average, is chosen as the fusion function, while in the classification case study, Chapter 5, majority vote, and several stacked generalization classifiers, are investigated.

3.5.3 Fusion Evaluation

Similar to the model evaluation, model fusion should be thoroughly evaluated. In addition to the ensemble performance, several aspects should be assessed. Such aspects include required computing resources, model selection and training time (needed to build all base models/classifiers and the fusion model/classifier), and prediction time (required to produce a final prediction by the ensemble for a new sample) [73]. These aspects are considered in the regression and classification case studies, Chapter 4 and Chapter 5, respectively.

The model fusion phase steps should be repeated until the final ensemble(s) produces an acceptable performance. If no improvement is achieved, the modeler should go back and start at phase 2 (to investigate different model diversity approaches or/and learning algorithms), phase 0 (to collect relevant data), or phase 1 (to examine engineering new features or/and different data processing approaches).

3.6 Experimental Design and Development Tools

During the model selection step of phase 2 (model building), several models/classifiers are experimentally compared, based on their performance, using the same data set. To have an accurate comparison, it is imperative to account for sources of variation. Such sources may include the choice of the testing/training datasets, the internal randomness of the training algorithm, and the random classification error [20, 46]. In addition to these sources of variation, it is also important to account for the randomness in parallel computing, if the model selection is performed on a cluster of computers/cores.

An essential ingredient, for successful ensemble predictive modeling, is a versatile development toolbox and powerful computing resources. The toolbox should offer techniques for data pre-processing, visualization, and a large collection of machine/statistical learning algorithms. Due to the intensive computations, especially during model selection using grid-search and n-fold cross-validation, the need for powerful computing resources becomes a requirement as the size and/or the dimension of the data increase.

The most common used programming languages in the industry/academia, for data analysis and machine learning modeling, are the open source *R* statistical computing language [66] and Python scikit-learn package [62]. As computing resources, computer clusters/cloud computing have been used. Apache Spark is the current industry's state-of-the-art computing resource for large-scale machine learning. Spark is an open source cluster computing framework, initially developed at the University of California, Berkeley [95].

This research uses the *R* statistical computing language and several machine/statistical learning *R* packages as the development tools. An instance of Amazon Elastic Compute Cloud (EC2), with 32 cores and 65 GB RAM, is used as the computing resource.

3.7 Summary

This chapter introduced the proposed framework for ensemble predictive modeling. The proposed framework contains four major phases, namely, phase 0: objective, phase 1: data preparation, phase 2: model building, and phase 3: model fusion. In the next chapter, the proposed framework will be validated using a complex regression case study.

Chapter 4

Regression Case Study: Predicting the Stock Market's Short-term Behavior Following Liquidity Shocks

4.1 Introduction

The prices in order driven markets, such as Nasdaq, London Stock Exchange, and Tokyo Stock Exchange, are determined by the publication of orders to buy or sell shares. Two types of orders, limit orders or market orders, may be submitted by the participants of such markets. In limit orders, the trader specifies the price at which he is willing to transact. The limit order book stores limit orders that do not execute immediately for later execution. Market orders are immediately executed against orders stored in the limit order book. The limit order book acts as a pool of trading interest over a range of prices. Standing buy (sell) orders with the highest bid (lowest ask) price have the highest probability of execution. The bid-ask spread is the difference between the two best prices, which is made small by the competition between market participants. Traders, exchanges, and regulators are interested in understanding the dynamic interplay between market and limit orders, and the state of the limit order book [40].

The two types of changes to the state of the limit order book are trade or quote events. A trade event takes place when shares are sold or bought while a quote event happens whenever the best ask or bid price is updated. Market liquidity is defined as the ability of market participants to trade large amounts of shares at low cost and quickly. Market resiliency (also known as liquidity replenishment) refers to how a market recovers after liquidity has been consumed. Market resiliency is of vital importance to traders willing to reduce their trades impact costs by splitting large orders across time. A liquidity shock occurs when any trade changes the best bid or ask price. Shocks often occur when all available volume at the best price is consumed by a large volume trade (or series of small trades). Following a liquidity shock, the bid-ask spread might be temporarily widened, and/or result in permanent price shifts, as shown in Figure 4.1 [40], and the order book is updated by new orders to buy and sell. Market resiliency is expressed by the ability of bid-ask spread to partially or completely revert to former levels following a liquidity shock. The aim of a liquidity replenishment model/system is to determine the relationship between recent past order book events and future stock price mean reversion following liquidity shocks. Some of the important factors in the development of such a model/system include order arrival rate, bid-ask spread, transaction size, timing, and trading volume [40].

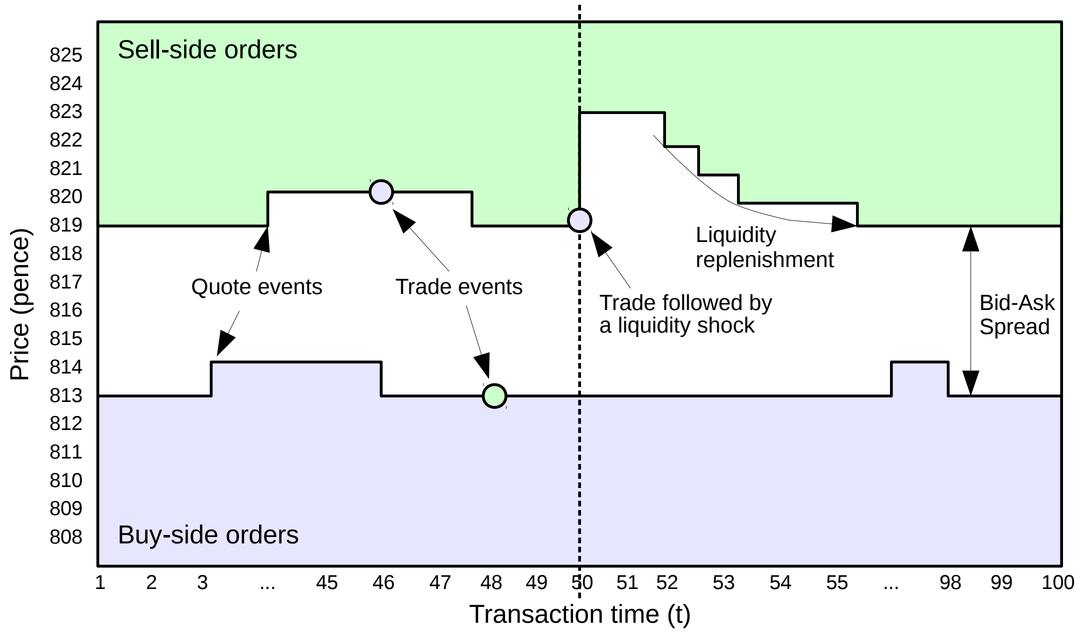


Figure 4.1: Stock market's dynamics

The data for this case study is obtained from the kaggle.com's competition "Algorithmic Trading Challenge" [40] held between Nov. 11th, 2011 and Jan. 8th, 2012. The provided dataset contains recent intraday trade and quote data from the London Stock Exchange

(LSE) before and after a liquidity shock. The raw dataset has 754,018 observations, 207 predictors, and 100 outcomes (50 bid and 50 ask prices to be predicted). Table 4.1 lists the description of the raw data fields.

Table 4.1: Raw Data fields

Predictor name	Description	Type
row_id	Unique row identifier	Integer
security_id	Unique stock identifier	Integer
p_tcount	Count of previous day's on-market trades in current security	Integer
p_value	Sum of previous day's on-market values in current security (£)	Integer
trade_vwap	Volume-Weighted average price of the trade causing the liquidity shock (pence)	Double
trade_volume	Size of the trade causing the liquidity shock (number of shares)	Integer
initiator	Whether the trade is buyer or seller initiated (B = Buyer, S =Seller)	String
transtype$\langle t \rangle$	Whether the time-series event is a trade or quote event (T=Trade, Q=Quote) at event time t	String
time$\langle t \rangle$	Event time (HH:MM:SS.mmm) at event time t	String
bid$\langle t \rangle$	Best buy price (pence) at event time t. Bid1 to Bid50 are predictors, while Bid51 to Bid100 are outcomes to be predicted	Double
ask$\langle t \rangle$	Best sell price (pence) at event time t. Ask1 to Ask50 are predictors, while Ask51 to Ask100 are outcomes to be predicted	Double

4.2 Objective

Most backtesting simulations of trading strategies assume zero market resiliency. Modeling market resiliency increases the realism of these simulations, and hence, improves the trading strategies evaluation methods [40].

The primary objective is to apply the proposed framework for ensemble predictive modeling to develop a system that predicts the stock market's short-term response following large trades. The system is required to predict the behavior of 50 bid and 50 ask consecutive prices following such liquidity shocks. As secondary objectives, the case study aims to compare empirically different modeling strategies and regression learning algorithms (NNET, MARS, GBM, SVM, and RF) based on their performance and computation costs (model selection time, training time, and prediction time). The work in this case study was inspired by the author's previous work on algorithmic trading systems [1, 2, 4].

4.3 Data Preparation

Data preparation has a significant effect on the predictive ability of a model, especially for regression applications [45]. This section presents the steps involved in data preparation phase, namely, exploratory data analysis, feature engineering, data pre-processing, and data partition.

4.3.1 Exploratory Data Analysis

As discussed in Section 3.3.1, the first step in predictive modeling for any application is to understand the data at hand using several statistical summaries and visualizations. This process helps in the development of appropriate predictive models by eliminating or sharpening potential hypotheses about the world that can be addressed by the data. In this section, some of the important aspects of the provided raw data are highlighted.

Figure 4.2 shows the distribution of the 102 securities/stocks in the dataset. The histogram clearly shows unbalanced distribution between securities (some securities have much larger number of observations than others). Since we assume all securities have the same weight, it is critical to use stratified sampling to maintain the same securities' distribution in all stages of predictive modeling (feature selection, model selection, model training, and model evaluation).

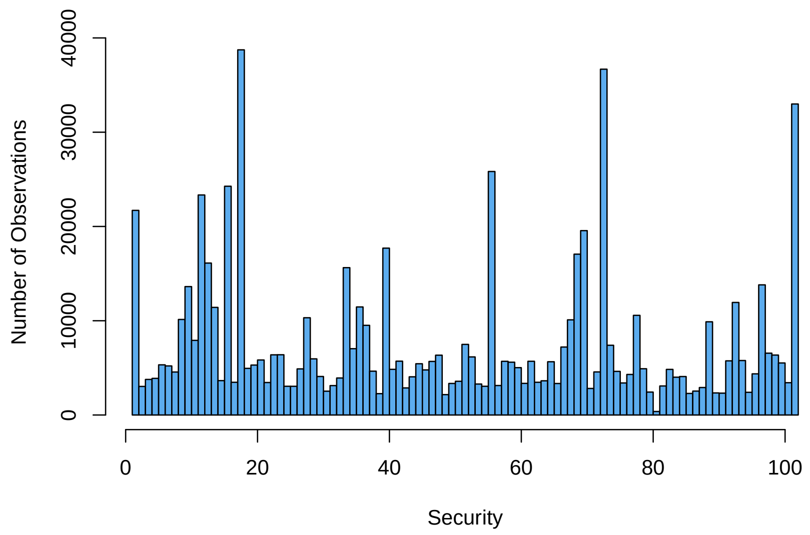


Figure 4.2: Distribution of securities

Another important aspect of the dataset is determined by the predictor *initiator*. This predictor specifies whether the trade, which caused the liquidity shock, is a sell or buy trade. Buy and sell initiated shocks have different behaviors. Therefore, it is essential to split the dataset into two categories according to the *initiator*, and build different predictive models for each category. Figure 4.3 shows the distribution of securities based on the two categories (Sell/Buy).

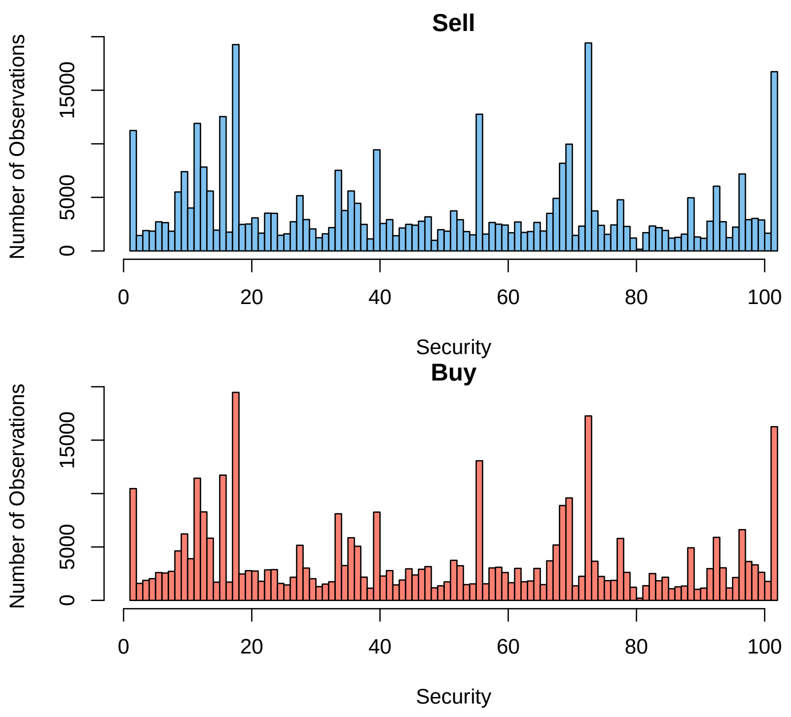


Figure 4.3: Distribution of securities based on the initiator (Sell/Buy)

Given that the provided dataset has an unbalanced distribution of securities, it is expected that the distributions of bid/ask prices and trade volume are significantly different. Figure 4.4 shows the Ask50's box and whisker diagrams for each security. The distribution of trade volume per security is shown in Figure 4.5.

This difference in distributions leads to the skewness of predictors. Figure 4.6 shows examples of skewed predictors. In these examples, the predictors are right-skewed, meaning that there is a greater concentration of data points at relatively small values and a small number of large values. These are some of the issues that will be addressed in the following sections.

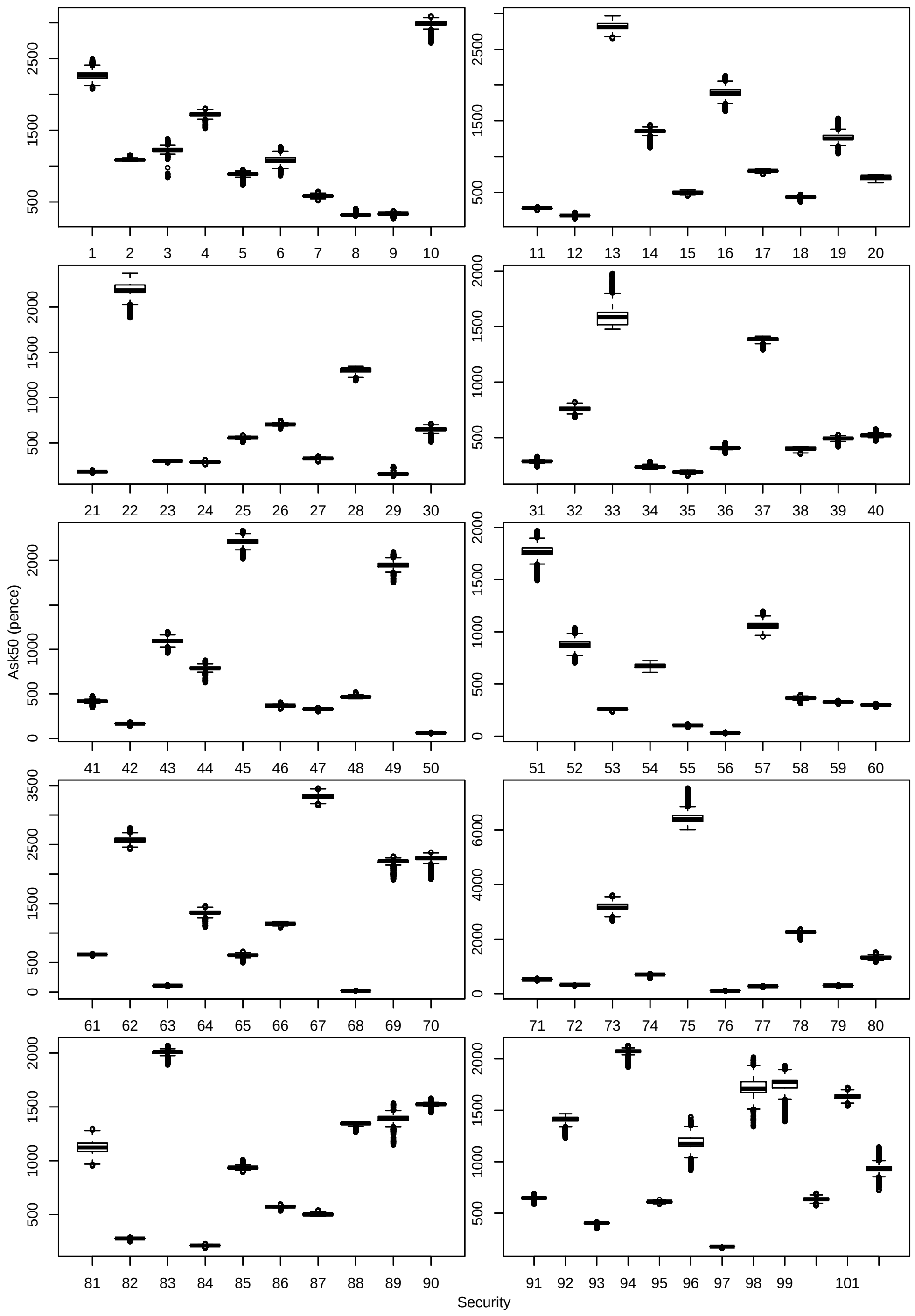


Figure 4.4: Ask50 per security

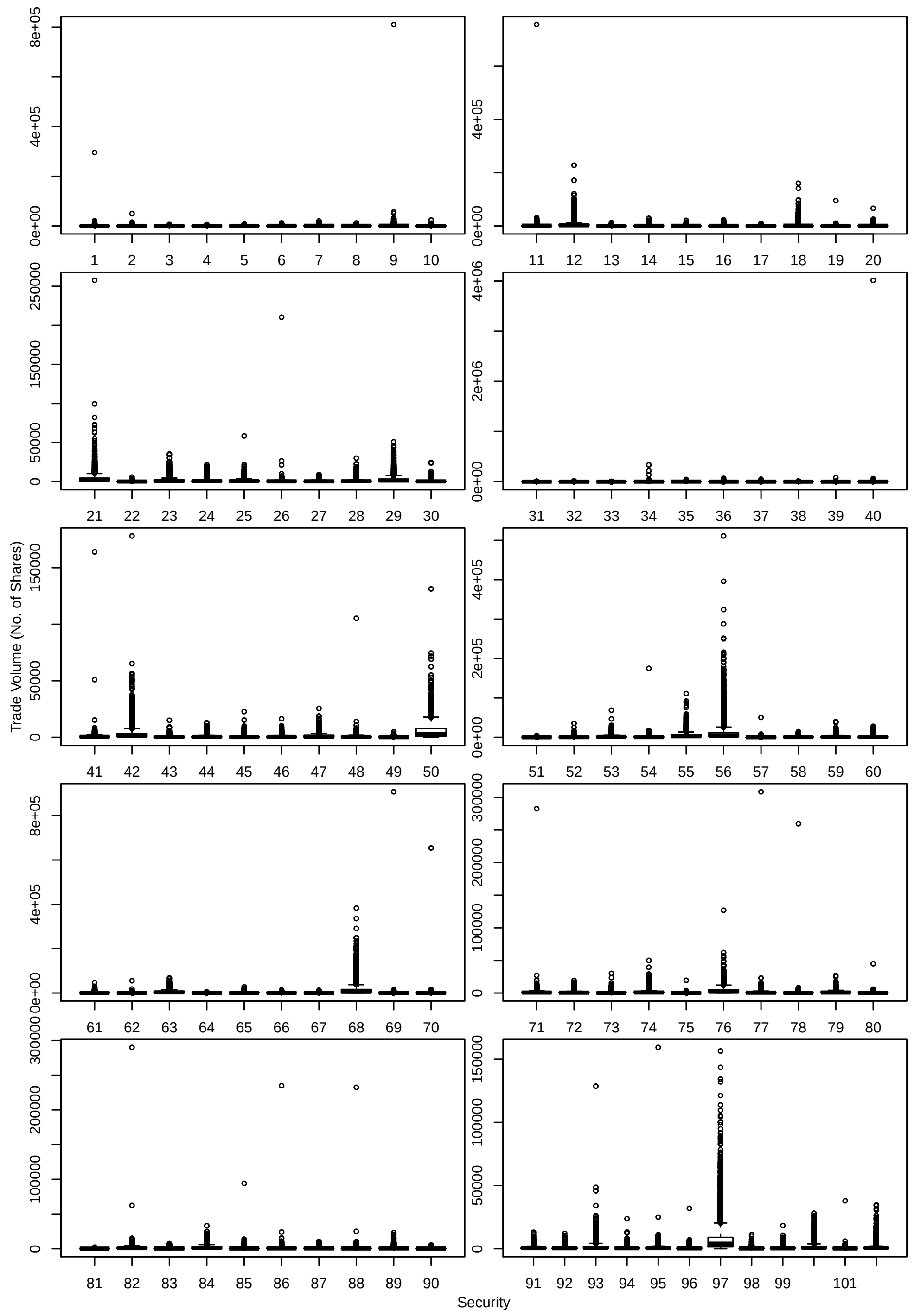


Figure 4.5: Trade volume causing the liquidity shock (per security)

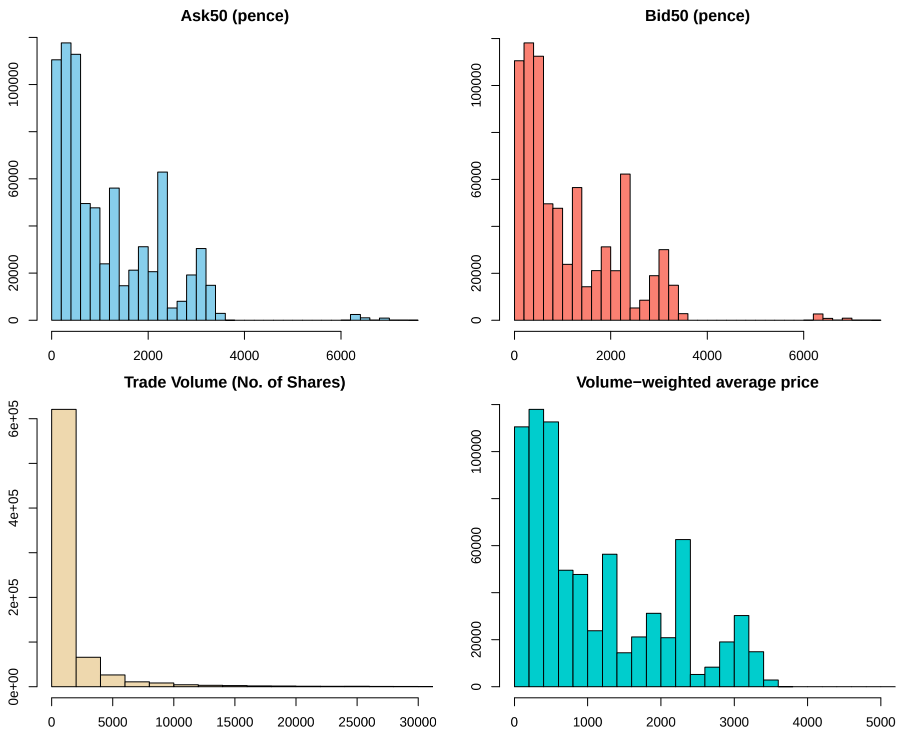


Figure 4.6: Skewness of predictors

4.3.2 Feature Engineering

Feature engineering, how the predictors are encoded, is an important part of data processing, which often has a significant impact on the performance of models. Some of the feature engineering tasks, such as adding/removing predictors, are informed by the modeler's knowledge of the problem domain at hand.

Adding Predictors

Based on the author's basic knowledge of algorithmic trading and stock markets, 74 new predictors are engineered. Table 4.2 presents the new predictors and their description.

Removing Predictors

To decreased computational time and complexity, some of the raw dataset's predictors are removed based on the author's hypotheses that these predictors have no information, and can be eliminated without compromising the performance of the models. The removed predictors include *row_id*, *security_id*, *initiator*, and *time*1 to *time*49

Table 4.2: Engineered predictors

`Predictor name`	`Description`	`Type`
shock_time_interval	Liquidity shock time grouped in hours	String
bid_mean	The mean of best buy price (pence)	Double
bid_sd	The standard deviation of best buy price (pence)	Double
ask_mean	The mean of best buy price (pence)	Double
ask_sd	The standard deviation of best buy price (pence)	Double
spread$\langle t\rangle$	The difference between (bid-ask) (pence) at event time t	Double
spread_mean	The mean of spread (pence)	Double
spread_sd	The standard deviation of spread (pence)	Double
bid_EMA3	The Exponential Moving Average (EMA) of the last 3 bid prices (pence)	Double
bid_EMA5	The EMA of the last 5 bid prices (pence)	Double
bid_EMA10	The EMA of the last 10 bid prices (pence)	Double
bid_low	The lowest bid price (pence)	Double
bid_high	The highest bid price (pence)	Double
ask_EMA3	The EMA of the last 3 ask prices (pence)	Double
ask_EMA5	The EMA of the last 5 ask prices (pence)	Double
ask_EMA10	The EMA of the last 10 ask prices (pence)	Double
ask_low	The lowest ask price (pence)	Double
ask_high	The highest ask price (pence)	Double
spread_EMA3	The EMA of the last 3 spreads (pence)	Double
spread_EMA5	The EMA of the last 5 spreads (pence)	Double
spread_EMA10	The EMA of the last 10 spreads (pence)	Double
spread_low	The lowest spread (pence)	Double
spread_high	The highest spread (pence)	Double
tradeV_pt_count_ratio	The ratio between trade_volume/p_tcount	Double
tradeVW_pvalue_ratio	The ratio between trade_vwap/p_value	Double

Binning Predictors

The $time\langle t \rangle$ predictors measures time at event t in HH:MM:SS.mmm. To simplify the liquidity shock's time ($time50$), it is binned into intervals of the form HH:MM to cover the daily trading hours (e.g. 08:00-8:59, 09:00-9:59, ..., 16:00-16:59). The newly created predictor, *shock_time_interval*, is used to replace the $time50$. The author hypothesizes that the liquidity shock's time could be a significant factor in predicting the resiliency of the London Stock Exchange since it indirectly captures the status of other stock markets worldwide (e.g. opening, closing, etc.).

Converting from Categorical to Dummy Predictors

The categorical predictors, *shock_time_interval* and $transtype1$ to $transtype50$, are re-encoded into smaller bits of information called "dummy variables". Each category gets its own dummy variable that is a zero/one indicator for that group. This step is mandatory in order to be able to apply data transformation to the individual predictors.

4.3.3 Data Transformations for Individual Predictors

The exploratory data analysis, Section 4.3.1, revealed the skewness of most predictors and significant differences in their scales. Some of the learning algorithms used in this case study, such as artificial neural networks and support vector machines, require the predictors to have a common scale. Resolving predictors' skewness is another issue that should be investigated. This section presents the two major data transformations performed.

Centering and Scaling

The predictors are centered by subtracting their average value from all the values. The centered predictors have a zero mean. Then, the predictors are scaled by dividing each value by the predictor's standard deviation. The scaled predictors have a common standard deviation of one. Performing both operations is commonly referred to as normalization. It should be noted that if data transformation is required to resolve skewness (e.g. Box-Cox transformation), it should be performed first, and then, followed by centering and scaling.

Resolving Distributional Skewness

The Box-Cox transformation method [10] is used to resolve the predictors skewness. However, as will be shown in Section 4.6, the predictive accuracy of some learning algorithms is worsened by incorporating Box-Cox transformation in the data pre-processing pipeline. On the other hand, Box-Cox transformation improves the predictive accuracy of other learning algorithms. Therefore, the modeler should investigate the predictive accuracy of the models with and without data transformation.

4.3.4 Data Partition

The proper allocation of data to different tasks (e.g., model/feature selection, model training, model fusion, performance evaluation) is one of the important aspects of ensemble predictive modeling. Figure 4.7 shows the partition of the available data. Stratified sampling is used to maintain similar class distribution across all data sets.

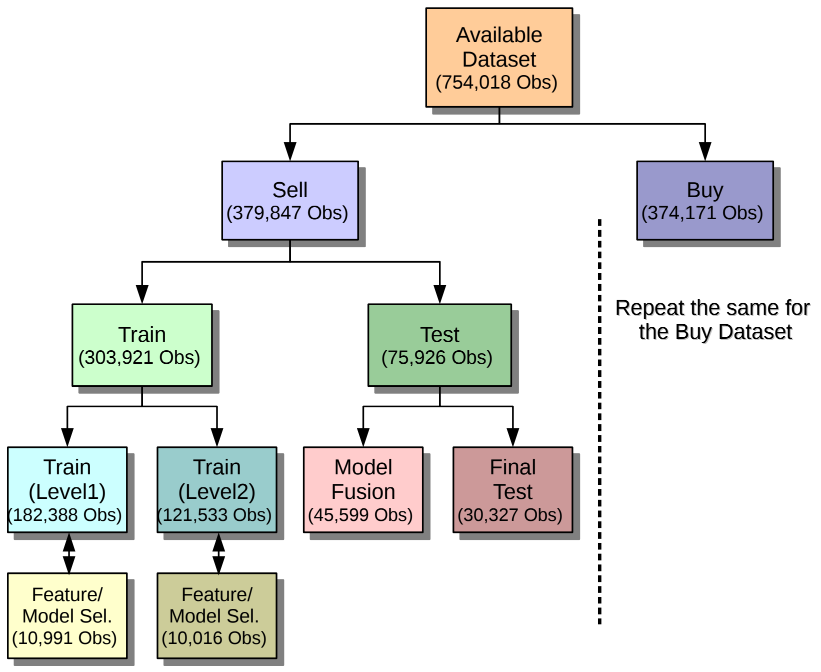

Available Dataset (754,018 Obs)
Sell (379,847 Obs)
Buy (374,171 Obs)
Train (303,921 Obs)
Test (75,926 Obs)
Repeat the same for the Buy Dataset
Train (Level1) (182,388 Obs)
Train (Level2) (121,533 Obs)
Model Fusion (45,599 Obs)
Final Test (30,327 Obs)
Feature/ Model Sel. (10,991 Obs)
Feature/ Model Sel. (10,016 Obs)

Figure 4.7: Data splitting

4.4 Model Building

Predicting the post-liquidity shock's 50 Ask/50 Bid prices is a multi-target regression task that can be formally described as follows:

Let X and Y be two random vectors, where X consists of d input predictors/variables $X_1, X_2, ..., X_d$ and Y consists of m target outcomes $Y_1, Y_2, ..., Y_m$. The samples of the form (X, Y) are assumed to be iid (independent and identically distributed) according to the joint probability distribution $P(X, Y)$ on $X \times Y$ where $X = R^d$ and $Y = R^m$ are the input and output space. In a sample (x, y), $x = [x_1, x_2, ..., x_d]$ and $y = [y_1, y_2, ..., y_m]$ are the realizations of X and Y respectively. Given a set $D = \{(x^1, y^1), ..., (x^n, y^n)\}$ of n training examples, the goal is to learn a model(s) $h : X \rightarrow Y$ such that given a an input vector x^p, it is able to predict an output vector $\hat{y}^p = h(x^p)$, which best approximates the true output vector y^p [87].

For this case study, we divide the input and output spaces based on the Ask/Bid prices such that the goal is to learn a model(s) for each group $h_{Ask} : X_{Ask} \rightarrow Y_{Ask}$ and $h_{Bid} : X_{Bid} \rightarrow Y_{Bid}$. The X_{Ask} and X_{Bid} input spaces are to be determined using feature selection, Section 4.5. The Y_{Ask} and Y_{Bid} are respectively represented by $Ask_{51}, Ask_{52}, ..., Ask_{100}$ and $Bid_{51}, Bid_{52}, ..., Bid_{100}$.

This section describes the modeling strategies used to predict the post-liquidity shock's 50 Ask/50 Bid prices, and systematically inject model diversity at data, feature, and structural levels. Model diversity at learning algorithm level is achieved by implementing these modeling strategies using several learning algorithms.

4.4.1 Single-model Strategy

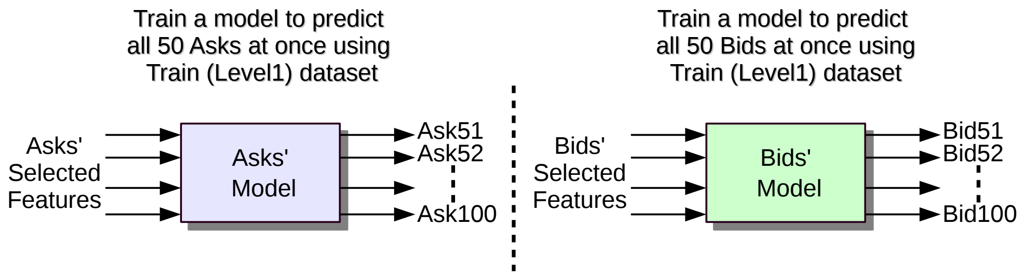

Train a model to predict all 50 Asks at once using Train (Level1) dataset
Asks' Selected Features
Asks' Model
Ask51
Ask52
Ask100
Train a model to predict all 50 Bids at once using Train (Level1) dataset
Bids' Selected Features
Bids' Model
Bid51
Bid52
Bid100

Figure 4.8: Single-model strategy

In the single-model strategy, shown in Figure 4.8, one model is trained for each price ($M_{Ask,level_1}/M_{Bid,level_1}$) to predict all 50 outcomes at once. This requires the development of only two models per initiator's dataset (Sell/Buy).

The strategy is represented by the equations:

$$\hat{Y}_{Ask,level_1} = M_{Ask,level_1}(X_{Ask,level_1}) \tag{4.1}$$

$$\hat{Y}_{Bid,level_1} = M_{Bid,level_1}(X_{Bid,level_1}) \tag{4.2}$$

Where $\hat{Y}_{Ask,level_1}$ and $\hat{Y}_{Bid,level_1}$ are matrices of the predicted $Ask_{51,level_1}$, $Ask_{52,level_1}$, ..., $Ask_{100,level_1}$ and $Bid_{51,level_1}$, $Bid_{52,level_1}$, ..., $Bid_{100,level_1}$ respectively. The $X_{Ask,level_1}/X_{Bid,level_1}$ are the selected features for each price. The $_{level_1}$ refers to using the partition of the available data for $level_1$ models, see Figure 4.7. $Level_2$ data partition will be used to train models in the cascading-model strategies, Section 4.4.3.

In theory, the learning algorithm used to implement this strategy should be able to capture the dependency between the outcomes, and hence, produce the best performance (minimum RMSE). However, few learning algorithms are capable of multi-target/multivariate regression. An example for such algorithms is the feedforward neural network, which will be used to implement this strategy using $level_1$ training dataset, Section 4.6.1.

4.4.2 Multi-model Strategy

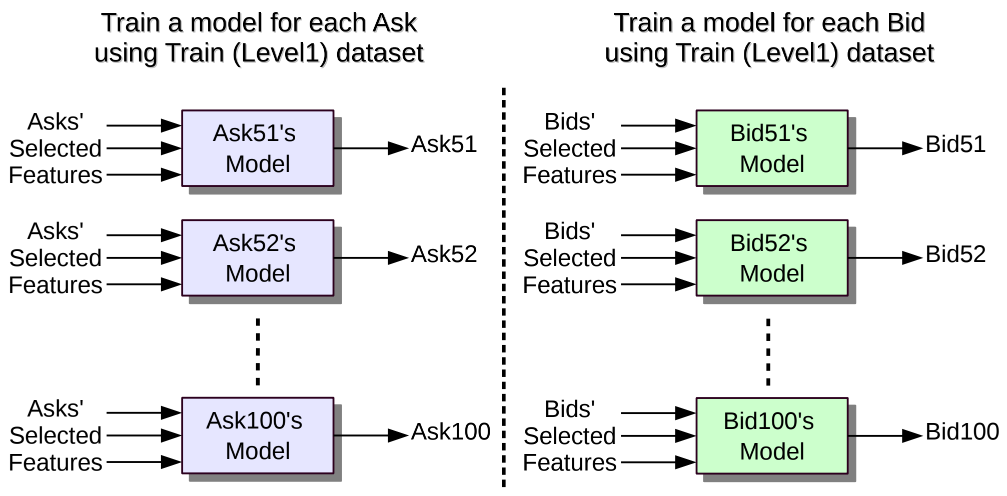


Figure 4.9: Multi-model strategy

The multi-model strategy decomposes the multi-target regression problem into single-target regression problems. Then, it trains a separate model for each post-liquidity shock's Ask/Bid price, as shown in Figure 4.9, using the same selected features for all models.

The 50 Asks' models can are represented by the following equations:

$$\hat{Y}_{Ask_{51},level_1} = M_{Ask_{51},level_1}(X_{Ask,level_1}) \tag{4.3}$$
$$\hat{Y}_{Ask_{52},level_1} = M_{Ask_{52},level_1}(X_{Ask,level_1}) \tag{4.4}$$
$$\dots$$
$$\hat{Y}_{Ask_{100},level_1} = M_{Ask_{100},level_1}(X_{Ask,level_1}) \tag{4.5}$$

Similarly, the 50 Bids' models are represented by the equations:

$$\hat{Y}_{Bid_{51},level_1} = M_{Bid_{51},level_1}(X_{Bid,level_1}) \tag{4.6}$$
$$\hat{Y}_{Bid_{52},level_1} = M_{Bid_{52},level_1}(X_{Bid,level_1}) \tag{4.7}$$
$$\dots$$
$$\hat{Y}_{Bid_{100},level_1} = M_{Bid_{100},level_1}(X_{Bid,level_1}) \tag{4.8}$$

where $\hat{Y}_{Ask_t,level_1}/\hat{Y}_{Bid_t,level_1}$ and $M_{Ask_t,level_1}/M_{Bid_t,level_1}$ are respectively the predicted Ask/Bid prices and trained Ask/Bid models at event t, $t \in [51, 100]$, using $level_1$ data partition. The decomposition of the multi-target regression problem into single-target regression problems gives the modeler unlimited options when it comes to what learning algorithm can be used to implement this strategy. In Section 4.6.2, four different machine learning algorithms, Multivariate Adaptive Regression Splines (MARS), stochastic Gradient Boosting Machines (GBM), Support Vector Machine (SVM), and Random Forest (RF) will be investigated to implement the multi-model strategy.

4.4.3 Cascading-model Strategies

Similar to the multi-model strategy, the two cascading-model strategies train a separate model for each Ask/Bid price. However, they use different features for each model. The strategies incorporate model fusion to build new models.

The cascading-model strategy 1, shown in Figure 4.10, trains new models for each Ask/Bid price on an expanding features set by chaining the predicted post-liquidity shock's prices Ask_t/Bid_t, $t \in [51, 99]$, using models trained on $level_1$ data, with the selected features from $level_2$ data partition. To avoid over-fitting, it is essential to use the models trained on $level_1$ data to predict the prices using $level_2$ selected features (e.g. $M_{Ask_{51},level_1}(X_{Ask,level_2})$).

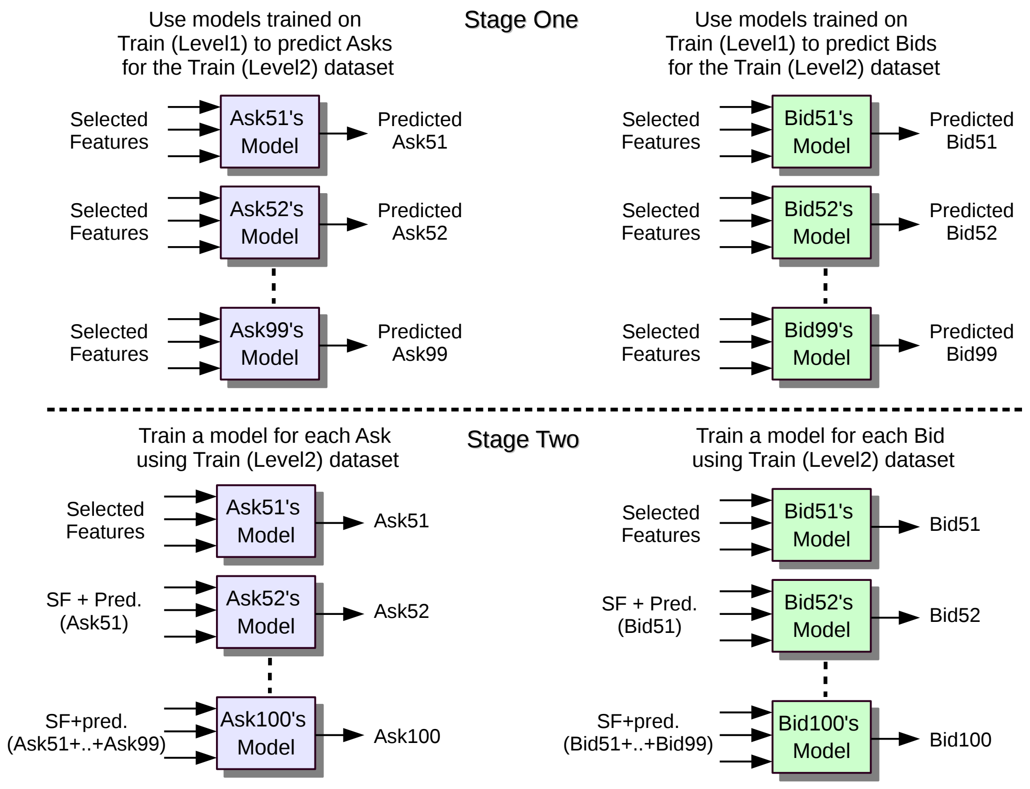
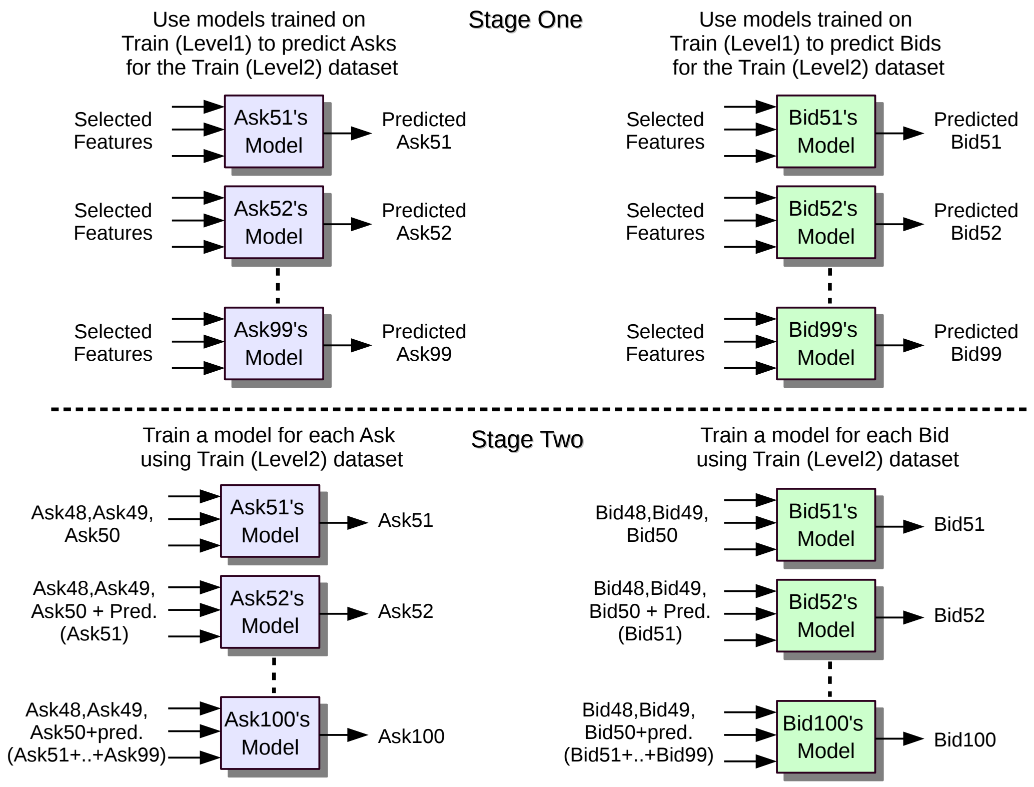

Stage One
Use models trained on Train (Level1) to predict Asks for the Train (Level2) dataset
Use models trained on Train (Level1) to predict Bids for the Train (Level2) dataset
Selected Features
Ask51's Model
Predicted Ask51
Ask52's Model
Predicted Ask52
Ask99's Model
Predicted Ask99
Bid51's Model
Predicted Bid51
Bid52's Model
Predicted Bid52
Bid99's Model
Predicted Bid99
Stage Two
Train a model for each Ask using Train (Level2) dataset
Train a model for each Bid using Train (Level2) dataset
SF + Pred. (Ask51)
SF+pred. (Ask51+..+Ask99)
Ask51
Ask52
Ask100's Model
Ask100
SF + Pred. (Bid51)
SF+pred. (Bid51+..+Bid99)
Bid51
Bid52
Bid100's Model
Bid100

Figure 4.10: Cascading-model (strategy 1)

The cascading-model strategy 2, shown in Figure 4.11, uses the same approach but the expanding features set is formed by chaining the predicted post-liquidity shock's prices with only the last three prices just before the liquidity shock.



Stage One
Use models trained on Train (Level1) to predict Asks for the Train (Level2) dataset
Use models trained on Train (Level1) to predict Bids for the Train (Level2) dataset
Selected Features
Ask51's Model
Predicted Ask51
Ask52's Model
Predicted Ask52
Ask99's Model
Predicted Ask99
Bid51's Model
Predicted Bid51
Bid52's Model
Predicted Bid52
Bid99's Model
Predicted Bid99
Stage Two
Train a model for each Ask using Train (Level2) dataset
Train a model for each Bid using Train (Level2) dataset
Ask48,Ask49, Ask50
Ask51's Model
Ask51
Ask48,Ask49, Ask50 + Pred. (Ask51)
Ask52's Model
Ask52
Ask48,Ask49, Ask50+pred. (Ask51+..+Ask99)
Ask100's Model
Ask100
Bid48,Bid49, Bid50
Bid51's Model
Bid51
Bid48,Bid49, Bid50 + Pred. (Bid51)
Bid52's Model
Bid52
Bid48,Bid49, Bid50+pred. (Bid51+..+Bid99)
Bid100's Model
Bid100

Figure 4.11: Cascading-model (strategy 2)

The cascading-model strategy 1 Ask/Bid models are represented by the equations:

$$\hat{Y}_{Ask_{51},level_2} = M_{Ask_{51},level_2}(X_{Ask,level_2}) \tag{4.9}$$

$$\hat{Y}_{Ask_{52},level_2} = M_{Ask_{52},level_2}([X_{Ask,level_2}, M_{Ask_{51},level_1}(X_{Ask,level_2})]) \tag{4.10}$$

$$\ldots$$

$$\begin{aligned}\hat{Y}_{Ask_{100},level_2} = M_{Ask_{100},level_2}([X_{Ask,level_2}, M_{Ask_{51},level_1}(X_{Ask,level_2}),\\ M_{Ask_{52},level_1}(X_{Ask,level_2}),\\ \ldots,\\ M_{Ask_{99},level_1}(X_{Ask,level_2})])\end{aligned} \tag{4.11}$$

$$\hat{Y}_{Bid_{51},level_2} = M_{Bid_{51},level_2}(X_{Bid,level_2}) \tag{4.12}$$
$$\hat{Y}_{Bid_{52},level_2} = M_{Bid_{52},level_2}([X_{Bid,level_2}, M_{Bid_{51},level_1}(X_{Bid,level_2})]) \tag{4.13}$$
$$...$$
$$\begin{aligned}\hat{Y}_{Bid_{100},level_2} = M_{Bid_{100},level_2}([X_{Bid,level_2}, M_{Bid_{51},level_1}(X_{Bid,level_2}),\\ M_{Bid_{52},level_1}(X_{Bid,level_2}),\\ ...,\\ M_{Bid_{99},level_1}(X_{Bid,level_2})])\end{aligned} \tag{4.14}$$

The cascading-model strategy 2 Ask/Bid models are represented by the equations:

$$\hat{Y}_{Ask_{51},level_2} = M_{Ask_{51},level_2}([[Ask_{48}, Ask_{49}, Ask_{50}]_{level_2}]) \tag{4.15}$$
$$\hat{Y}_{Ask_{52},level_2} = M_{Ask_{52},level_2}([[Ask_{48}, Ask_{49}, Ask_{50}]_{level_2}, M_{Ask_{51},level_1}(X_{Ask,level_2})]) \tag{4.16}$$
$$...$$
$$\begin{aligned}\hat{Y}_{Ask_{100},level_2} = M_{Ask_{100},level_2}([[Ask_{48}, Ask_{49}, Ask_{50}]_{level_2},\\ M_{Ask_{51},level_1}(X_{Ask,level_2}),\\ M_{Ask_{52},level_1}(X_{Ask,level_2}),\\ ...,\\ M_{Ask_{99},level_1}(X_{Ask,level_2})])\end{aligned} \tag{4.17}$$

$$\hat{Y}_{Bid_{51},level_2} = M_{Bid_{51},level_2}([[Bid_{48}, Bid_{49}, Bid_{50}]_{level_2}]) \tag{4.18}$$
$$\hat{Y}_{Bid_{52},level_2} = M_{Bid_{52},level_2}([[Bid_{48}, Bid_{49}, Bid_{50}]_{level_2}, M_{Bid_{51},level_1}(X_{Bid,level_2})]) \tag{4.19}$$
$$...$$
$$\begin{aligned}\hat{Y}_{Bid_{100},level_2} = M_{Bid_{100},level_2}([[Bid_{48}, Bid_{49}, Bid_{50}]_{level_2},\\ M_{Bid_{51},level_1}(X_{Bid,level_2}),\\ M_{Bid_{52},level_1}(X_{Bid,level_2}),\\ ...,\\ M_{Bid_{99},level_1}(X_{Bid,level_2})])\end{aligned} \tag{4.20}$$

4.4.4 Market-based vs Security-based Approach

Given our multi-target regression problem for 102 securities/stocks from the London Stock Exchange, there are two possible approaches: market-based or security-based. In a market-based approach, Ask/Bid models are developed to predict the post-liquidity shock's prices for all 102 securities. In a security-based approach, different Ask/Bid models are developed to predict the post-liquidity shock's prices for each security of the 102 securities. Table 4.3 lists the number of required models to be developed for each approach using only one learning algorithm. Given the massive number of models required by the security-based approach (30,804 models, not including the model selection step), the market-based approach is chosen to implement our modeling strategies for two main reasons. Firstly, to minimize the cost of using the Amazon Elastic Cloud EC2 for model selection, and training time. Secondly, we hypothesize that the market-based approach will reasonably capture the dynamic of the stock market following liquidity shocks, and any performance gains might be achieved by the security-based approach will be diminished by its computations cost.

Table 4.3: Number of required models per strategy for market-based and security-based approaches (for only one learning algorithm)

Modeling strategy	Market-based models	Security-based models
Single-model	$2(Ask/Bid)$	$2(Ask/Bid) \times 102 = 204$
Multi-model	$2(Ask/Bid) \times 50 = 100$	$2(Ask/Bid) \times 50 \times 102 = 10200$
cascading-model st1	$2(Ask/Bid) \times 50 = 100$	$2(Ask/Bid) \times 50 \times 102 = 10200$
cascading-model st2	$2(Ask/Bid) \times 50 = 100$	$2(Ask/Bid) \times 50 \times 102 = 10200$
Total models	302	$30,804$

4.5 Feature Selection

Given the large number of observations and relatively high-dimensional dataset, a Genetic Algorithm (GA) feature selection approach is employed to reduce the computations cost of training a large number of models. To handle the intensive computations, an Amazon EC2 instance with 32 cores and 65 GB RAM is used. To minimize over-fitting the GA to the features, 3-fold cross-validation (formed using stratified sampling to maintain the same securities distributions across all folds), shown in Figure 4.12, is used for Ask/Bid prices' features. The GA has maximum generations of 100, a population of 5 per generation, 0.8 crossover probability, and 0.1 mutation probability. To assess the performance of the selected features for each population, a linear regression model is fit, and the RMSE and R^2 are calculated for the out-of-fold samples. The set of features with the minimum average of RMSE are selected.

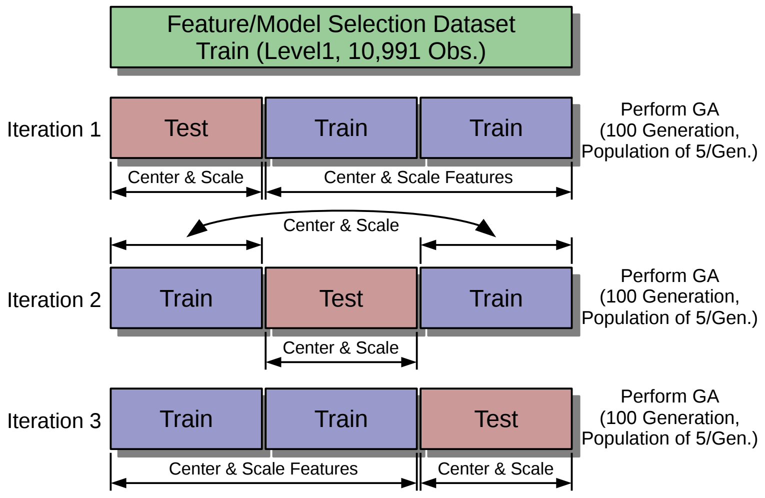


Figure 4.12: GA feature selection using 3-fold cross-validation

It should be noted that combining GA with model's performance (e.g. linear regression, random forest, etc.) and cross-validation for relatively a large dataset (e.g. 10,991 observation and 184 features) requires powerful resources to handle the intensive computations. For example, we started with 10-fold cross-validation and a population of 50 per generation, but even a powerful Amazon EC2 instance with 32 cores and 65 GB RAM could not handle the computations and crashed. We had to reduce the number of folds and population per generation several times to complete the feature selection process.

Ideally, the feature selection process should be applied to each post-liquidity shock's Ask/Bid price. However, to reduce the computations cost, we implement the feature selection step to only the Ask_{100}/Bid_{100} prices, and use the selected features for the remaining post-liquidity shock's prices.

Figure 4.13 shows the Ask_{100}'s GA feature selection. Out of 184 features/predictors, 34 were selected. The external performance of the selected features is RMSE = 1.775 and R^2=1.0. The Bid_{100}'s GA feature selection performance is shown in Figure 4.14. 44 predictors were selected out of 184 with RMSE = 1.834 and R^2=1.0. The Ask/Bid selected features' names are shown in Figure 4.15.

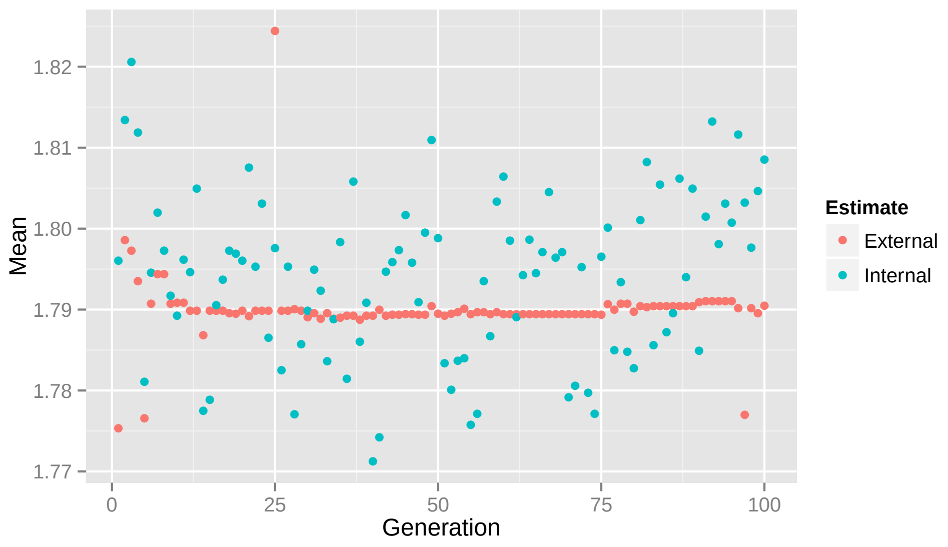


Figure 4.13: Ask100's GA feature selection

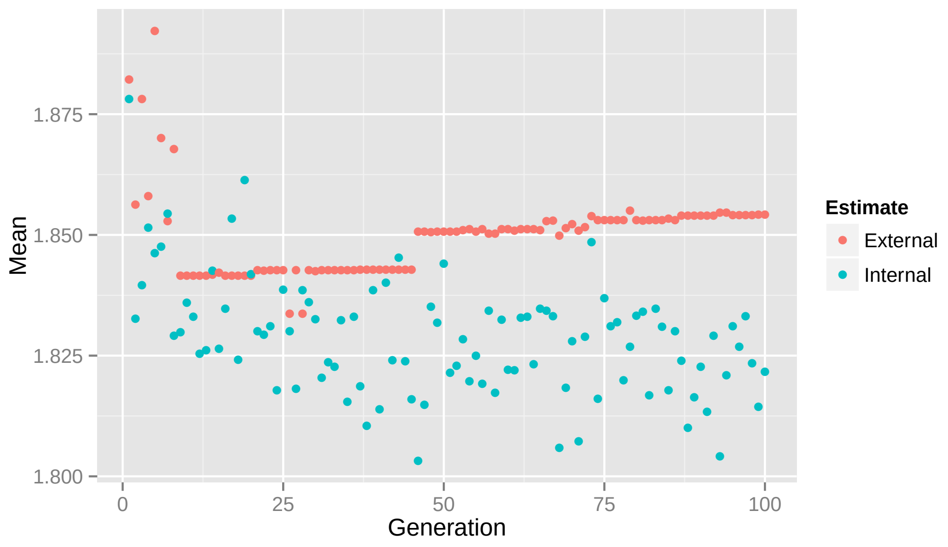
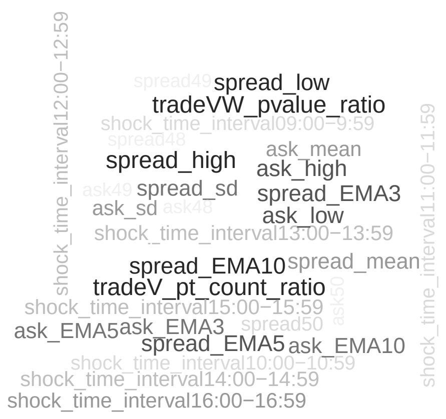


Figure 4.14: Bid100's GA feature selection



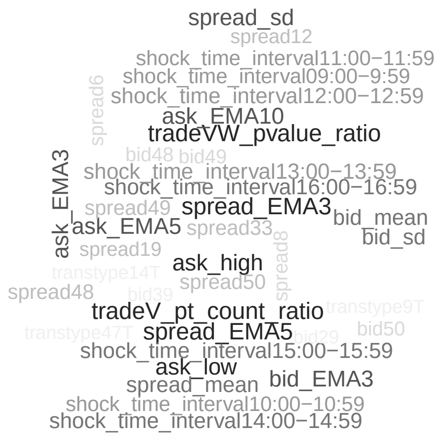


Figure 4.15: Ask100's (left)/ Bid100's (right) selected features

4.6 Model Selection and Training

As advised by the proposed framework for ensemble predictive modeling, a proper sampling technique, such as cross-validation with stratified sampling, should be employed to produce quantitative assessments of the models to help us make the choice. In the first part of this section, the focus is on selecting the best hyper-parameters and using them to train the models. The selection of learning algorithms will be discussed at the end of the section.

Similar to the feature selection step, model selection should ideally be applied to all post-liquidity shock's Ask/Bid prices. However, to reduce the computations cost, the model selection step is implemented to the Ask_{100}/Bid_{100} prices for each learning algorithm and modeling strategy. Given the intensive computations involved, 10-fold cross-validation (with stratified sampling) is applied to a relatively small sample of the dataset ($level_1$ dataset: 10,991 observations and $level_2$ dataset: 10,016 observations), as shown in Figure 4.16, to determine the best hyper-parameters for each learning algorithm. To understand the effect of different combinations of hyper-parameters values, the performance of the hyper-parameters grid-search is visualized. The performance's uncertainty of the optimal model is also visualized using its resampling accuracy. Another important aspect this section investigates is the effect of data pre-processing with and without data transformation (e.g. Box-Cox transformation to resolve the predictors' skewness) on the performance of the learning algorithms. The model selection tasks are performed on an Amazon EC2 instance with 32 cores and 65 GB RAM.

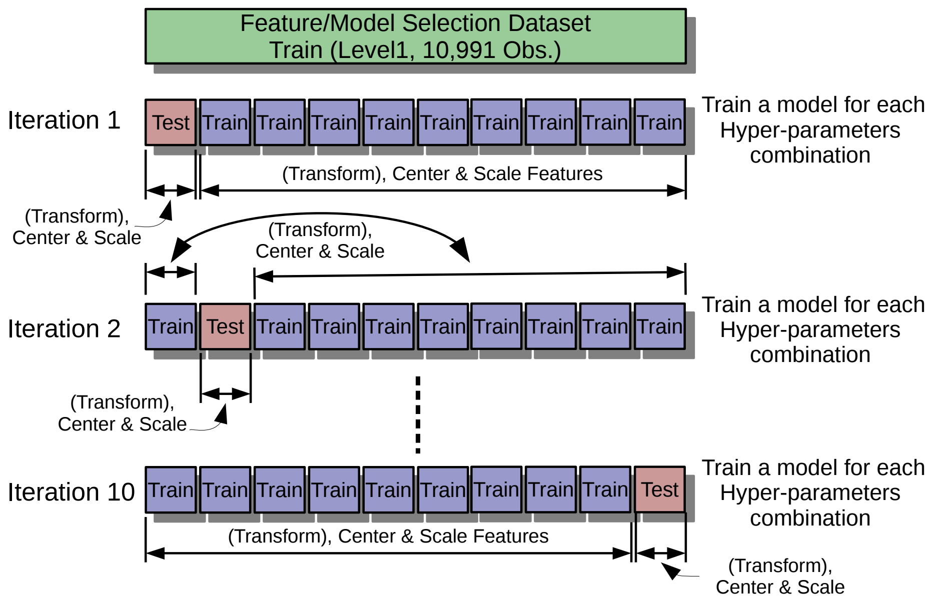
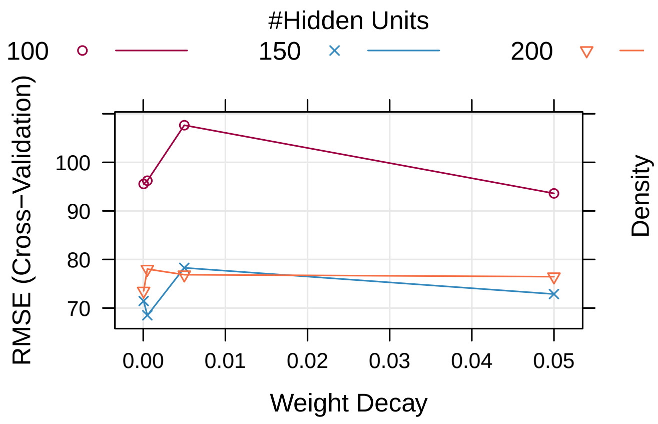

Feature/Model Selection Dataset
Train (Level1, 10,991 Obs.)
Iteration 1
Test
Train
Train a model for each Hyper-parameters combination
(Transform), Center & Scale Features
(Transform), Center & Scale
Iteration 2
Iteration 10

Figure 4.16: Model selection using 10-fold cross-validation

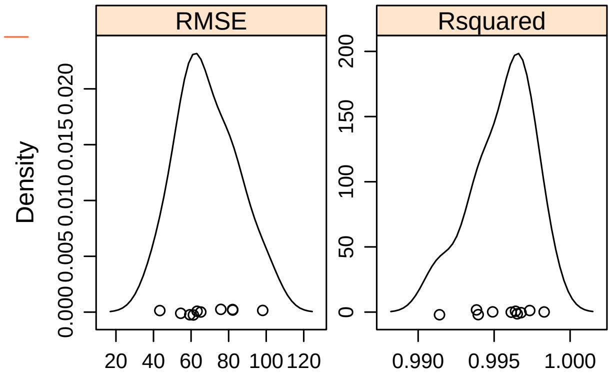
The best hyper-parameters for each learning algorithm are then used to train all post-liquidity models for each Ask/Bid price on larger training data sets ($level_1$ dataset: 182,388 observations and $level_2$ dataset: 121,533 observations). To minimize the financial cost of renting Amazon EC2 instances, the training of models is performed on a local PC with 6 cores and 8 GB RAM.

4.6.1 Single-model Strategy

As discussed in Section 4.4.1, in this strategy one model is trained for each price ($M_{Ask,level_1}$/ $M_{Bid,level_1}$) to predict all post-liquidity shock's 50 prices at once. This strategy is implemented using a feedforward Neural NETwork (NNET), which is one of few learning algorithms capable of multi-target regression. A single-layer feedforward neural network [9] has two main hyper-parameters: the weight decay and number of hidden units. Before they are fed to an NNET, predictors should be on the same scale; hence, they should be normalized (centered and scaled). The effect of predictors' transformation, to resolve skewness before normalization is also investigated. The number of predictors for Ask models is 34 while for Bid models are 44. $Level_1$ model selection dataset, with 10,991 observations, is used to select the Ask/Bid neural network's hyper-parameters, while $level_1$ train dataset, with 182,388 observations, is used to train the Ask/Bid neural networks.

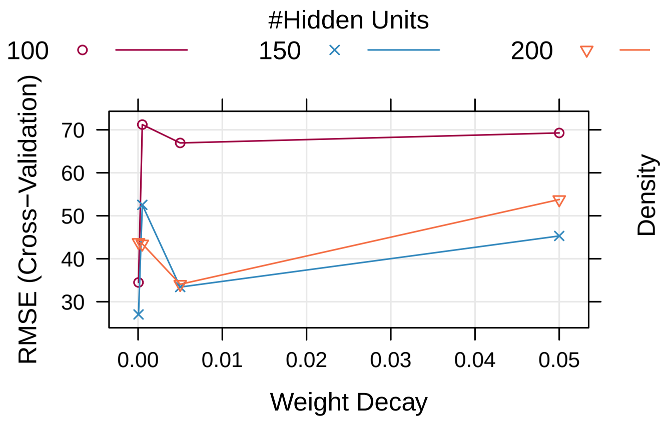


(a) Hyper-parameters grid-search

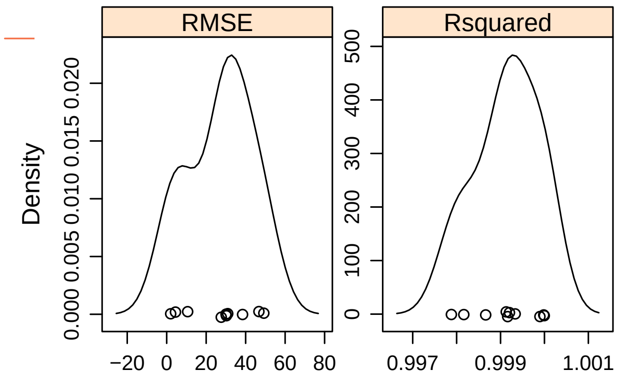


(b) Optimal model resampling accuracy

Figure 4.17: Bid_{100}'s NNET $level_1$ model selection with normalization only



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

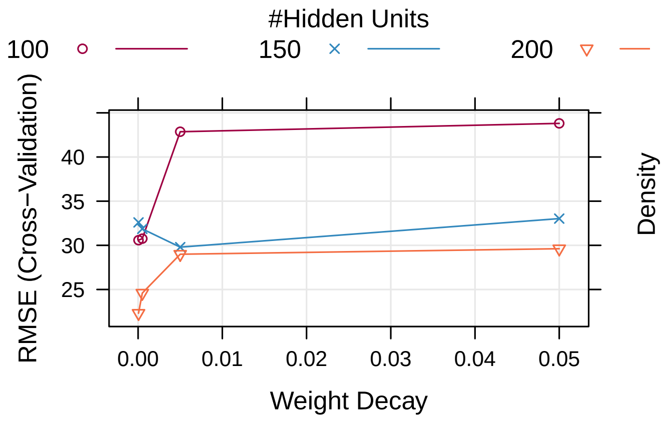
Figure 4.18: Bid_{100}'s NNET $level_1$ model selection with transformation and normalization

Figure 4.17 shows the results of Bid_{100}'s NNET model selection with predictors' normalization only as a data pre-processing step. Figure 4.17a shows the effect of different combinations of the number of hidden units and weight decay, while Figure 4.17b shows the uncertainty of the optimal model's performance using the 10-fold cross-validation accuracy.

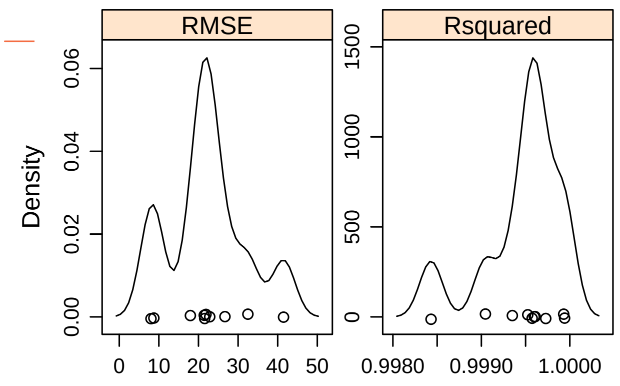
Similarly, Figure 4.18 shows the model selection results of the Bid_{100}'s NNET model with predictors transformed, using Box-Cox method, and then normalized. It should be noted that due to the current technical limitations of the used R language's caret package, the model selection for the NNET models are performed using a single-target (Ask_{100}/Bid_{100}). The optimal hyper-parameters of the NNET Bid_{100} model (with Box-Cox transformation and/or normalization) are presented in Table 4.4. Clearly, applying Box-cox transformation to predictors before normalization has significantly improved the accuracy of Bid_{100}'s NNET model.

Table 4.4: The optimal hyper-parameters of the Bid_{100}'s NNET $level_1$ model (with Box-Cox transformation and/or normalization)

Preprocessing	size	decay	RMSE	R2	RMSE SD	R2 SD
Norm.	150	5e-04	68.50369	0.99556	16.02323	0.00204
Trans. & Norm.	150	5e-05	27.03484	0.99914	16.49858	0.00073



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.19: Ask_{100}'s NNET $level_1$ model selection with transformation and normalization

The same model selection process is repeated for the Ask_{100}'s NNET model with Box-Cox transformation and normalization, as shown in Figure 4.19. The optimal hyper-parameters and their results are presented in Table 4.5.

Table 4.5: The optimal hyper-parameters of the Ask_{100}'s NNET $level_1$ model (with Box-Cox transformation and normalization)

Preprocessing	size	decay	RMSE	R2	RMSE SD	R2 SD
Trans. & Norm.	200	5e-05	22.31050	0.99947	9.99622	0.00045

The optimal Ask_{100}/Bid_{100} hyper-parameters are then used to train the two Asks/Bids NNET models on the level1 train dataset, with 182,388 observations. Given the large number of observations, the NNET's number of iterations becomes critical concerning training time and model's accuracy. To select the best number of iterations, based on model's accuracy (RMSE), several Bid's NNET models, with iterations 1000, 2000, 3000, and 5000, are trained using level1 dataset. Then, a validation dataset is used to compare the accuracy of the models, as shown in Figure 4.20. The best performing number of iterations, 3000, is then used to train the Ask NNET model. Due to its high computations cost, the NNET is only used to implement the single-model strategy.

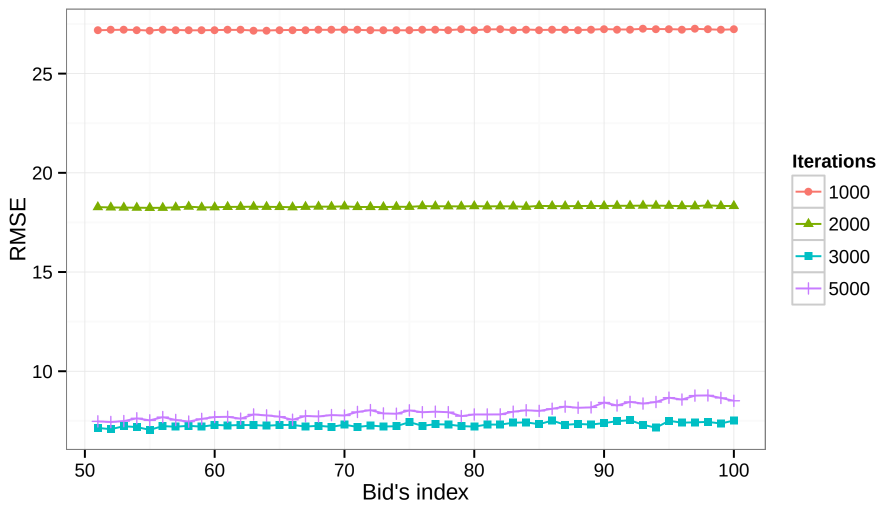


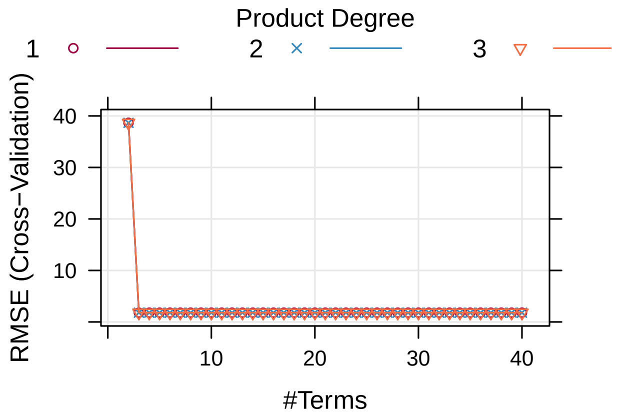
Figure 4.20: Bids NNET's RMSE using different iterations

4.6.2 Multi-model Strategy

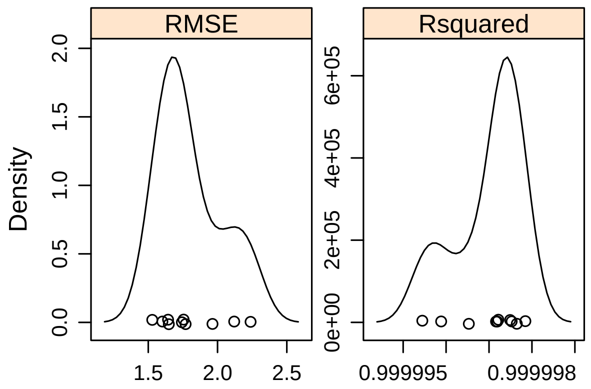
The multi-model strategy, discussed in Section 4.4.2, trains a separate model for each post-liquidity shock's Ask/Bid price. Grid-search and 10-fold cross-validation is applied to the $level_1$ model selection data set (with 10,991 observations and 34/44 selected features for Ask/Bid prices), to choose the best hyper-parameters for four learning algorithms. The algorithms are Multivariate Adaptive Regression Splines (MARS), Stochastic Gradient Boosting Machines (GBM), Support Vector Machines (SVM), and Random Forest (RF). The primary objective of using different learning algorithms is to inject model diversity at the learning algorithm level. A second objective is to compare the performance of these learning algorithms, given a relatively large number of observations, concerning accuracy and computational cost. Model selection across the learning algorithms, based on the outcome of the comparison, is then performed to select algorithms for $level_2$ cascading-model strategies. To have a fair comparison, the samples in the ten folds must be exactly the same for all learning algorithms, and sampled using stratified sampling to maintain the same securities distributions. Similar to the single-model strategy, the effect of predictors' normalization with/without Box-Cox transformation on the performance of the algorithms is investigated.

Model Selection for the MARS Algorithm

The MARS algorithm [28] has two hyper-parameters, the number of retained terms and the degree of predictors involved in the hinge function.

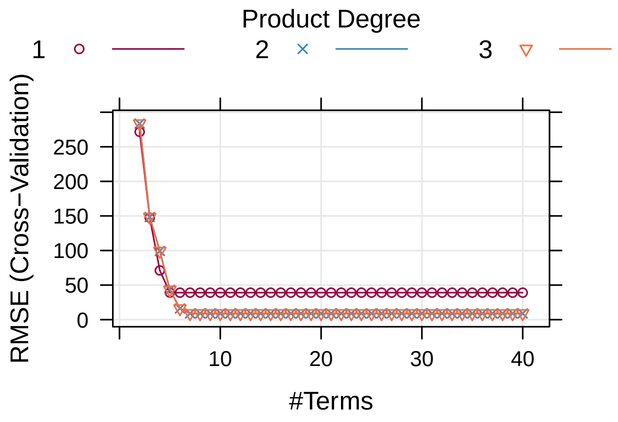


(a) Hyper-parameters grid-search

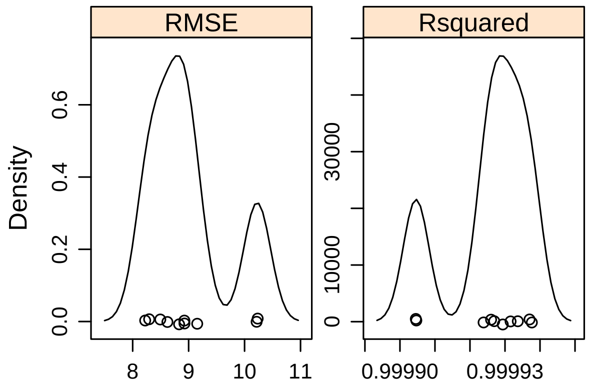


(b) Optimal model resampling accuracy

Figure 4.21: Bid_{100}'s MARS $level_1$ model selection with normalization only



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

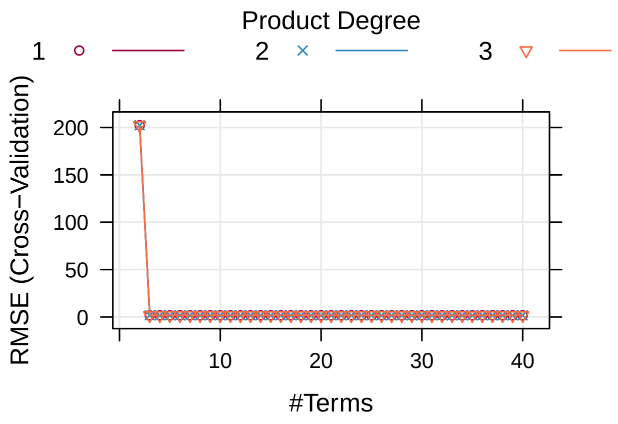
Figure 4.22: Bid_{100}'s MARS $level_1$ model selection with transformation and normalization

The results of model selection for Bid_{100}'s MARS algorithm, with only normalization applied to the predictors, is shown in Figure 4.21. Figure 4.22 shows the same model selection but with both transformation and normalization applied to the predictors. Opposite to the feedforward neural network results, Table 4.4, adding Box-Cox transformation before the normalization of predictors has significantly degraded the accuracy of the MARS algorithm. Table 4.6 lists the optimal hyper-parameters for the two Bid_{100}'s MARS models and their performance.

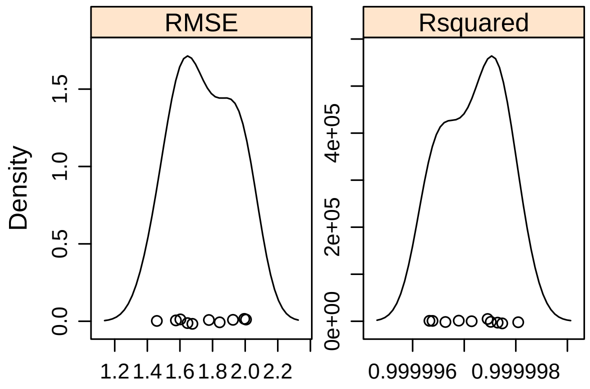
Table 4.6: The optimal hyper-parameters of the Bid_{100}'s MARS $level_1$ model (with Box-Cox transformation and/or normalization)

Preprocessing	#Terms	degree	RMSE	R2	RMSE SD	R2 SD
Norm.	3	1	1.80114	0.99999	0.23290	7.9193e-07
Trans. & Norm.	7	2	8.99145	0.99993	0.71213	1.1913e-05

Similarly, the model selection process, with predictors' normalization, is applied to the Ask_{100} MARS model to tune the algorithm's hyper-parameters. The optimal Ask MARS model's hyper-parameters and its performance are listed in Table 4.7.



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.23: Ask_{100}'s MARS $level_1$ model selection with normalization only

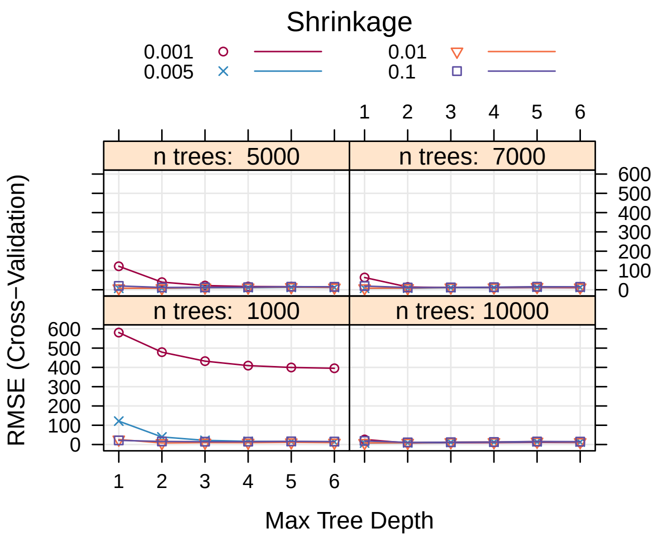
Table 4.7: The optimal hyper-parameters of the Ask_{100}'s MARS $level_1$ model (with normalization only)

Preprocessing	#Terms	degree	RMSE	R2	RMSE SD	R2 SD
Norm.	3	1	1.75041	0.99999	0.18818	5.9686e-07

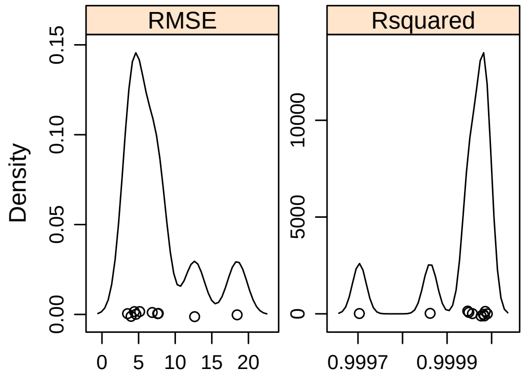
The optimal Ask_{100}/Bid_{100} MARS's hyper-parameters are then used to train 50 Ask/50 Bid MARS models on the $level_1$ training dataset, with 182,388 observations.

Model Selection for the GBM Algorithm

The Stochastic Tree-Based GBM [29] learning algorithm has five main hyper-parameters that control the complexity and performance of the GBM model. These are: the loss function (*distribution*), the number of iterations (*n.trees*), the depth of each tree (*interaction.depth*), the *shrinkage* (or learning rate), and the subsampling rate (*bag.fraction*). For regression problems, the *gaussian* distribution is used. The subsampling rate (*bag.fraction*) of 0.5 is recommended by [69]. The remaining three hyper-parameters are tuned for the Ask_{100}/Bid_{100} GBM models, using 10-fold cross-validation.

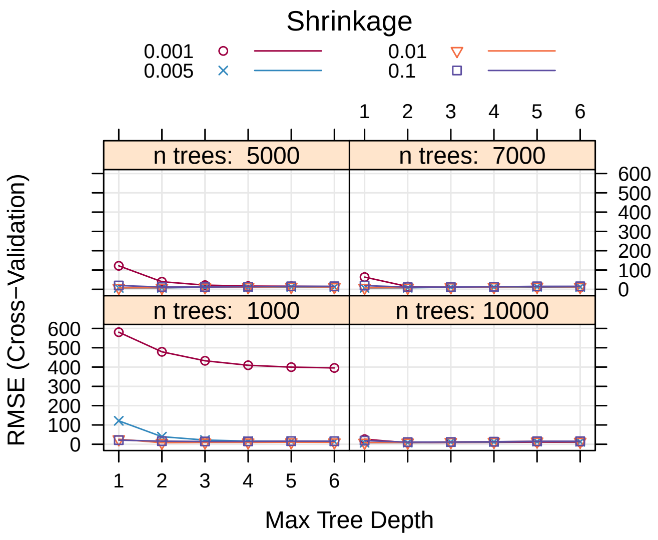


(a) Hyper-parameters grid-search

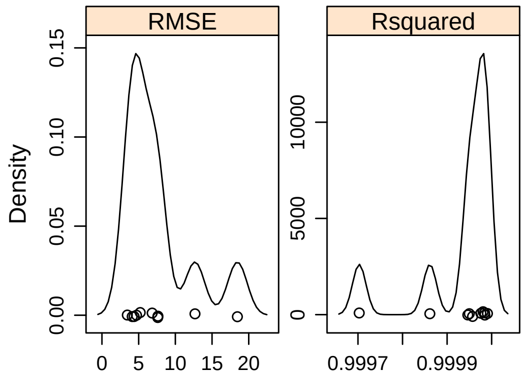


(b) Optimal model resampling accuracy

Figure 4.24: Bid_{100}'s GBM $level_1$ model selection with normalization only



(a) Hyper-parameters grid-search



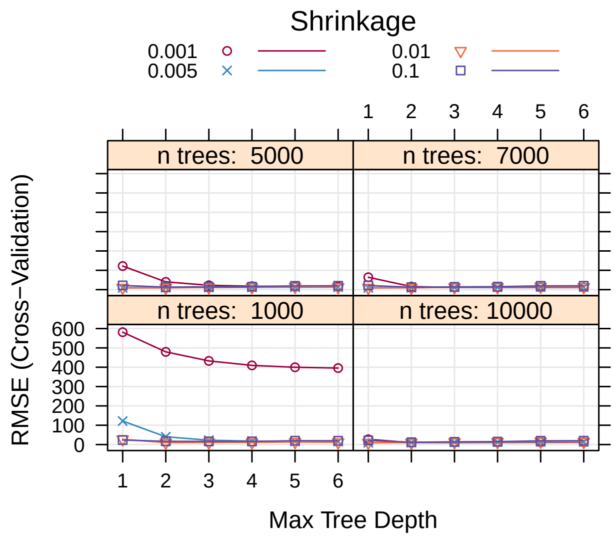
(b) Optimal model resampling accuracy

Figure 4.25: Bid_{100}'s GBM $level_1$ model selection with transformation and normalization

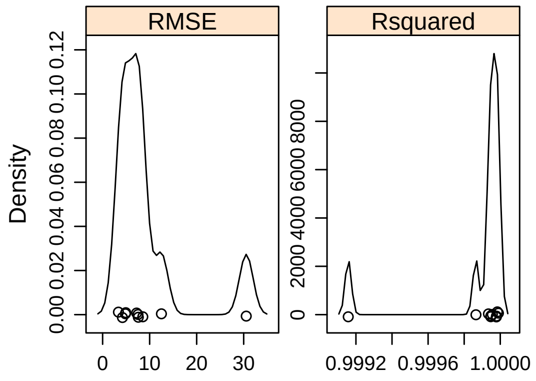
Another important aspect when comparing different models of a learning algorithm, which has an internal element of randomness such as the GBM's subsampling rate, is setting the same random seeds for all models to have the same subsampled observations. Setting the seed becomes especially important if the 10-fold cross-validation is being executed using parallel computing over several numbers of cores (e.g. an Amazon instance with 32 cores). In this case, the same random seeds should be passed to all computing processes. The results of model selection for Bid_{100}'s GBM algorithm with normalization only are shown in Figure 4.24. Figure 4.25 shows the results with both transformation and normalization applied to the predictors. Table 4.8 indicates that applying Box-Cox transformation prior to normalization resulted in the same performance and hyper-parameters.

Table 4.8: The optimal hyper-parameters of the Bid_{100}'s GBM $level_1$ model (with Box-Cox transformation and/or normalization)

Preprocessing	Shrinkage	Depth	Trees	RMSE	R2	RMSE SD	R2 SD
Norm.	0.005	1	7000	7.50759	0.99993	4.69944	0.00009
Trans. & Norm.	0.005	1	7000	7.49348	0.99993	4.68744	0.00009



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.26: Ask_{100}'s GBM $level_1$ model selection with transformation and normalization

Although the Box-Cox transformation of predictors, to resolve the skewness, before normalization has the same performance as applying normalization only, the transformation is applied to the GBM models to inject diversity at the data pre-processing level, opposite to the MARS models, where only normalization is used. Figure 4.26 shows the model selection results for the Ask_{100}'s GBM model. The model's optimal hyper-parameters and its performance are listed in Table 4.9.

Table 4.9: The optimal hyper-parameters of the Ask_{100}'s GBM $level_1$ model (with Box-Cox transformation and/or normalization)

Preprocessing	Shrinkage	Depth	Trees	RMSE	R2	RMSE SD	R2 SD
Trans. & Norm.	0.005	1	10000	9.12861	0.99987	7.97573	0.00025

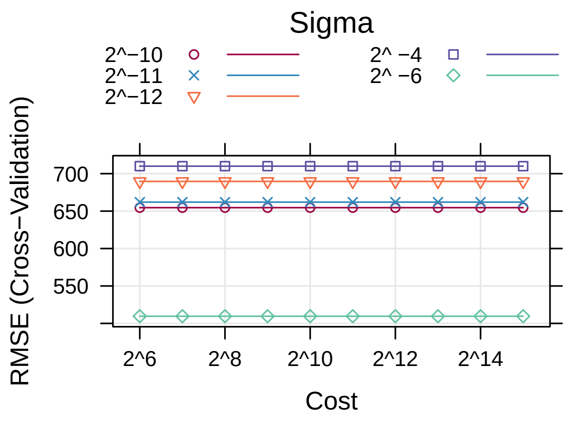
The optimal Ask_{100}/Bid_{100} GBM's hyper-parameters are then used to train 50 Ask/50 Bid GBM models on $level_1$ training dataset, with 182,388 observations.

Model Selection for the SVM Algorithm

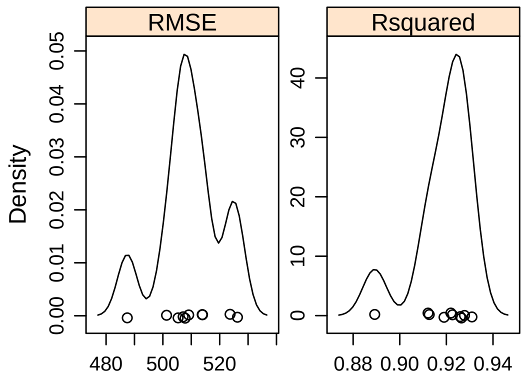
SVMs [91] for regression use a threshold (denoted as ϵ), set by the modeler, to select the data points that contribute to the regression fit (data points with an absolute residual greater than ϵ contribute to the regression fit). The choice of ϵ, as will see in this section, has significant implications on the SVM model's performance and training time, especially for a relatively large number of observations. Other hyper-parameters are defined by the type of kernel selected. For this case study, an SVM with a radial basis function is used, which has two hyper-parameters, σ and $Cost$, that control the complexity of the model.

Similar to the MARS and GBM models, model selection, using 10-fold cross-validation and grid-search, is performed for Bid_{100} price to select the optimal σ and $Cost$, given two different pre-processing steps. This process is repeated for three ϵ values (1, 0.1, and 0.01).

Figure 4.27 to Figure 4.32 show the results of the Bid_{100}'s SVM model selection with predictors transformation and/or normalization for the three ϵ values (1, 0.1, and 0.01), respectively.

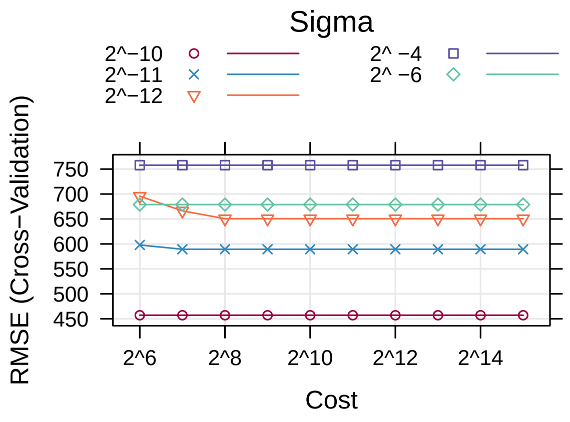


(a) Hyper-parameters grid-search

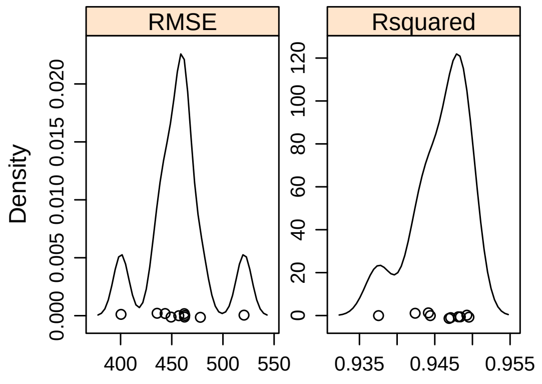


(b) Optimal model resampling accuracy

Figure 4.27: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 1$) with normalization only

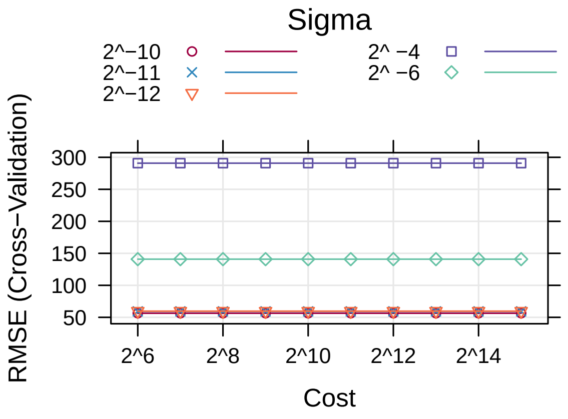


(a) Hyper-parameters grid-search

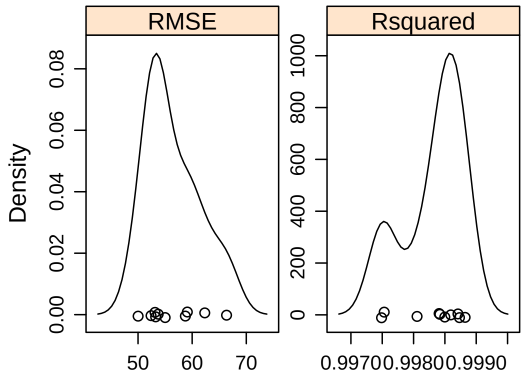


(b) Optimal model resampling accuracy

Figure 4.28: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 1$) with transformation and normalization

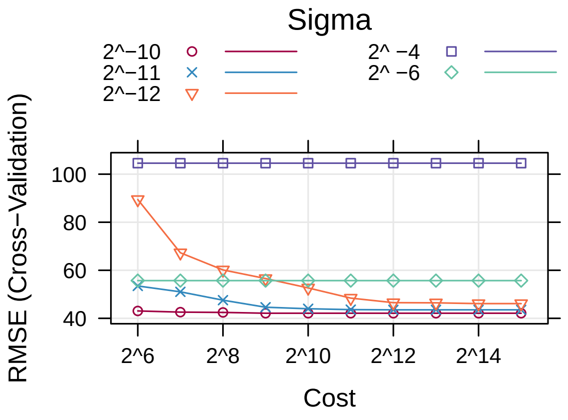


(a) Hyper-parameters grid-search

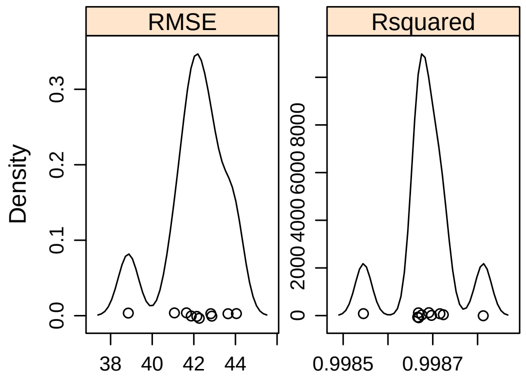


(b) Optimal model resampling accuracy

Figure 4.29: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 0.1$) with normalization only

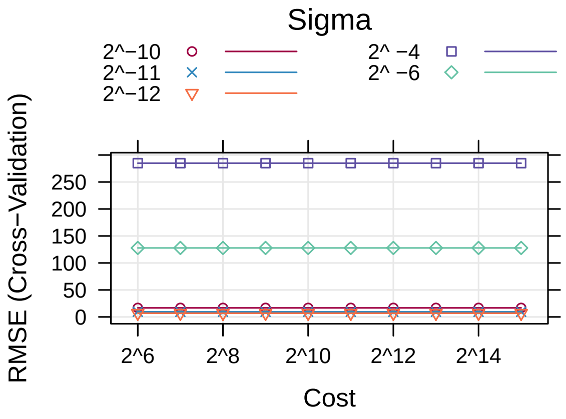


(a) Hyper-parameters grid-search

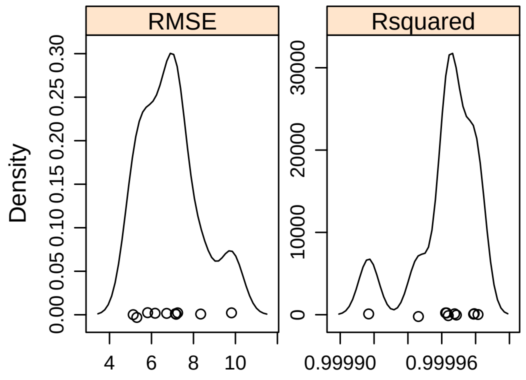


(b) Optimal model resampling accuracy

Figure 4.30: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 0.1$) with transformation and normalization

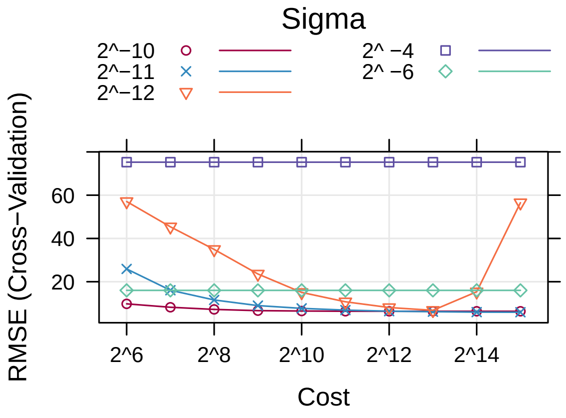


(a) Hyper-parameters grid-search

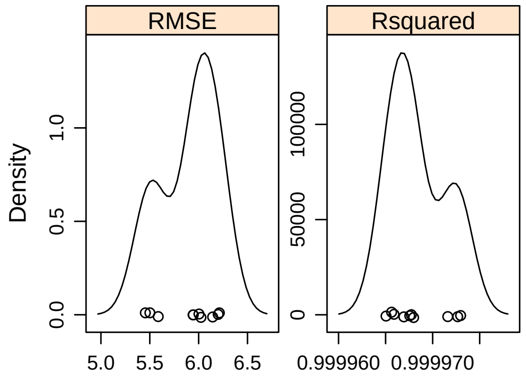


(b) Optimal model resampling accuracy

Figure 4.31: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 0.01$) with normalization only



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.32: Bid_{100}'s SVM $level_1$ model selection ($\epsilon = 0.01$) with transformation and normalization

Table 4.10 presents the optimal model's hyper-parameters and performance for each value of ϵ and pre-processing method. The results indicate that applying Box-Cox transformation to predictors, before normalization, results in a slight performance improvement, but considerably different models. The choice of a smaller ϵ significantly increases the model performance.

Table 4.10: The optimal hyper-parameters of the Bid_{100}'s SVM $level_1$ model (with Box-Cox transformation and/or normalization, and different values of Epsilon (ϵ))

Preprocessing	Epsilon	Sigma	Cost	RMSE	R2	RMSE SD	R2 SD
Norm.	1.0	2^{-6}	2^{6}	509.5908	0.91887	11.04999	0.01210
Trans. & Norm.	1.0	2^{-10}	2^{6}	457.1774	0.94578	30.68183	0.00374
Norm.	0.1	2^{-6}	2^{6}	56.40770	0.99833	5.09217	0.00048
Trans. & Norm.	0.1	2^{-10}	2^{9}	42.12075	0.99869	1.45918	6.6202e-05
Norm.	0.01	2^{-12}	2^{9}	6.88528	0.99996	1.42417	1.9209e-05
Trans. & Norm.	0.01	2^{-11}	2^{15}	5.90125	0.99997	0.28531	2.9439e-06

To investigate the effect of ϵ's value on the model's training time, for a relatively large dataset, the optimal hyper-parameters of the Bid_{100}'s SVM models with $\epsilon = 1$ and $\epsilon = 0.1$ are applied to $level_1$ training dataset, with 182,388 observations. The training is performed on a local PC with 6 cores and 8 GB RAM.

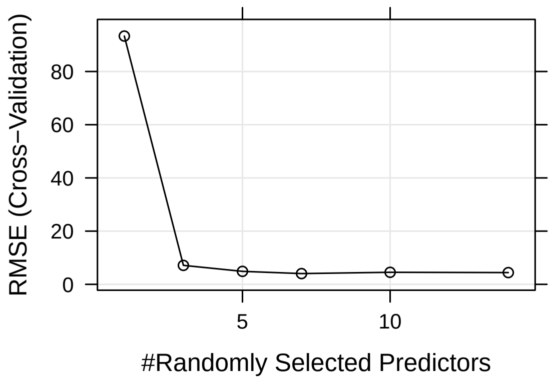
Table 4.11 shows that training an SVM regression model, using a large dataset, takes a very long time. A small reduction in ϵ's value results in an enormous increase in the training time. For the purpose of this research, the computation cost of training 100 SVM Ask/Bid models, just for $level_1$, is considered unacceptable; and therefore, the SVM learning algorithm is abandoned.

Table 4.11: Training time for a SVM model with different ϵ values ($level_1$ dataset with 182,388 observations)

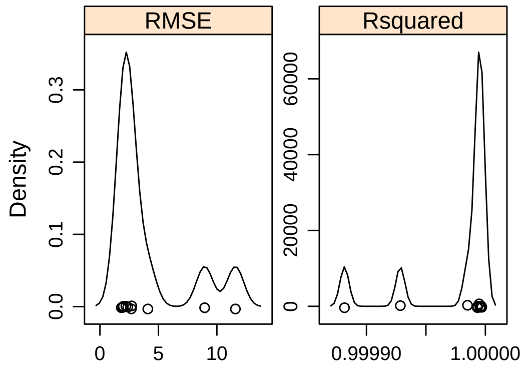
Epsilon	Time per model	Estimated time for all 100 Bid/Ask models
1	8.73 hours	8.73 hours $\times$ 100 $\approx$ 36 days
0.1	3.72 days	3.72 days $\times$ 100 $\approx$ 372 days

Model Selection for the RF Algorithm

Random forest [12] learning algorithm has a major tuning hyper-parameter called m_{try}, which is the number of randomly selected predictors to choose from at each split. For a regression problem, Breiman [12] recommends setting m_{try} to the number of predictors (P) divided by 3. Although RF is relatively insensitive to different values of m_{try} around the recommended value, grid-search and cross-validation are used to select the optimal m_{try} value. A secondary hyper-parameter is the number of trees to grow. Figure 4.33 to Figure 4.35 show the results of the Bid_{100}'s RF model selection with 1000, 3000, and 5000 trees, respectively. Box-Cox transformation and normalization are applied to the predictors.

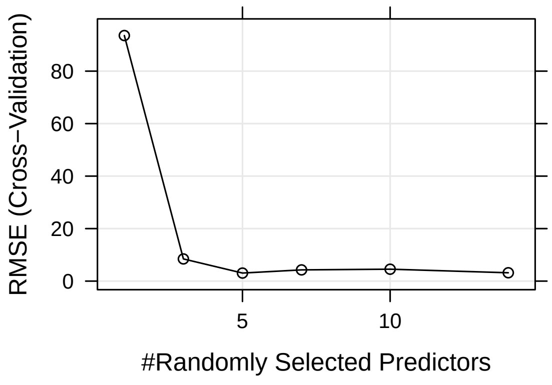


(a) Hyper-parameters grid-search

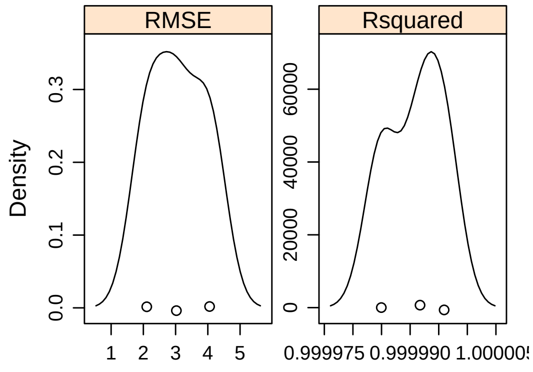


(b) Optimal model resampling accuracy

Figure 4.33: Bid_{100}'s RF (1000 tree) $level_1$ model selection

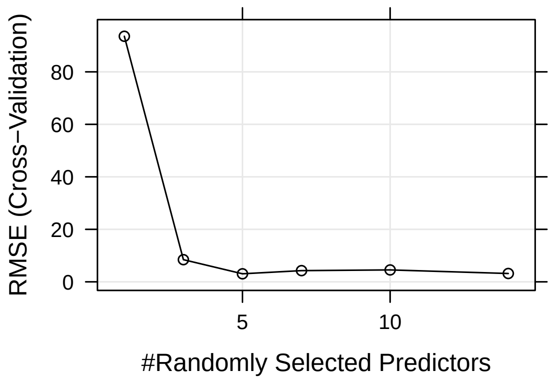


(a) Hyper-parameters grid-search

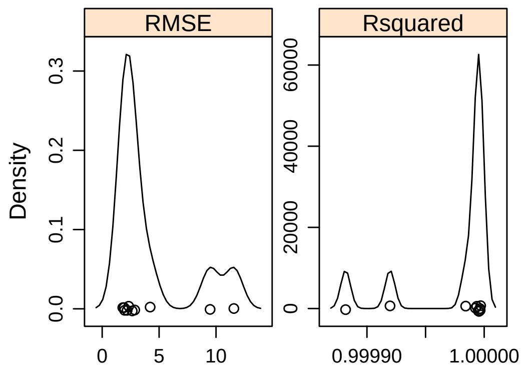


(b) Optimal model resampling accuracy

Figure 4.34: Bid_{100}'s RF (3000 tree) $level_1$ model selection



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.35: Bid_{100}'s RF (5000 tree) $level_1$ model selection

The results of the Bid_{100}'s RF $level_1$ model selection with the three values of number of trees are listed in Table 4.12. Although the number of trees appears to have no significant effect on the RMSE, the RMSE standard divination for the 3000 tree model is lower than the other two models. Therefore, the hyper-parameters $m_{try} = 5$ and $n.trees = 3000$ are selected.

Table 4.12: The optimal hyper-parameters of the Bid_{100}'s RF $level_1$ model (with Box-Cox transformation/normalization and different number of trees)

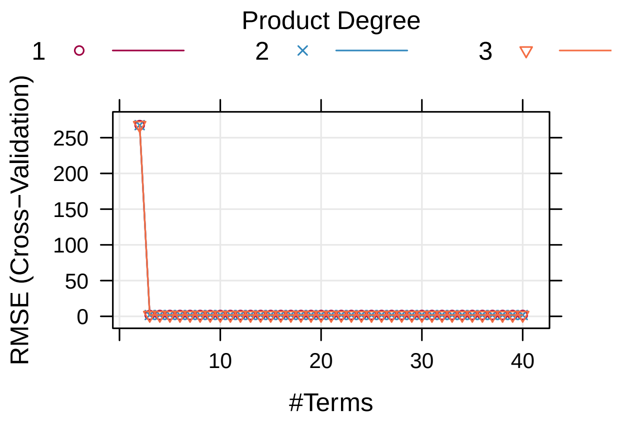
n.trees	mtry	RMSE	R2	RMSE SD	R2 SD	Model size	Time
1000	7	4.03995	0.99998	3.40319	3.9263e-05	0.22 GB	0.5 hours
3000	5	3.06172	0.99999	0.97509	5.5409e-06	0.66 GB	1.12 hours
5000	7	4.09987	0.99998	3.48991	4.0441e-05	1.10 GB	37.73 hours

The table also lists the model size and computation time, on an Amazon EC2 instance with 32 cores and 65 GB, using $level_1$ model selection dataset with 10,991 observations. The selected hyper-parameters are then used to train the one Bid model, on a local PC with 6 cores and 8 GB, using the $level_1$ training dataset with 182,388 observations. However, the available RAM (6 GB) could not handle the required model size with 3000 trees and a large number of observations. The $n.trees$ was reduced to 1000, but the computer crashed again. Although RF models can be trained on the Amazon EC instance, with enough RAM and cores, the estimated high financial costs, due to long training time of 100 Ask/Bid RF models, is considered unacceptable for the purpose of this research; and therefore, the random forest learning algorithm is abandoned.

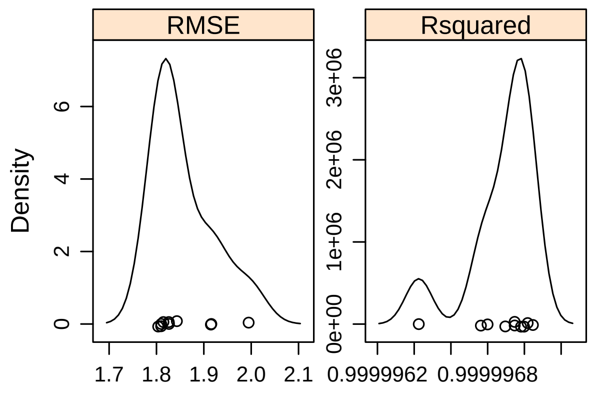
In this section, model selection using grid-search and 10-fold cross-validation was performed for four learning algorithms (MARS, GBM, SVM, and RF). Out of the four, only MARS and GBM algorithms are selected to implement the multi-model strategy, and train Ask/Bid post-liquidity shock's models using $level_1$ training dataset. The trained models will be utilized in the next section to implement $level_2$ cascading-model strategies.

4.6.3 Cascading-Model Strategies

As explained in Section 4.4.3, the two cascading-model strategies use the predictions of $level_1$ models to train new post-liquidity shock's 50 Ask/50 Bid prices for $level_2$ dataset, with 121,533 observations. Similar to $level_1$ models, the model selection process is reapplied to the new features of the Bid_{100}/Ask_{100} prices using $level_2$ model selection dataset, with 10,016 observations.

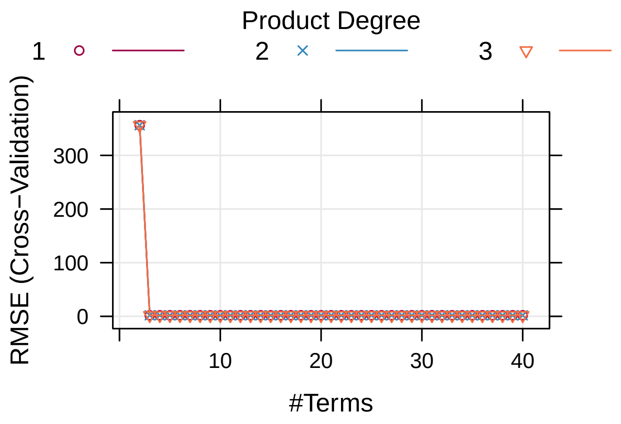


(a) Hyper-parameters grid-search

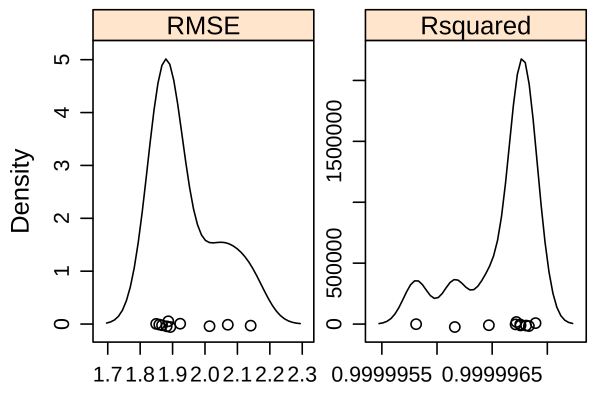


(b) Optimal model resampling accuracy

Figure 4.36: Bid_{100}'s MARS $level_2$ model selection (cascading strategy 1)



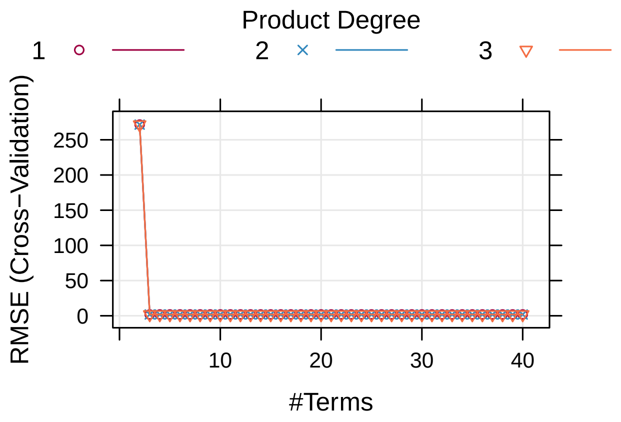
(a) Hyper-parameters grid-search



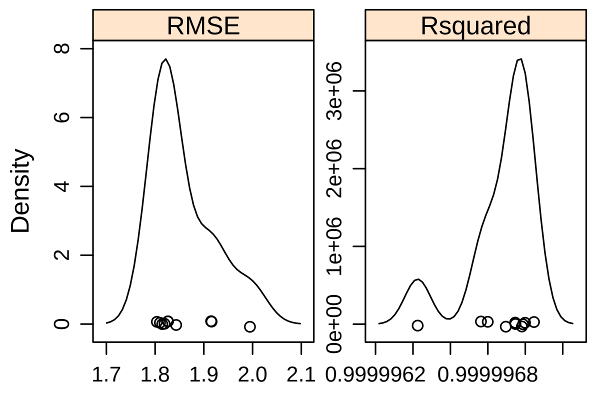
(b) Optimal model resampling accuracy

Figure 4.37: Ask_{100}'s MARS $level_2$ model selection (cascading strategy 1)

The cascading-model strategy 1 trains a new model for each post-liquidity shock's Ask/Bid price on an expanding features set by chaining the predicted post-liquidity shock's prices Bid_t/Ask_t, $t \in [51, 99]$ with the selected features from $level_2$ dataset. So, the number of features for Bid_{100}/Ask_{100} models are 93/83. The model selection results for Bid_{100}/Ask_{100} are shown in Figure 4.36 and Figure 4.37, respectively.

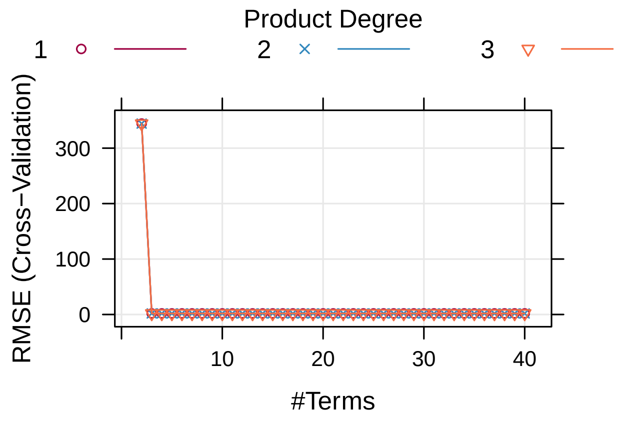


(a) Hyper-parameters grid-search

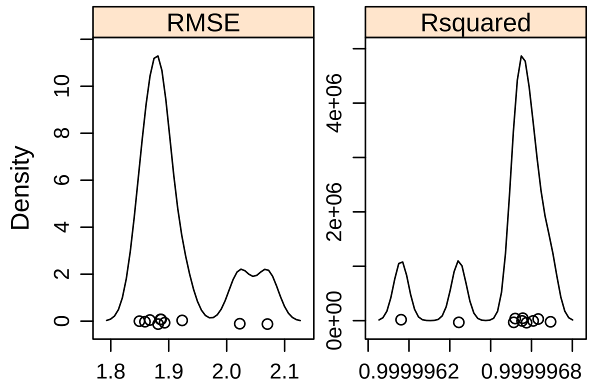


(b) Optimal model resampling accuracy

Figure 4.38: Bid_{100}'s MARS $level_2$ model selection (cascading strategy 2)



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

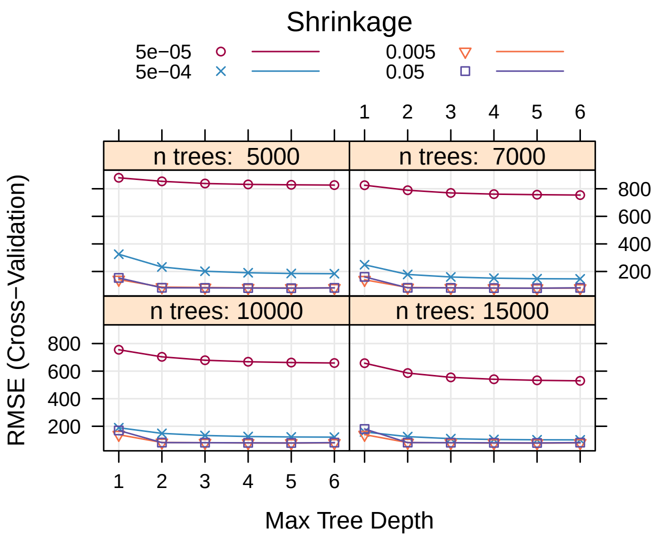
Figure 4.39: Ask_{100}'s MARS $level_2$ model selection (cascading strategy 2)

The cascading-model strategy 2 uses the same approach but the expanding features set is formed by chaining the predicted post-liquidity shock's prices with only the last three prices just before the liquidity shock. So, the number of features for Bid_{100}/Ask_{100} models is the same (52). Strategy 2 model selection results for Bid_{100}/Ask_{100} are shown in Figure 4.38 and Figure 4.39, respectively. Table 4.13 presents the cascading-model strategies' optimal hyper-parameters and results for the Bid_{100}/Ask_{100} MARS $level_2$ models. Both cascading-model strategies appear to have almost the same performance.

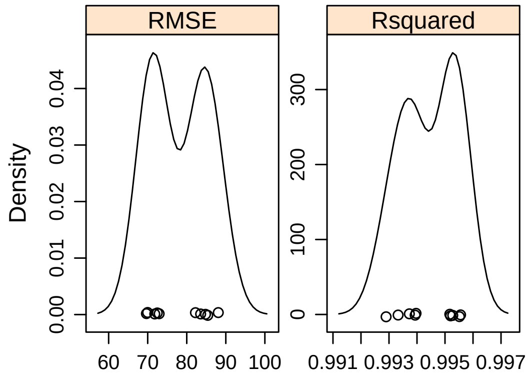
Table 4.13: The optimal hyper-parameters of the Bid_{100}/Ask_{100} MARS $level_2$ models (cascading strategy 1 and 2)

Model	#Terms	degree	RMSE	R2	RMSE SD	R2 SD
Bid_{100} (str. 1)	3	1	1.85590	0.99999	0.06404	1.8475e-07
Bid_{100} (str. 2)	3	1	1.85688	0.99999	0.06335	1.8249e-07
Ask_{100} (str. 1)	3	1	1.93861	0.99999	0.10085	3.4815e-07
Ask_{100} (str. 2)	3	1	1.91392	0.99999	0.07355	2.1929e-07

The same Bid_{100}/Ask_{100} model selection processes are repeated for the GBM learning algorithm. Figure 4.40 and Figure 4.41 show the results for cascading-model strategy 1, while the results for strategy 2 are shown in Figure 4.42 and Figure 4.43, respectively.

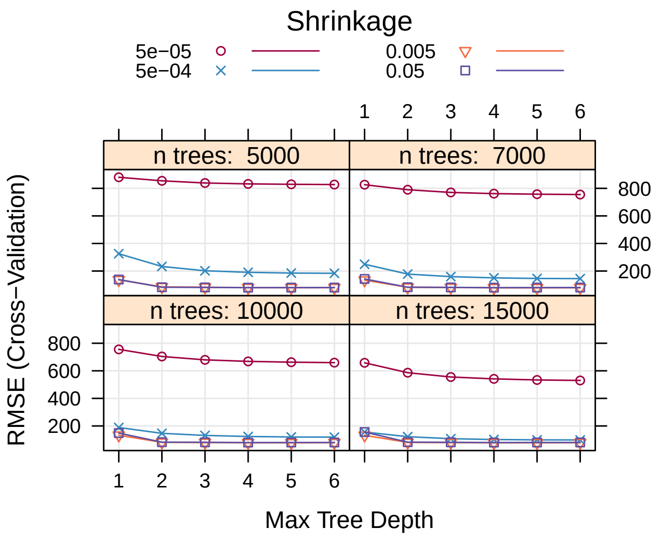


(a) Hyper-parameters grid-search

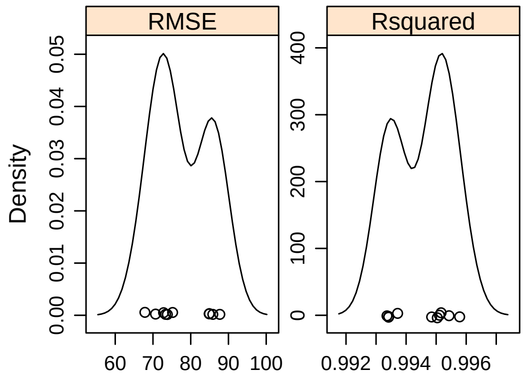


(b) Optimal model resampling accuracy

Figure 4.40: Bid_{100}'s GBM $level_2$ model selection (cascading strategy 1)

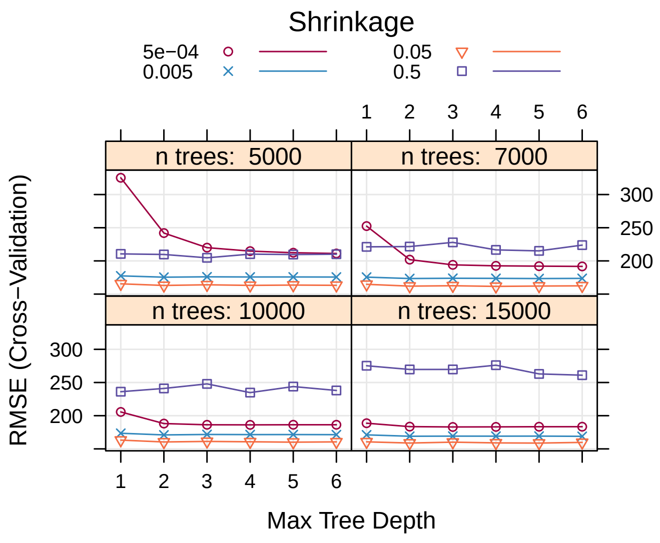


(a) Hyper-parameters grid-search

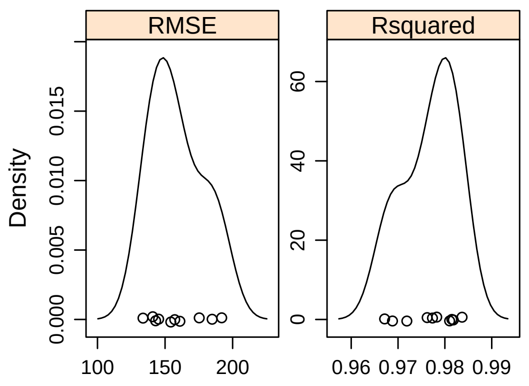


(b) Optimal model resampling accuracy

Figure 4.41: Ask_{100}'s GBM $level_2$ model selection (cascading strategy 1)

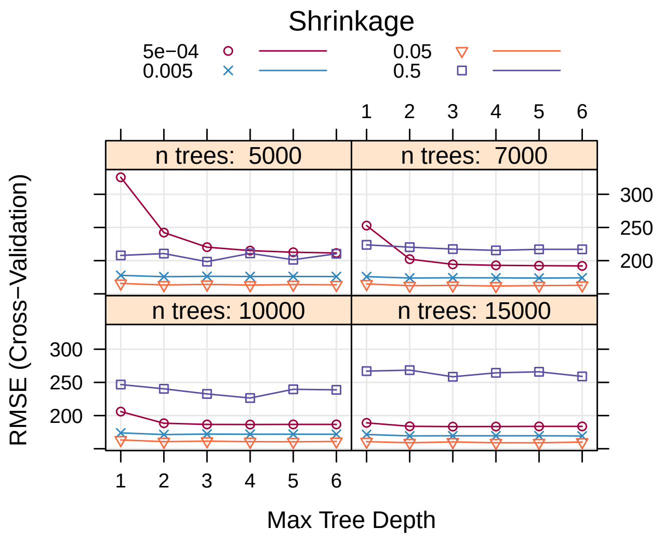


(a) Hyper-parameters grid-search

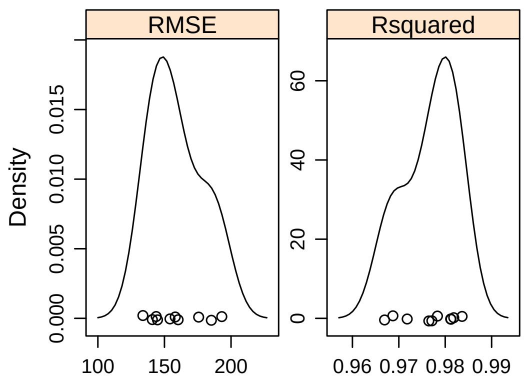


(b) Optimal model resampling accuracy

Figure 4.42: Bid_{100}'s GBM $level_2$ model selection (cascading strategy 2)



(a) Hyper-parameters grid-search



(b) Optimal model resampling accuracy

Figure 4.43: Ask_{100}'s GBM $level_2$ model selection (cascading strategy 2)

Table 4.14 lists the cascading-model strategies' optimal hyper-parameters and results for the Bid_{100}/Ask_{100} GBM $level_2$ models. Strategy 1 appears to have significantly higher performance than strategy 2.

Table 4.14: The optimal hyper-parameters of the Bid_{100}/Ask_{100} GBM $level_2$ models (cascading strategy 1 and 2)

Model	Shrinkage	Depth	Trees	RMSE	R2	RMSE SD	R2 SD
Bid_{100} (str. 1)	5e-03	5	10000	78.49496	0.99440	6.81351	0.00093
Bid_{100} (str. 2)	5e-02	5	10000	160.1098	0.97637	19.0325	0.00558
Ask_{100} (str. 1)	5e-03	5	10000	77.70901	0.99453	7.31307	0.00094
Ask_{100} (str. 2)	5e-02	5	10000	160.4779	0.97630	19.2094	0.00563

The optimal Ask_{100}/Bid_{100} hyper-parameters for MARS and GBM are then used to train Ask/Bid post-liquidity shock's models on the $level_2$ training dataset, with 121,533 observations. The training is performed on a local PC with 6 cores and 8 GB RAM.

In this section, model selection and training were conducted for single-model, multi-model, and cascading-model strategies. To inject model diversity at the data, features, and learning algorithm levels, a total of 602 NNET, MARS, and GBM post-liquidity shock's models were trained using different datasets and features. The performance of these models is evaluated in the next section.

4.7 Performance Evaluation

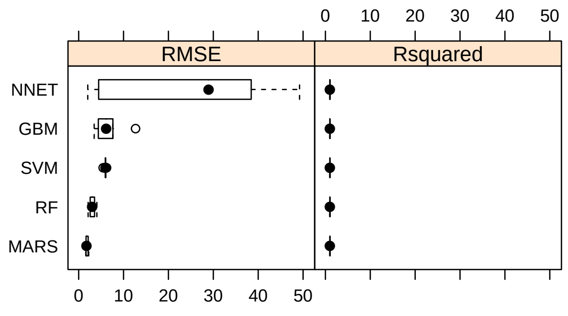
The most common metric to asses the predictive capabilities of a regression model is the Root Mean Square Error (RMSE). The value of the RMSE is interpreted as either the average distance between the observed values and the model predictions or as how far, on average, the residuals (errors) are from zero. For this case study, the unit of the RMSE is in pence (1 £=100 pence). Another common metric for regression models is the coefficient of determination, commonly referred to as R^2. Its value can be interpreted as the proportion of the variance, in the test dataset, that is explained by the model [45].

In this section, the performance of the single-model, multi-model, and cascading-model strategies is evaluated using 10-fold cross-validation (out-of-fold) and a testing dataset (with 30,327 observations). Although the 10-fold CV was only used for Ask_{100}/Bid_{100} model selection with a relatively small number of observations ($level_1$ 10,991 and $level_2$ 10,016), it can be used to compare the performance of the five learning algorithms (NNET, MARS, GBM, SVM, and RF).

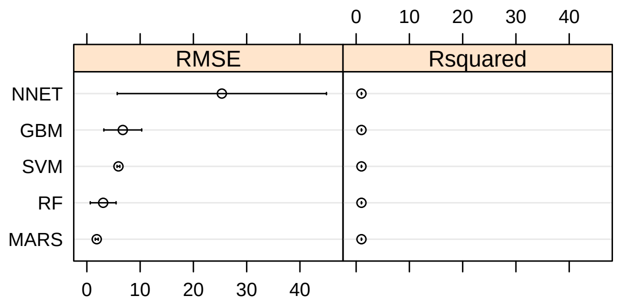
4.7.1 Performance Evaluation Using the 10-fold CV Resampling

To make a fair comparison between the learning algorithms, the same order of the ten folds, with exactly the same observations, was used during model selection for all algorithms in Section 4.6. This section makes use of the model selection 10-fold CV resampling to compare the performance of the optimal models for $level_1$ and $level_2$ models. For each Ask_{100}/Bid_{100}, two plots are used to visualize the relative performance of the algorithms. The first plot is the relative performance distributions, using box-and-whisker plots. The second plot is the relative performance estimates with 95% Confidence Level (CL).

$Level_1$ model selection involved tuning the hyper-parameters of the Bid_{100}/Ask_{100} NNET, MARS, GBM, SVM, and RF learning algorithms. Figure 4.44a shows $level_1$'s Bid_{100} models performance distributions, while Figure 4.44b shows their performance estimates with 95% CL. The order of the algorithms based on the best performance (minimum RMSE) is as follows: MARS, RF, SVM, GBM, and NNET. MARS and SVM algorithms have very low-performance uncertainty (standard deviation), while RF and GBM algorithms have higher uncertainty. The neural network has the highest performance uncertainty. On the other hand, all five learning algorithms have almost similar R^2 values. Since SVM and RF algorithms were abandoned, Figure 4.45 shows the relative resampling performance of only the Ask_{100}'s NNET, MARS, and GBM models. Similarly, the MARS algorithm outperforms the GBM and NNET algorithms.

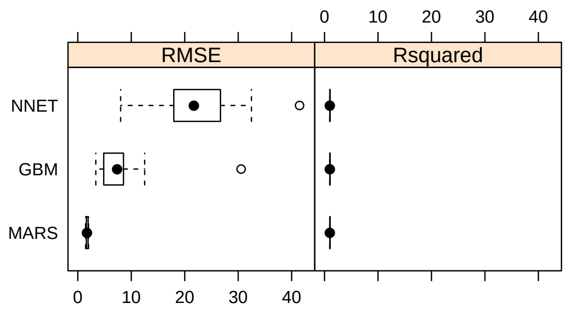


(a) Performance distributions

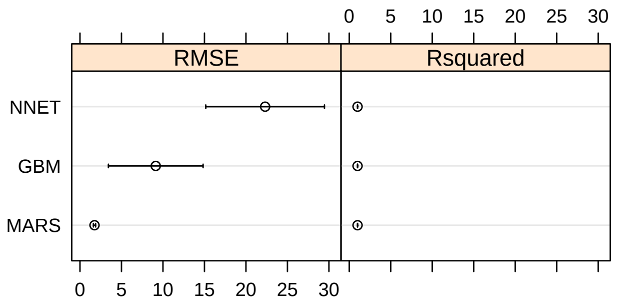


(b) Estimates with 95% CL

Figure 4.44: $Level_1$ Bid_{100} models resampling performance



(a) Performance's distributions

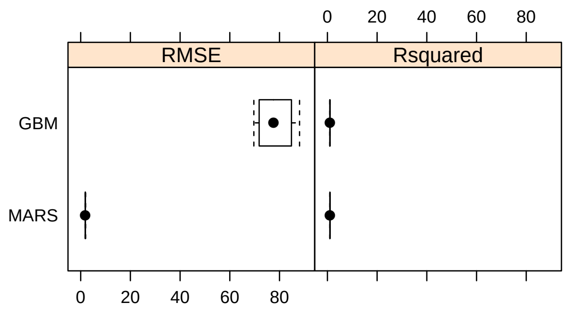


(b) Estimates with 95% CL

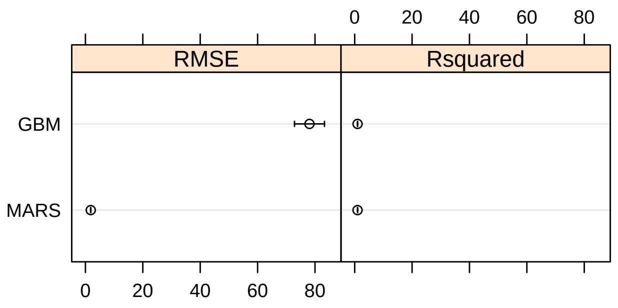
Figure 4.45: $Level_1$ Ask_{100} models resampling performance

In Section 4.6.3, the hyper-parameters of the Bid_{100}/Ask_{100} MARS and GBM models were tuned using $level_2$ dataset and two cascading-model strategies. Figure 4.46 and Figure 4.47 show the resampling performance of $level_2$ optimal Bid_{100}/Ask_{100} (strategy 1) models, respectively. The performance of $level_2$ MARS algorithm is consistent with its $level_1$ performance. However, the performance of the GBM algorithm is worse than its $level_1$ performance. This could be due to different patterns in $level_2$ dataset that the GBM model could not fully capture.

The same trend follows in $level_2$ Bid_{100}/Ask_{100} (strategy 2) MARS/GBM resampling performance, shown in Figure 4.48 and Figure 4.49, respectively. Comparing the cross-validation performance of models clearly suggests, even before using fully trained models and a test dataset, that the MARS models will outperform the NNET and GBM models. In the next section, a test dataset is used to evaluate the performance of fully trained (on large datasets) models.

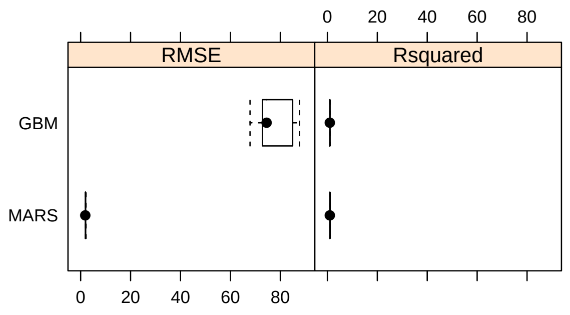


(a) Performance's distributions

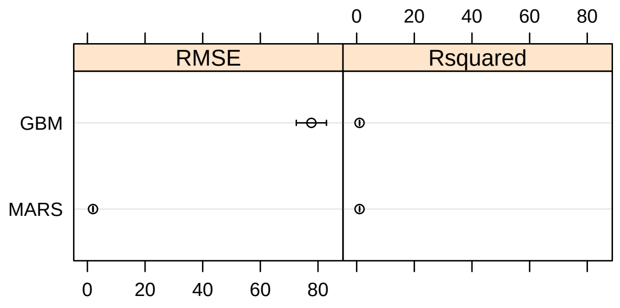


(b) Estimates with 95% CL

Figure 4.46: $Level_2$ Bid_{100} (strategy 1) models resampling performance

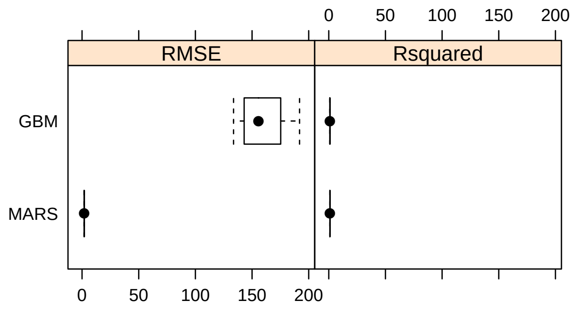


(a) Performance's distributions

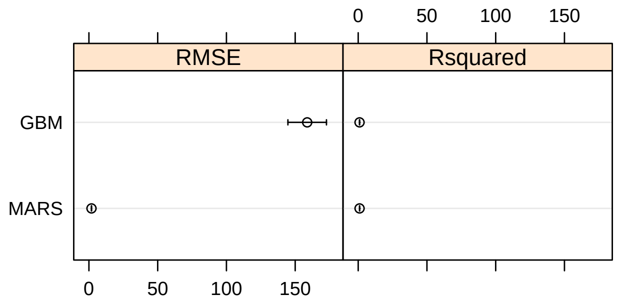


(b) Estimates with 95% CL

Figure 4.47: $Level_2$ Ask_{100} (strategy 1) models resampling performance

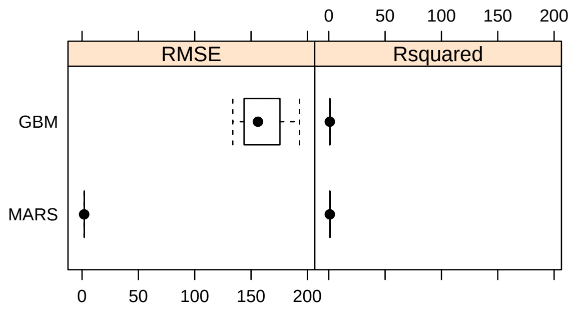


(a) Performance's distributions

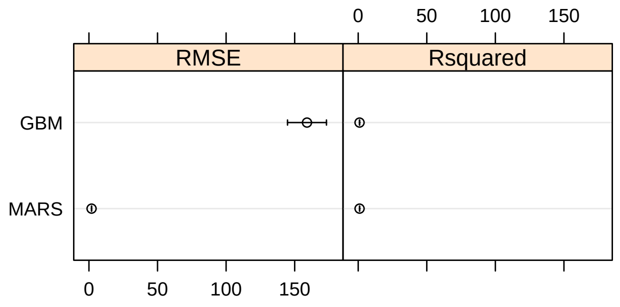


(b) Estimates with 95% CL

Figure 4.48: $Level_2$ Bid_{100} (strategy 2) models resampling performance



(a) Performance's distributions



(b) Estimates with 95% CL

Figure 4.49: $Level_2$ Ask_{100} (strategy 2) models resampling performance

4.7.2 Performance Evaluation Using the Testing Dataset

In this section, the performance of the fully trained Bid/Ask NNET, MARS, and GBM models is evaluated using a testing dataset, with 30,327 observations. The RMSE of the post-liquidity shock's 50 Bid/50 Ask prices are visualized. To better understand the performance of a regression model, the predicted vs. observed and residuals plots are used. Given the large number of outcomes (50 Bid and 50 Ask prices), the two plots are shown only for the Bid_{100}/Ask_{100} prices.

It should be noted that the interpretation of the model's RMSE dependence on the outcome variance. For example, Table 4.15 shows the statistics of the Bid_{100}/Ask_{100} prices. Given this huge variance, a small RSME, for example less than the minimum price, would be considered a very good performance. As will be shown in this section, our hypothesis, in Section 4.4.4, that a market-based approach will be able to handle the huge variance, and reasonably capture the dynamics of the stock market.

Table 4.15: Statistics of the Bid_{100}/Ask_{100} prices (pence)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Bid_{100}	20.82	326.40	707.00	1134.00	1866.00	7530.00
Ask_{100}	20.85	326.60	707.50	1135.00	1868.00	7540.00

Performance of the NNET Models

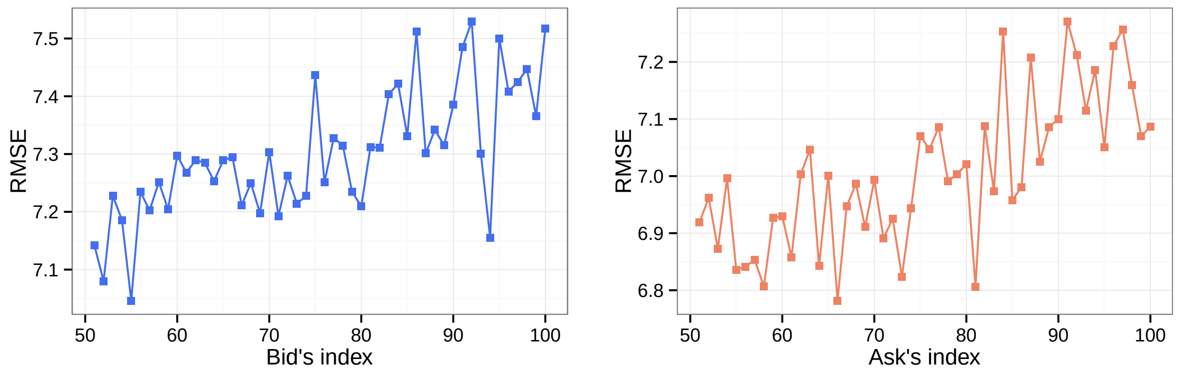
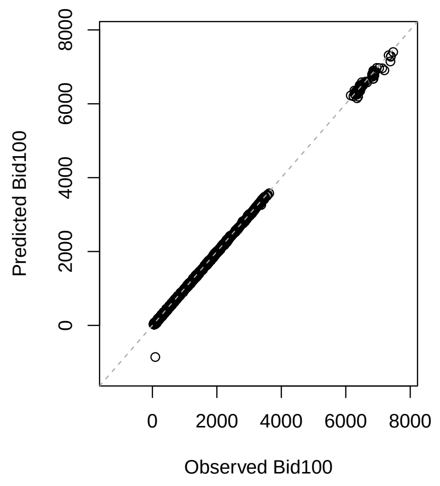
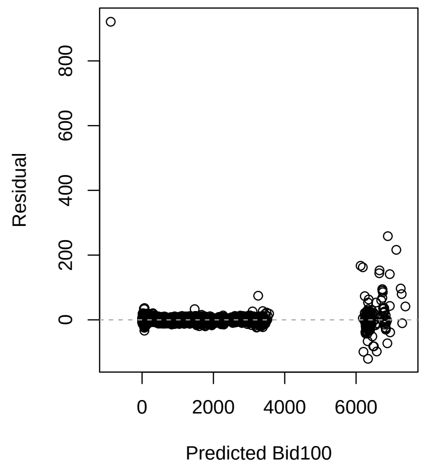


Figure 4.50: RMSE for the NNET $level_1$ models (test dataset, $N = 30,327$)

The two NNET models predict all post-liquidity shock's Bid/Ask prices at once. Figure 4.50 shows the RMSE for the NNET $level_1$ post-liquidity 50 Bid/50 Ask prices. Although the model selection for the single-model strategy, implemented by the NNET, was performed only for Bid_{100}/Ask_{100}, the performance of the NNET models is considered very reasonable.

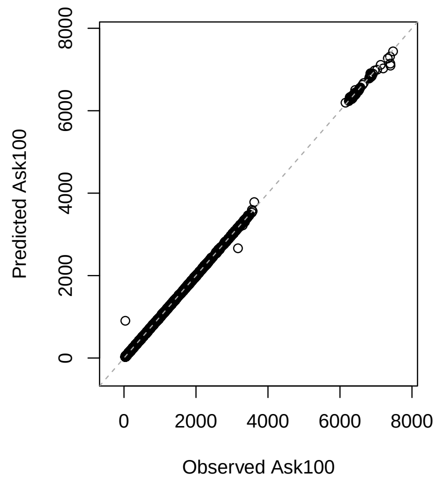


(a) Bid_{100}'s predicted vs observed

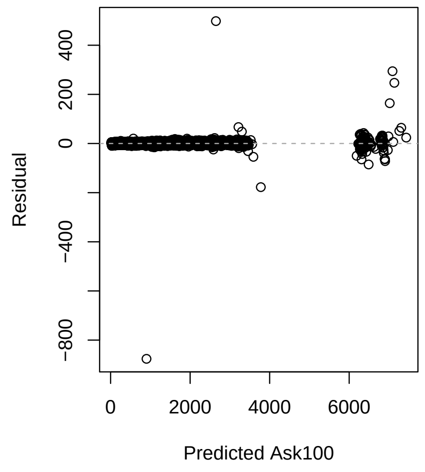


(b) Bid_{100}'s residual

Figure 4.51: Performance of the NNET $level_1$ Bid_{100}



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.52: Performance of the NNET $level_1$ Ask_{100}

The predicted vs. observed values plot for NNET Bid_{100} is shown in Figure 4.51a. For a model with good performance, the data points should align on the diagonal line. Given the wide range of values (20.82 to 7530.00 pence), the plot is hard to interpret. A better plot to zoom in on the performance of the model is the residual plot, shown in Figure 4.51b. The same two plots are shown for the NNET Ask_{100} in Figure 4.52. The figures show that the largest residuals occur for securities/stocks with Bid/Ask prices higher than 4000 (pence). This systematic pattern will be investigated at the end of this section. In addition, few outliers (large residuals) exist for other securities. Table 4.16 presents the RMSE statistics for the NNET $level_1$ Bids/Asks models. The total RMSE for all post-liquidity shock's 50 Bid/50 Ask prices is 364.951 and 350.228 pence, receptively.

Table 4.16: RMSE statistics for the NNET $level_1$ Bids/Asks models ($N = 30,327$)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids RMSE	7.046	7.228	7.292	7.299	7.360	7.529	364.951
Asks RMSE	6.782	6.920	6.995	7.005	7.086	7.271	350.228

Performance of the MARS Models

The MARS learning algorithm was used to implement the $level_1$ multi-model strategy, and $level_2$ cascading-model strategies (1 and 2). Similarly, this section presents the performance of the MARS models for each predictive modeling strategy.

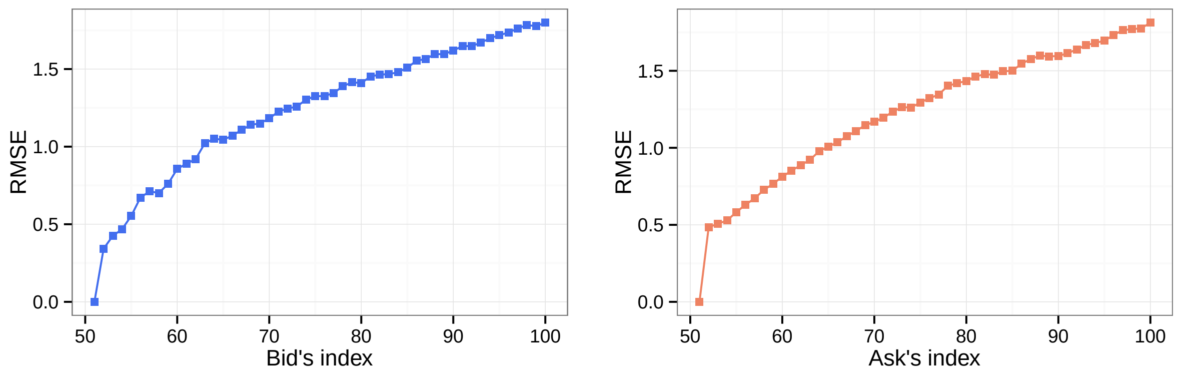
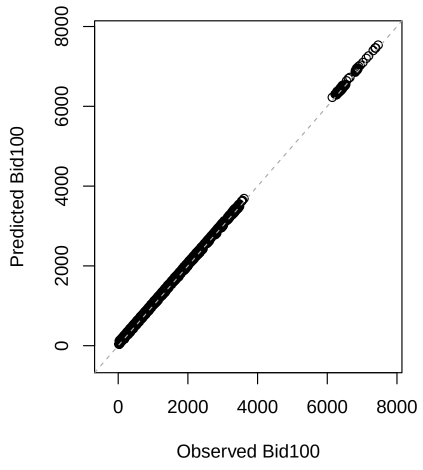
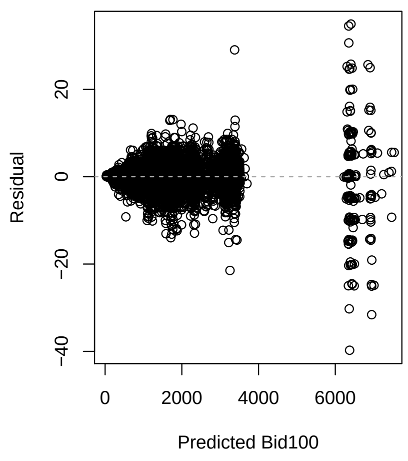


Figure 4.53: RMSE for the MARS $level_1$ models (test dataset, $N = 30,327$)

Figure 4.53 shows the RMSE for the MARS $level_1$ Bid/Ask models. Given the high variance of Bid/Ask prices, the performance of the MARS models is considered very high. For example, for the post-liquidity Bid_{100} price, the range of values is from 20.82 to 7530.00 (pence). The Bid_{100}'s RMSE is 1.799 pence, which is interpreted as on average, the predicted value is ± 1.799 (pence) from the observed value.

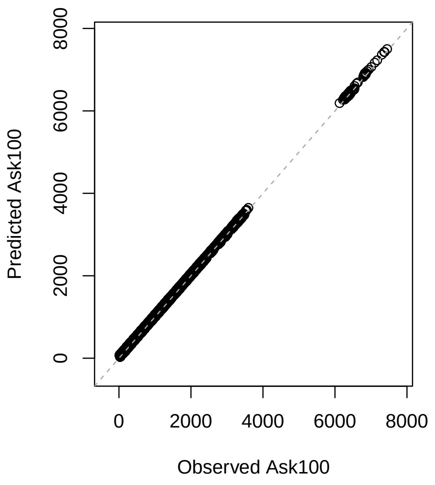


(a) Bid_{100}'s predicted vs observed

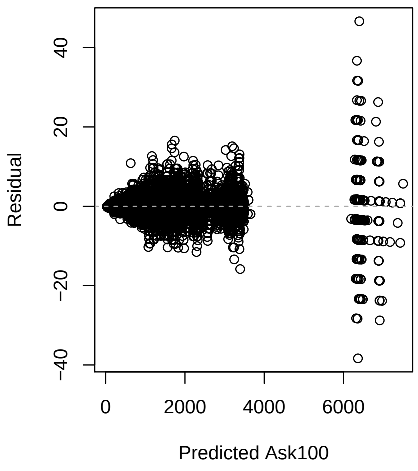


(b) Bid_{100}'s residual

Figure 4.54: Performance of the MARS $level_1$ Bid_{100}



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.55: Performance of the MARS $level_1$ Ask_{100}

Figure 4.54 and Figure 4.55 show a closer look at the performance of the MARS $level_1$ Bid_{100}/Ask_{100}, respectively. Similar to the NNET residuals, the MARS $level_1$ residuals appear to worsen for securities/stocks with Bid/Ask prices higher than 4000 (pence). However, the MARS residuals for these securities is lower than the NNET residuals.

The RMSE for the MARS $level_2$ (cascading strategy 1) Bid/Ask models are shown in Figure 4.56. The predicted vs observed and residuals plots for MARS $level_2$ Bid_{100}/Ask_{100} models are shown in Figure 4.57 and Figure 4.58, respectively. The figures show similar, to MARS $level_1$, performance and residuals patterns.

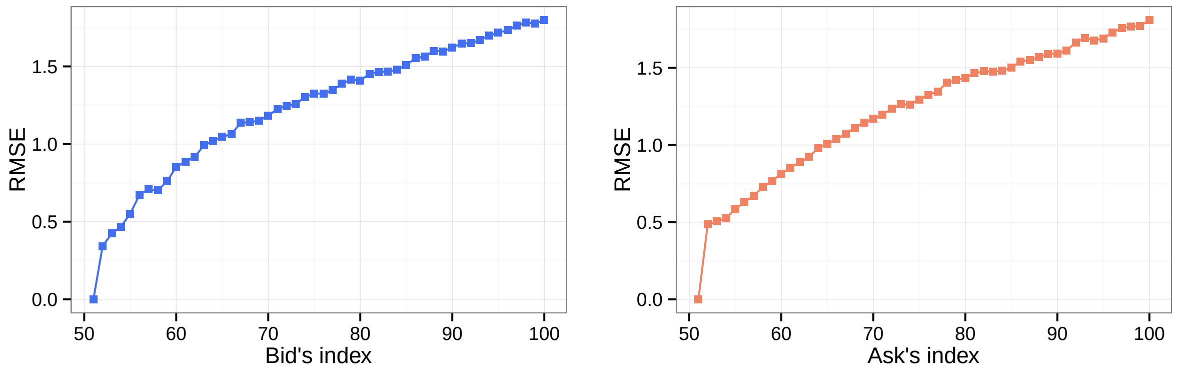
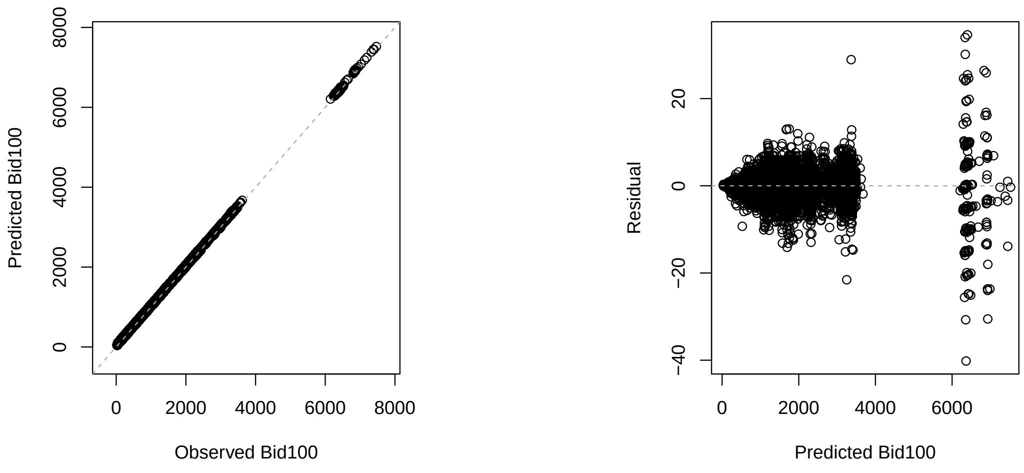
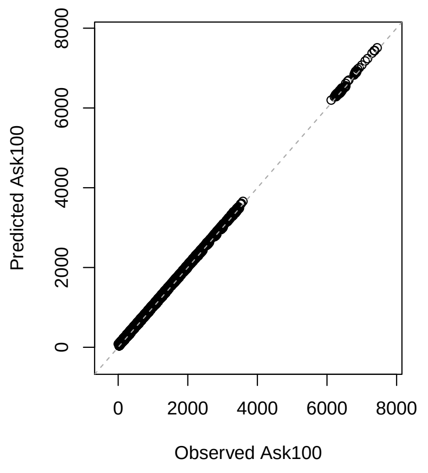


Figure 4.56: RMSE for the MARS $level_2$ (cascading strategy 1) models (test, $N = 30,327$)

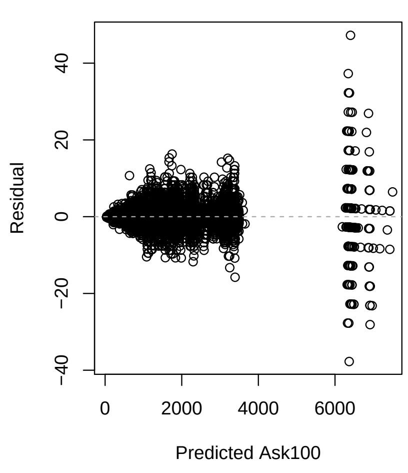


(a) Bid_{100}'s predicted vs observed
(b) Bid_{100}'s residual

Figure 4.57: Performance of the MARS $level_2$ Bid_{100} (cascading strategy 1)



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.58: Performance of the MARS $level_2$ Ask_{100} (cascading strategy 1)

Finally, the RMSE for the MARS $level_2$ (cascading strategy 2) Bid/Ask models are shown in Figure 4.59. Figure 4.60 and Figure 4.61 show detailed performance overviews for the MARS $level_2$ Bid_{100}/Ask_{100} models, respectively.

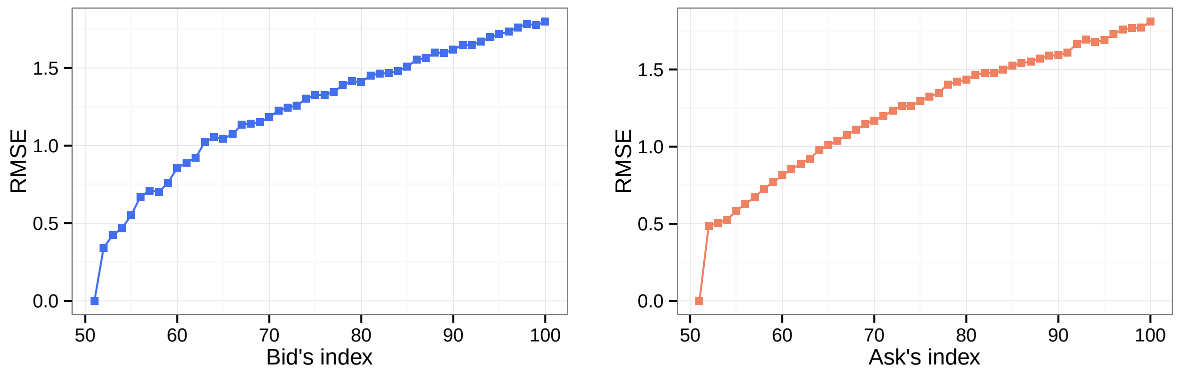
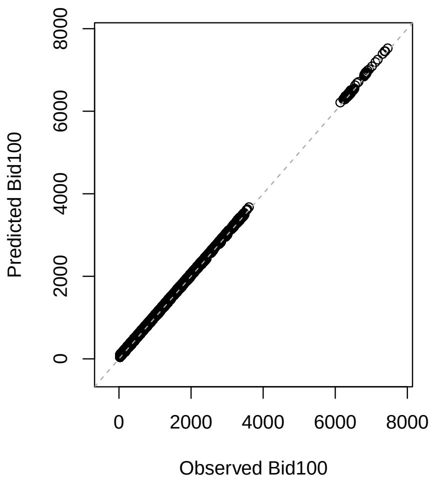
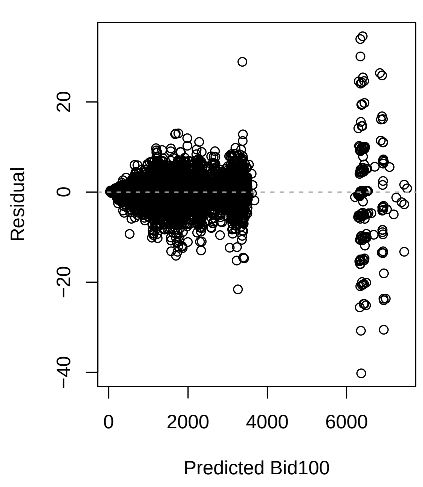


Figure 4.59: RMSE for the MARS $level_2$ models (cascading strategy 2)(test, $N = 30,327$)

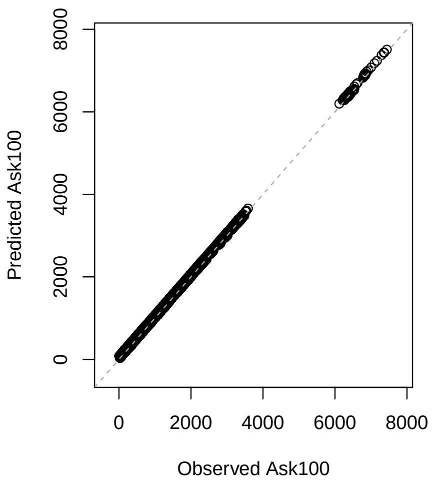


(a) Bid_{100}'s predicted vs observed

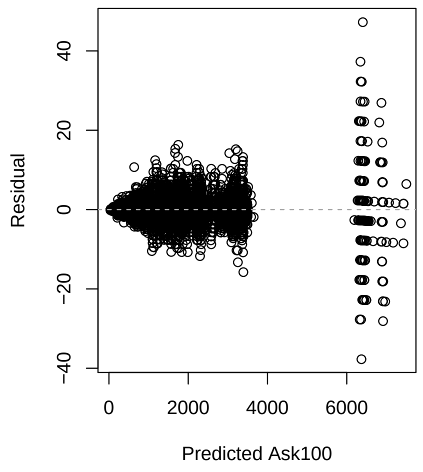


(b) Bid_{100}'s residual

Figure 4.60: Performance of the MARS $level_2$ Bid_{100} (cascading strategy 2)



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.61: Performance of the MARS $level_2$ Ask_{100} (cascading strategy 2)

As presented in Table 4.17, the MARS models for multi-model and cascading-model strategies have very high and consistent performance, as predicted by the 10-fold CV in Section 4.7.1. The fusion of $level_1$ models' predictions with $level_2$ models (cascading strategy 1) appear to slightly improve the performance. On the other hand, $level_2$ models (cascading strategy 2) and $level_1$ models have almost identical performance. The total RMSE for all post-liquidity shock's 50 Bid/50 Ask MARS models, for the three modeling strategies, are significantly lower than the RMSE for the single-model strategy, implemented by the feedforward neural network, Table 4.16.

Table 4.17: RMSE statistics for the MARS 50 Bids/50 Ask models

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	0.000	1.028	1.326	1.237	1.587	1.799	61.857
Bids $level_2$ (str. 1)	0.000	0.999	1.325	1.236	1.589	1.798	61.795
Bids $level_2$ (str. 2)	0.000	1.029	1.325	1.238	1.587	1.798	61.877
Asks $level_1$	0.000	0.936	1.308	1.231	1.589	1.813	61.537
Asks $level_2$ (str. 1)	0.000	0.937	1.310	1.230	1.564	1.809	61.494
Asks $level_2$ (str. 2)	0.000	0.937	1.310	1.231	1.564	1.809	61.537

Performance of the GBM Models

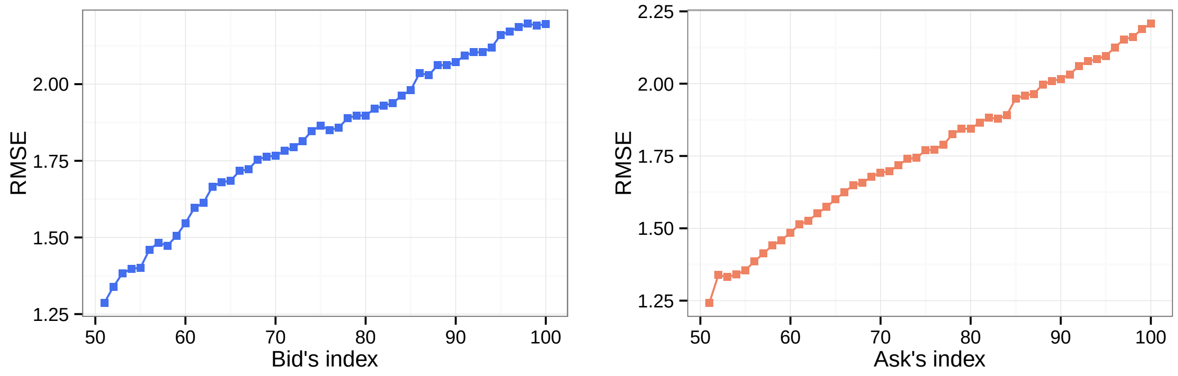
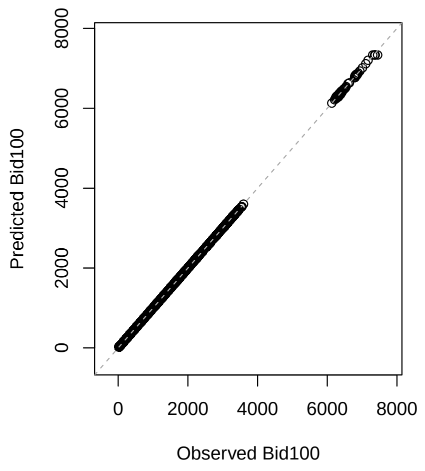


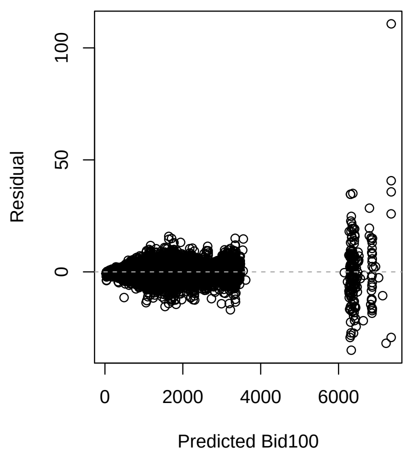
Figure 4.62: RMSE for the GBM $level_1$ models (test dataset, $N = 30,327$)

To add model diversity at the learning algorithm level, the GBM learning algorithm, in addition to the MARS algorithm, was used to implement $level_1$ multi-model strategy, and $level_2$ cascading-model strategies (1 and 2). In this section, the performance of the GBM models is evaluated using the test dataset.

Figure 4.62 show the RMSE for the GBM $level_1$ multi-model strategy. Although the performance is slightly lower than the performance of the MARS $level_1$ models, it is higher than the performance of the single-model strategy, implemented by the NNET

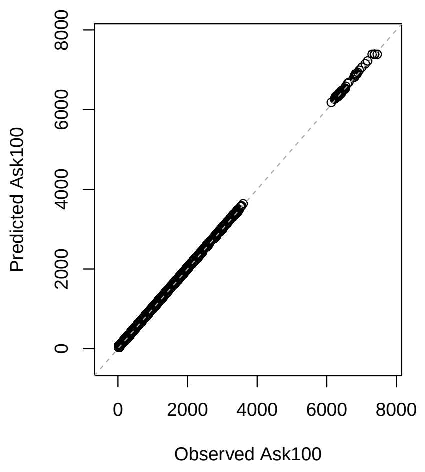


(a) Bid_{100}'s predicted vs observed

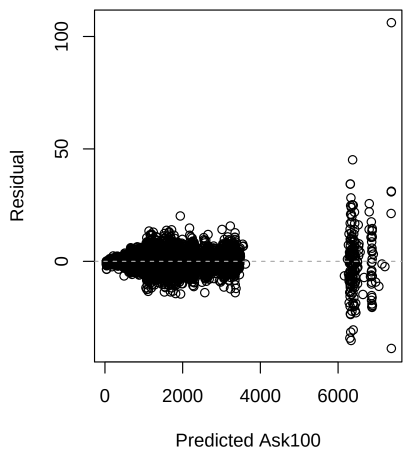


(b) Bid_{100}'s residual

Figure 4.63: Performance of the GBM $level_1$ Bid_{100}



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.64: Performance of the GBM $level_1$ Ask_{100}

To zoom in on the performance of the GBM regression models, predicted vs. observed and residuals plots can be used. Figure 4.63 and Figure 4.64 show these plots for the GBM $level_1$ Bid_{100}/Ask_{100} models, respectively. The figures show repeated residuals patterns for securities with price higher than 4000 (pence), where the difference between the predicted and observed prices is widened.

The Bid/Ask RMSE for GBM $level_2$ cascading-model strategy 1 are shown in Figure 4.65. It seems that feeding the predictions of $level_1$ models to $level_2$ models produced slightly lower performance (higher RMSE).

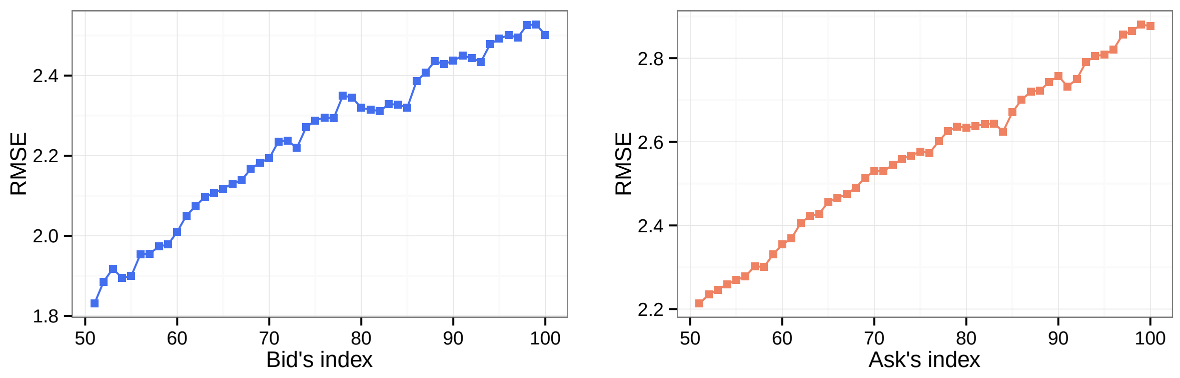
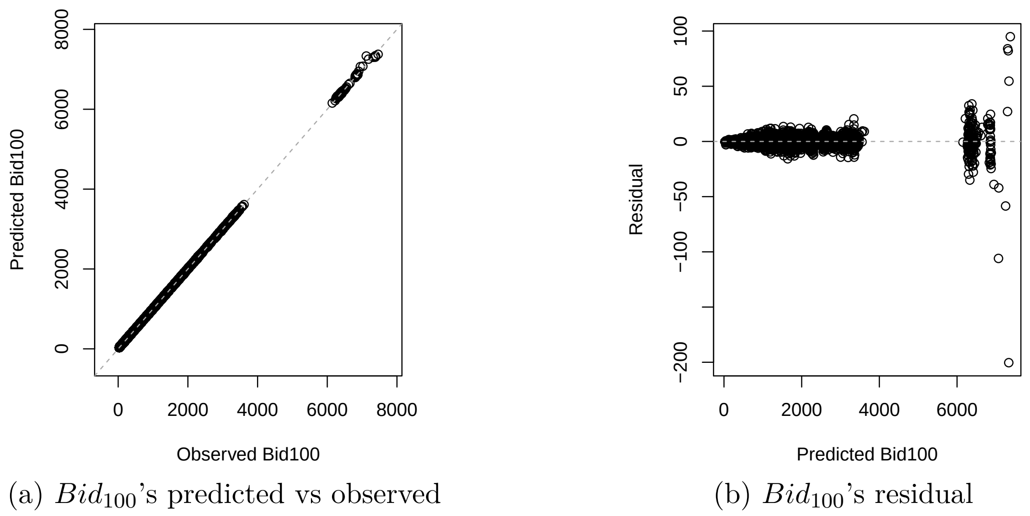


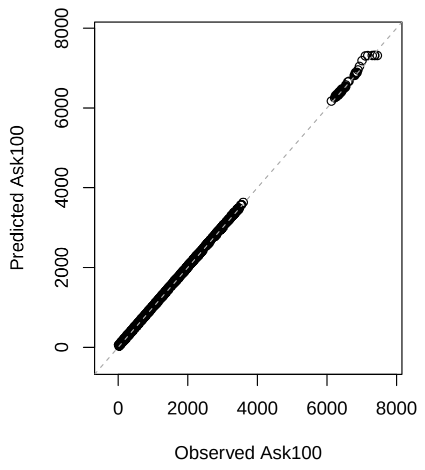
Figure 4.65: RMSE for the GBM $level_2$ (cascading strategy 1) models (test, $N = 30,327$)



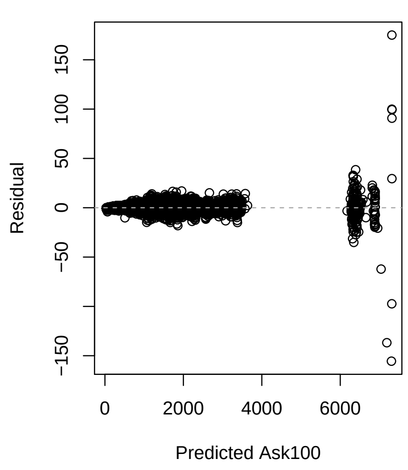
(a) Bid_{100}'s predicted vs observed

(b) Bid_{100}'s residual

Figure 4.66: Performance of the GBM $level_2$ (cascading strategy 2) Bid_{100}



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.67: Performance of the GBM $level_2$ (cascading strategy 1) Ask_{100}

Again, the Bid/Ask residuals patterns for securities with price higher than 4000 (pence) are repeated, as shown in Figure 4.66 and Figure 4.67.

Finally, the RMSE for the GBM $level_2$ (cascading strategy 2) Bid/Ask models are shown in Figure 4.72. Figure 4.69 and Figure 4.70 show detailed performance overviews for the GBM $level_2$ Bid_{100}/Ask_{100} models, respectively. The results and residuals patterns of the strategy are consistent with the previous two strategies, but did not outperform GBM $level_1$ models.

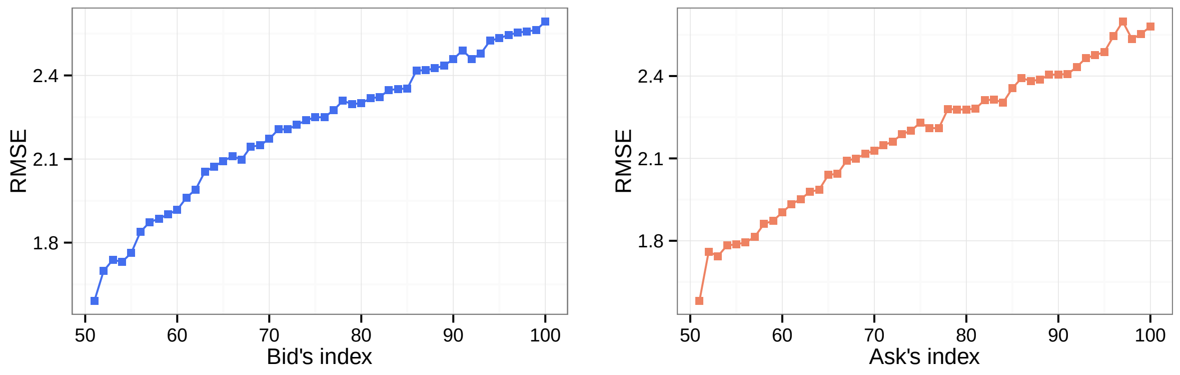
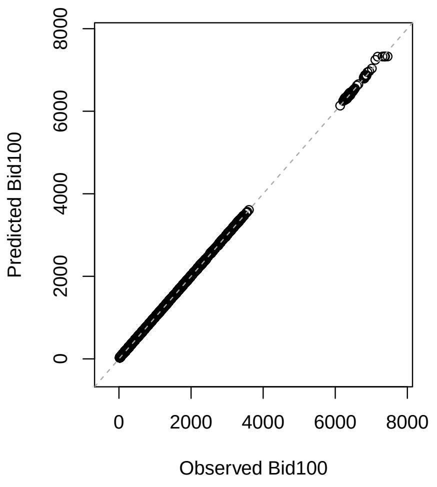
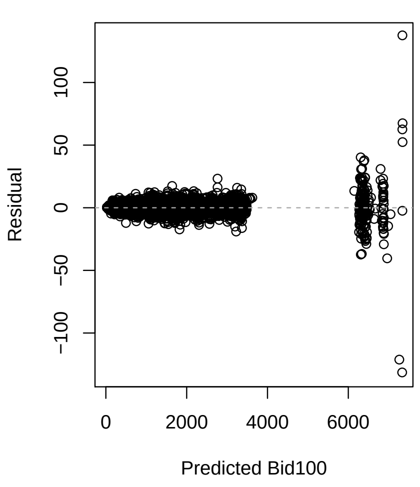


Figure 4.68: RMSE for the GBM $level_2$ (cascading strategy 2) models (test, $N = 30,327$)

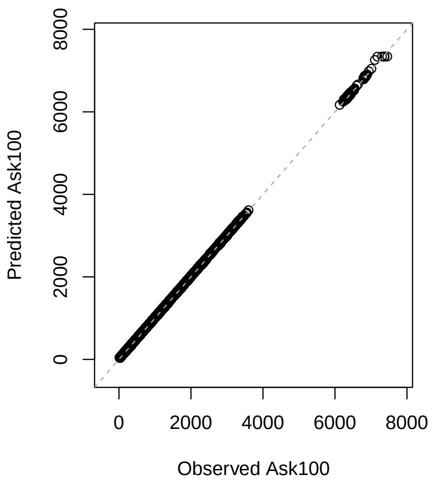


(a) Bid_{100}'s predicted vs observed

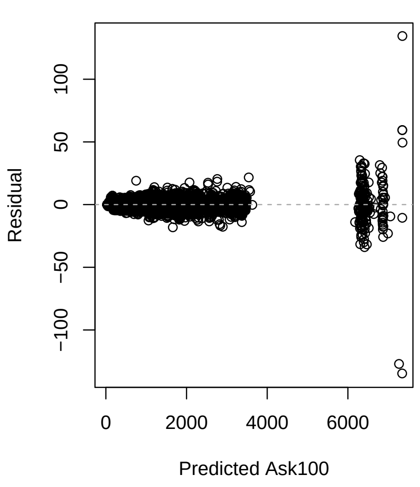


(b) Bid_{100}'s residual

Figure 4.69: Performance of the GBM $level_2$ (cascading strategy 2) Bid_{100}



(a) Ask_{100}'s predicted vs observed



(b) Ask_{100}'s residual

Figure 4.70: Performance of the GBM $level_2$ (cascading strategy 2) Ask_{100}

As presented in Table 4.18, the GBM models for multi-model and cascading-model strategies have slightly different performance, with GBM $level_1$ models having the best performance. It seems that the feeding of GBM $level_1$ models' predictions to GBM $level_2$ models produces lower performance (higher RMSE). However, as will be shown later, this is not the case after the outliers are removed and the RMSE recalculated. The total RMSE for all post-liquidity shock's 50 Bid/50 Ask GBM models, for the three modeling strategies, are slightly higher than the RMSE for the RMSE models, Table 4.17, but significantly lower than NNET models, Table 4.16.

Table 4.18: RMSE statistics for the GBM 50 Bids/50 Ask models

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	1.287	1.670	1.854	1.825	2.055	2.197	91.248
Bids $level_2$ (str. 1)	1.831	2.100	2.291	2.239	2.424	2.528	111.959
Bids $level_2$ (str. 2)	1.591	2.059	2.250	2.210	2.424	2.593	110.486
Asks $level_1$	1.242	1.557	1.771	1.764	1.989	2.208	88.204
Asks $level_2$ (str. 1)	2.213	2.424	2.575	2.565	2.722	2.881	128.240
Asks $level_2$ (str. 2)	1.582	1.980	2.210	2.181	2.391	2.598	109.075

Investigating Relatively Poor Performance for Some Securities

The performance of the NNET, MARS, and GBM post-liquidity shock's Bid/Ask models were evaluated using the same test data set, with 30,327 observations. A clear deterioration in performance was revealed, by the residuals plots, for securities/stocks with Bid/Ask prices greater than 4000 (pence). Such a systematic pattern, which was repeated in all modeling strategies/models, calls for an additional investigation by the modeler to understand/pin down the source of this behavior. Using exploratory data analysis, security/stock number 75 is identified as the only security with Bid/Ask price higher than 4000 (pence). To understand this pattern of relatively poor performance for security number 75, two important factors need to be explored. First, the number of observations for security 75 in $level_1$ and $level_2$ training data sets. Second, how volatile is security 75.

Table 4.19: Number of observations for security 75

Partition	Sec. 75 obs.	Total obs.	Percentage
$level_1$ training	1147	182,388	0.6289%
$level_2$ training	764	121,533	0.6286%
testing dataset	190	30,327	0.6265%

Table 4.20: Prices statistics for security no. 75

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Bid_{50} $level_1$	6040	6300	6370	6457	6495	7530
Bid_{50} $level_2$	6030	6295	6370	6457	6505	7510
Ask_{50} $level_1$	6050	6310	6385	6469	6505	7540
Ask_{50} $level_2$	6040	6305	6380	6469	6515	7520
Bid_{50} test	6130	6306	6362	6475	6514	7440
Ask_{50} test	6140	6320	6372	6487	6524	7455

Table 4.19 shows the number of observations, in training and testing data sets, for security 75. The security has a very small number of observations, compared to all securities, for each training level (around 0.629%). This suggests that the available number of observations is not enough for the NNET, MARS, and GBM learning algorithms to capture the behavior of the security completely. A simple solution to this problem is to collect more observations for security 75. The volatility of security 75 is explored by Table 4.20, where the statistics for Bid_{50}/Ask_{50} prices (prices at the liquidity shock) are presented. It is clear that security 75 prices have a high variance, as compared to the other securities. Since the learning algorithms try to minimize the overall RMSE, the high variance combined with the tiny number of observations, the optimal models failed to capture fully the volatility of security 75. This problem can be solved by separating the observations of security 75 from the market-based training data sets, and training specific Bid/Ask models for security 75 using these observations. However, for simplicity and the purpose of this research, the 190 observations for security 75 are removed from the test data set.

The RMSE for NNET, MARS, and GBM models, previously presented in Table 4.16, Table 4.17, and Table 4.18, are recalculated without security 75 and represented in Table 4.21, Table 4.27, and Table 4.29, receptively. The recalculated RMSE show that security 75, even with only 190 out of 30,327 observations, has significant effect on the total 50 Bid/50 Ask RMSE, especially for the GBM models. Similar to the MARS models, the feeding of GBM $level_1$ models' predictions to GBM $level_2$ models produces higher performance.

Table 4.21: RMSE statistics for the NNET $level_1$ Bids/Asks models (without security 75)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids RMSE	6.091	6.296	6.341	6.361	6.429	6.598	318.037
Asks RMSE	6.209	6.355	6.443	6.439	6.496	6.693	321.926

Table 4.22: RMSE statistics for the MARS 50 Bids/50 Ask models (without security 75)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	0.000	0.844	1.080	1.027	1.303	1.505	51.332
Bids $level_2$ (str. 1)	0.000	0.821	1.078	1.025	1.303	1.505	51.255
Bids $level_2$ (str. 2)	0.000	0.844	1.078	1.027	1.302	1.505	51.337
Asks $level_1$	0.000	0.745	1.059	1.005	1.300	1.494	50.225
Asks $level_2$ (str. 1)	0.000	0.742	1.057	1.002	1.288	1.490	50.116
Asks $level_2$ (str. 2)	0.000	0.742	1.057	1.003	1.288	1.490	50.126

Table 4.23: RMSE statistics for the GBM 50 Bids/50 Ask models (without security 75)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	1.023	1.338	1.516	1.489	1.684	1.827	74.441
Bids $level_2$ (str. 1)	0.441	0.946	1.186	1.148	1.401	1.581	57.419
Bids $level_2$ (str. 2)	0.651	1.308	1.521	1.472	1.676	1.837	73.618
Asks $level_1$	1.013	1.255	1.457	1.449	1.640	1.827	72.430
Asks $level_2$ (str. 1)	1.044	1.290	1.502	1.488	1.688	1.852	74.381
Asks $level_2$ (str. 2)	0.660	1.218	1.484	1.433	1.659	1.838	71.643

In Section 4.8, model fusion will be used to combine the predictions of the three modeling strategies for the same learning algorithm (MARS/GBM), to investigate if the injection of model diversity at data and features levels will result in better performance (lower RMSE). The effect of adding model diversity at the learning algorithm (by combining MARS and GBM models) will be also investigated.

4.7.3 Computational Cost

Often the computational cost of building machine/statistical learning models is ignored for small and/or low-dimensional datasets. However, for large and/or high-dimensional datasets, the computational cost of building such predictive models becomes a critical factor to consider in choosing the best learning algorithm for the application at hand. In this section, the learning algorithms (NNET, MARS, GBM, SVM, and RF) are compared based on the computing power required, model selection time, training time, and prediction time.

Model Selection Time

In Section 4.6, grid-search and 10-fold cross-validation was used to select the optimal hyper-parameters for the NNET, MARS, GBM, SVM, and RF learning algorithms. Given the intensive computation required to generate a large number of models per learning algorithm, a powerful computing resource is essential to handle memory constrains, and speed-up the model selection process by using parallel computing to distribute the 10-fold CV over several cores/computers. For this research, an Amazon EC2 instance, with 32 cores and 65 GB RAM, was used.

Table 4.24: Model selection time (using an Amazon EC2 instance with 32 cores and 65 GB RAM)

Model	Preproc.	Level1	Level2 (str.1)	Level2 (str.2)
NNET Bid100	N.	8.069 hours	N/A	N/A
NNET Bid100	T. & N.	7.886 hours	N/A	N/A
NNET Ask100	T. & N.	5.578 hours	N/A	N/A
MARS Bid100	N.	4.932 mins	3.128 mins	1.619 mins
MARS Bid100	T. & N.	5.238 mins	N/A	N/A
MARS Ask100	N.	1.991 mins	2.637 mins	1.644 mins
GBM Bid100	N.	42.53 mins	N/A	N/A
GBM Bid100	T. & N.	45.22 mins	34.490 mins	24.32 mins
GBM Ask100	T. & N.	49.57 mins	39.813 mins	23.83 mins
SVM ($\epsilon = 1$) Bid100	N.	40.25 secs	N/A	N/A
SVM ($\epsilon = 1$) Bid100	T. & N.	2.160 mins	N/A	N/A
SVM ($\epsilon = 0.1$) Bid100	N.	1.727 mins	N/A	N/A
SVM ($\epsilon = 0.1$) Bid100	T. & N.	3.592 mins	N/A	N/A
SVM ($\epsilon = 0.01$) Bid100	N.	29.83 mins	N/A	N/A
SVM ($\epsilon = 0.01$) Bid100	T. & N.	1.68 hours	N/A	N/A
RF (T=1000) Bid100	T. & N.	29.07 mins	N/A	N/A
RF (T=3000) Bid100	T. & N.	1.119 hours	N/A	N/A
RF (T=5000) Bid100	T. & N.	37.73 hours	N/A	N/A

Table 4.24 lists the model selection time for $level_1$ and $level_2$ modeling strategies. Since NNET, SVM, and RF algorithms were only used for $level_1$ strategy, they are not applicable (N/A) for $level_2$ strategies. The results show significant differences between the learning

algorithms. For the NNET, MARS, and GBM (selected algorithms to train post-liquidity shock's Bid/Ask models), the MARS algorithm is the fastest, while NNET algorithm is the worst. The table also shows that a slight reduction in the value of ϵ for the SVM algorithm increases the model selection time. On the other hand, increasing the number of trees for the RF algorithm significantly increases its model selection time.

Training Time

To minimize the financial cost of renting Amazon instances, the Bid/Ask models for $level_1$ and $level_2$ strategies were trained using a PC with 6 cores and 8 GB RAM.

Table 4.25: Training time statistics for the 50 Bid/50 Ask models (using a PC with 6 cores and 8 GB RAM)

Models	Min.	Median	Mean	Max.	Total
$level_1$ Bids					
NNET	N/A	N/A	N/A	N/A	51.45 hours
MARS	53.14 secs.	54.21 secs.	55.68 secs.	70.10 secs.	46.40 mins.
GBM	27.02 mins.	30.96 mins.	30.41 mins.	33.57 mins.	25.84 hours
$level_1$ Asks					
NNET	N/A	N/A	N/A	N/A	63.95 hours
MARS	41.61 secs.	41.97 secs.	42.23 secs.	45.01 secs.	35.19 mins.
GBM	29.77 mins.	30.81 mins.	30.87 mins.	35.02 mins.	25.73 hours
$level_2$ (st1) Bids					
MARS	38.84 secs.	61.90 secs.	60.98 secs.	85.00 secs.	50.82 mins
GBM	16.61 mins.	25.89 mins.	25.27 mins.	33.90 mins.	21.05 hours
$level_2$ (st1) Asks					
MARS	28.02 secs.	48.60 secs.	49.31 secs.	79.04 secs.	41.09 mins.
GBM	20.62 mins.	36.20 mins.	34.47 mins.	48.08 mins.	28.72 hours
$level_2$ (st2) Bids					
MARS	3.94 secs.	23.92 secs.	23.86 secs.	44.56 secs.	19.88 mins.
GBM	10.19 mins.	66.13 mins.	67.24 mins.	124.7 mins.	56.03 hours
$level_2$ (st2) Asks					
MARS	3.78 secs.	23.20 secs.	23.36 secs.	43.65 secs.	19.46 mins.
GBM	8.92 mins.	73.22 mins.	72.53 mins.	139.5 mins.	60.44 hours

Table 4.25 presents the training time for NNET, MARS, and GBM learning algorithms. Again, the MARS algorithm outperforms the NNET and GBM algorithms by huge margins. The GBM algorithm is faster than the NNET algorithm by about 50%.

Prediction Time

How long a model takes to produce a prediction becomes vitally important for real-time applications, such as algorithmic trading systems. Table 4.26 presents the prediction time of the 50 Bids/50 Asks prices. The NNET models are incredibly fast, while the MARS models significantly outperform the GBM models only for $level_2$ cascading-model (strategy 2).

Table 4.26: Prediction time statistics for the 50 Bid/50 Ask prices (using a PC with 6 cores and 8 GB RAM, test $N = 30,327$)

Models	Min.	Median	Mean	Max.	Total
$level_1$ Bids					
NNET	N/A	N/A	N/A	N/A	0.895 secs.
MARS	16.34 secs.	16.71 secs.	16.85 secs.	20.61 secs.	14.05 mins.
GBM	16.41 secs.	16.54 secs.	16.56 secs.	17.03 secs.	13.80 mins.
$level_1$ Asks					
NNET	N/A	N/A	N/A	N/A	1.01 secs.
MARS	12.04 secs.	12.25 secs.	12.35 secs.	12.91 secs.	10.29 mins.
GBM	25.69 secs.	25.88 secs.	25.96 secs.	26.91 secs.	21.63 mins.
$level_2$ (st1) Bids					
MARS	6.42 secs.	16.51 secs.	16.06 secs.	44.41 secs.	13.38 mins.
GBM	15.50 secs.	15.97 secs.	16.34 secs.	19.38 secs.	13.62 mins.
$level_2$ (st1) Asks					
MARS	3.155 secs.	5.676 secs.	5.839 secs.	9.546 secs.	4.87 mins.
GBM	5.87 secs.	6.02 secs.	5.99 secs.	6.27 secs.	4.99 mins.
$level_2$ (st2) Bids					
MARS	0.41 secs.	2.48 secs.	2.54 secs.	4.73 secs.	2.54 mins.
GBM	17.47 secs.	17.65 secs.	17.69 secs.	18.24 secs.	14.74 mins.
$level_2$ (st2) Asks					
MARS	0.38 secs.	2.51 secs.	2.53 secs.	4.81 secs.	2.11 mins.
GBM	17.60 secs.	17.75 secs.	17.78 secs.	18.08 secs.	14.81 mins.

4.8 Model Fusion

In Section 4.7.2, it was shown that feeding the predictions of $level_1$ models to $level_2$ (strategies 1 and 2) improves the performance (lower RMSE) of both MARS and GBM $level_1$ models. In this section, a second layer of model fusion is added by combining the predictions of the three modeling strategies. Due to the complexity of model fusion by stacking (e.g. 200 new fusion models need to be trained), fusion by taking the arithmetic mean of the three modeling strategies for each post-liquidity shock's Ask/Bid prices. Two model fusion approaches will be investigated.

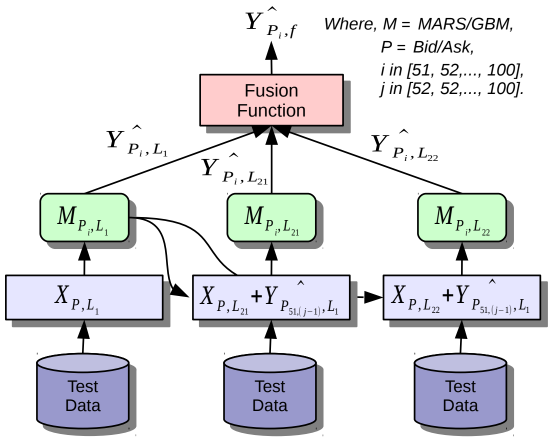


Figure 4.71: Model fusion topology of models with diversity injected at the data and feature levels

Table 4.27: RMSE of Mean Fusion - MARS modeling strategies (model diversity at the data and feature levels)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	0.000	0.844	1.080	1.027	1.303	1.505	51.332
Bids $level_2$ (str. 1)	0.000	0.821	1.078	1.025	1.303	1.505	51.255
Bids $level_2$ (str. 2)	0.000	0.844	1.078	1.027	1.302	1.505	51.337
Bids mean fusion	0.000	0.836	1.078	1.025	1.303	1.505	51.233
Asks $level_1$	0.000	0.745	1.059	1.005	1.300	1.494	50.225
Asks $level_2$ (str. 1)	0.000	0.742	1.057	1.002	1.288	1.490	50.116
Asks $level_2$ (str. 2)	0.000	0.742	1.057	1.003	1.288	1.490	50.126
Asks mean fusion	0.000	0.743	1.057	0.999	1.270	1.491	49.950

First, the fusion of MARS/GBM models separately (model diversity injected at the data and feature levels). Figure 4.27 shows the fusion topology for this approach. The predictions of the three modeling strategies are fed to the fusion function (arithmetic mean) to produce a final prediction for each Ask_i/Bid_i, where $i \in [51, 100]$. Table 4.27 presents the RMSE for the MARS's three modeling strategies and the mean fusion. The results for GBM modeling strategies with the mean fusion are shown in Table 4.29. Both tables indicate that the performance of each learning algorithm's models is further improved by combining all strategies.

Table 4.28: RMSE of Mean Fusion - GBM modeling strategies (model diversity at data and feature levels)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids $level_1$	1.023	1.338	1.516	1.489	1.684	1.827	74.441
Bids $level_2$ (str. 1)	0.441	0.946	1.186	1.148	1.401	1.581	57.419
Bids $level_2$ (str. 2)	0.651	1.308	1.521	1.472	1.676	1.837	73.618
Bids mean fusion	0.435	0.969	1.200	1.157	1.400	1.574	57.860
Asks $level_1$	1.013	1.255	1.457	1.449	1.640	1.827	72.430
Asks $level_2$ (str. 1)	1.044	1.290	1.502	1.488	1.688	1.852	74.381
Asks $level_2$ (str. 2)	0.660	1.218	1.484	1.433	1.659	1.838	71.643
Asks mean fusion	0.555	0.941	1.201	1.173	1.415	1.613	58.665

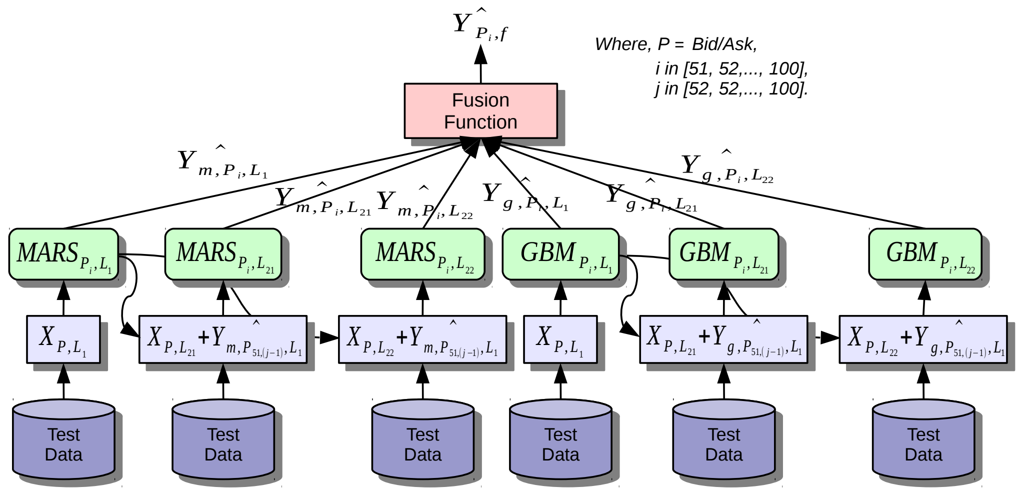


Figure 4.72: Model fusion topology of models with diversity injected at data, feature, and learning algorithm levels

The second fusion approach is the fusion of all MARS and GBM models together (diversity injected at data, feature, and learning algorithm levels). Figure 4.72 shows the fusion topology of this approach, where RMSE results are presented in Table 4.29. Although adding diversity at the learning algorithm did not produce a better performance (slightly worse) than the MARS's modeling strategies, it outperformed all GBM strategies, including the performance enhancement achieved by combining only GBM's modeling strategies. This slightly lower performance might be enhanced by applying weighted mean or model stacking by first training the fusion models, to combine MARS and GBM predictions, using the fusion dataset (45,599 observations), and then applying the trained models to the test dataset. However, for the purpose of this research, this complex process is considered as future work.

Table 4.29: RMSE of Mean Fusion - MARS and GBM modeling strategies (model diversity at data, feature, and learning algorithm levels)

Price	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Total
Bids mean fusion	0.218	0.859	1.101	1.052	1.317	1.505	52.588
Asks mean fusion	0.278	0.781	1.074	1.031	1.290	1.498	51.536

4.9 Summary

In this chapter, the proposed framework for ensemble predictive modeling was validated using a complex regression case study, to predict the stock market's short-term behavior following liquidity shocks. The developed ensembles outperformed the performance of the individual models. Moreover, the effectiveness of model diversity approaches was evaluated. Finally, several predictive modeling strategies and learning algorithms were compared based on their performance and computational cost. In the next chapter, the proposed framework will be applied to a high-dimensional classification case study.

Chapter 5

Classification Case Study: Predicting a Biological Response of Molecules from Their Chemical Properties

5.1 Introduction

The activity of predicting the biological response of molecules from their chemical properties is classified as Quantitative Structure-Activity Relationship (QSAR) [6]. QSAR methods require the identification of at least one lead molecule that elicits the activity. Then, the QSAR method correlates the computed properties from molecules' structure with their activities. The resulting correlation is then used to predict the activity of hypothetical molecules. This correlation may also help in understanding the structural features that contribute to activities. Based on the set of features of molecules that are correlated with activity, and methods used to identify the correlation, QSAR methods are broadly classified into *traditional* methods, and *3D* methods [39].

QSAR *traditional* methods use a combination of three types of features. The first type is bulk molecular properties, such as computed/measured molar refractivity, molar volume, and computed/measured octanol/water partition coefficient. The second type is topological and geometrical features, such as the lengths of the principal axes, aspect ratios, the number of aromatic bonds, and connectivity indices. The third type only applies in cases where the molecules consist of substitutions of a common parent structure. Traditional methods predictive accuracy is high in some systems and poor in others. QSAR *3D* methods use as features direct measurements of the three-dimensional shapes of molecules and three-dimensional distribution of charges in and about molecules. 3D methods often produce better predictions than traditional methods [39].

The development of a new drug mostly depends on trial and error. It typically involves synthesizing thousands of compounds that finally becomes a drug. This process requires, on average, thousands of dollars, and a considerable amount of time, ranging from few days to few weeks. As a consequence, drug discovery is extremely expensive and slow [39]. Therefore, the ability to accurately predict the biological activity of molecules, and understand the rationale behind those predictions would be of great value to the pharmaceutical industry.

The data for this classification case study is obtained from the Kaggle.com's competition: "Predicting a Biological Response" [41] held between March 16th, 2012 and June 15th, 2012. The objective of the competition was to build a predictive model to relate optimally molecular information to an actual biological response. The competition's sponsor (Boehringer Ingelheim Ltd.) provided the data in the comma separated values (CSV) format. Each row in the data set represents a molecule. The first column contains experimental data describing an actual biological response; the molecule was seen to elicit this response (1 or Active), or not (0 or Inactive). The remaining columns represent molecular descriptors (D1 through D1776). These are calculated properties that can capture some of the characteristics of the molecule - for example, size, shape, or elemental constitution. The descriptor matrix has been normalized (by the competition's sponsor). Only the training data (with 3751 observations, and 1776 variables/predictors) is used to investigate the thesis objectives and develop the multi-classifier system.

5.2 Objective

When building a predictive model for a domain-specific area, such as drug discovery, one approach is to create a model from theory and then adjust its parameters based on the observed data. However, in most real-life applications such models are not available. Moreover, basic knowledge about the relationships between the input variables and outcomes is not available. These limitations lead to an alternative approach using predictive modeling to build a model directly from the data. The standard approach in data-driven modeling is to build a single strong predictive model. However, recently, the focus has shifted towards creating an ensemble of models or for the same task.

The primary objective of this case study is to apply the proposed framework for ensemble predictive modeling to develop an ensemble system, for a high-dimensional data application, to predict the biological response of molecules from their chemical properties. Our objective is to develop an ensemble system that outperforms the best base classifier in several performance metrics, namely, the area under ROC curve, specificity, sensitivity, accuracy, and Kappa. Some of the work in this case study has been published in [3, 5].

As secondary objectives, this case study is used to investigate experimentally:

- The effectiveness of model diversity approaches: evaluate the effect of injecting model diversity, at data, feature, or/and learning algorithm levels, on the performance of the ensemble.
- The relative performance of several feature selection approaches: for the same learning algorithm, compare the performance of classifiers built using all features, PCA, predictors' area under ROC curve, and Relief algorithm.
- The relative performance of learning algorithms: for the same feature selection approach, compare the performance of classifiers build using Stochastic Gradient Boosting Machines (GBM), Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), and Flexible Discriminate Analysis (FDA).

5.3 Data Preparation

As discussed in the proposed framework, Section 3.3, the data preparation phase typically involves exploratory data analysis, feature engineering, data processing, and data partition.

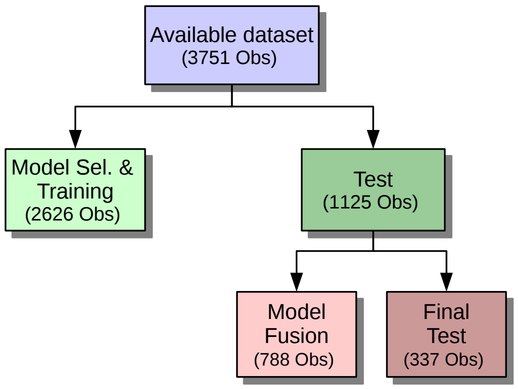


Figure 5.1: Data partition

However, the available dataset was normalized (centered and scaled) by the competition's sponsor. Also, the names of the predictors were removed. Due to the large number of predictors (1776), the exploratory data analysis steps performed are only presented in the *R* code. Figure 5.1 shows the data partition for different modeling tasks. The stratified sampling was used to maintain similar class distribution across all data sets.

5.4 Model Building

Due to the relatively small data set, model diversity at the data level is achieved by the built-in bagging and boosting within the GBM learning algorithm, and bagging within the Random Forest learning algorithm.

At the feature level, model diversity is achieved by training classifiers on subsets of features selected by five feature selection approaches. In addition, an extra layer of model diversity at feature level is added to the Random Forest models by its built-in algorithm.

Model diversity at learning algorithm level is mainly achieved using several learning algorithms. A second layer of diversity is added, where different optimal hyper-parameters values are selected for the same learning algorithm (due to the use of data sets with different features).

5.5 Feature Selection

This section describes the feature selection approaches used to inject systematically model diversity at the feature level.

5.5.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most widely used methods of modern data analysis. It is a simple, non-parametric method able to reduce a complex high-dimensional dataset to a lower dimension. PCA identifies relationships by generating new variables/components which are linear combinations of variables that have common variation [86].

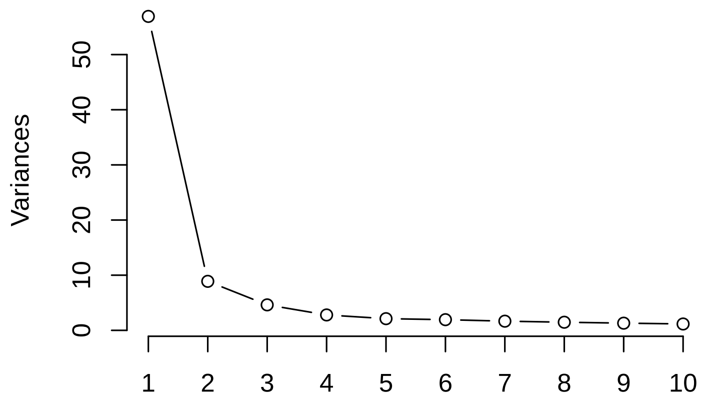


Figure 5.2: Variance explained by first 10 PCA components

For our data set, PCA generates 1776 components (equals to the number of predictors). Figure 5.2 shows the variance explained by the first 10 PCA components. Around 70% of the variance is explained by first two components. However, plotting the first component against the second component, as shown in Figure 5.3, reveals the complexity of the data set, suggesting that more components are needed to separate the classes.

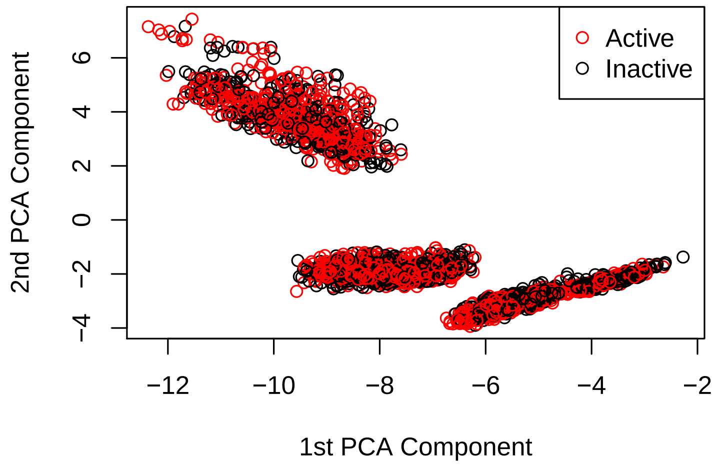


Figure 5.3: First vs. second PCA components

Several methods have been proposed to determine the number of PCA components to retain for further analysis and predictive modeling [63]. We use two popular methods to build two GBM ensembles, namely, the proportion of total variance explained, and Kaiser-Guttman rule.

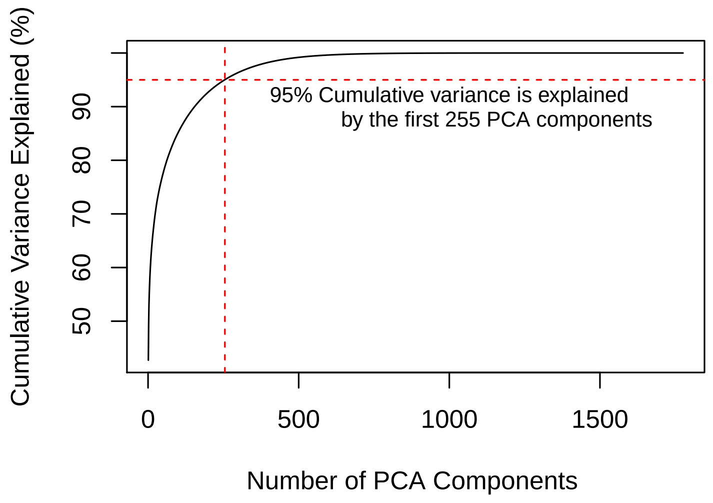


Figure 5.4: Cumulative variance explained by PCA components

In the proportion of total variance explained method, enough components are selected to explain at least x% of the total variance. As shown in Figure 5.4, the first 255 PCA components are required to explain 95% of the total variance.

In Kaiser-Guttman rule [32], PCA components with eigenvalues larger than 1.0 are selected. Using this rule, the first 11 PCA components of our dataset are chosen to build our models.

5.5.2 Predictors' Area Under ROC Curve

Receiver Operating Characteristic (ROC) Curves [7, 13, 24] are general methods used to determine an effective threshold such that values above the threshold are indicative of a particular event. With categorical outcomes and numeric predictors, the area under the ROC curve can be used to quantify predictor relevance. If the predictor could perfectly separate the classes, there would be a cutoff for the predictor that would achieve a sensitivity and specificity of 1 and the area under the curve would be 1. An entirely irrelevant predictor would have an area under the curve of approximately 0.5 [45].

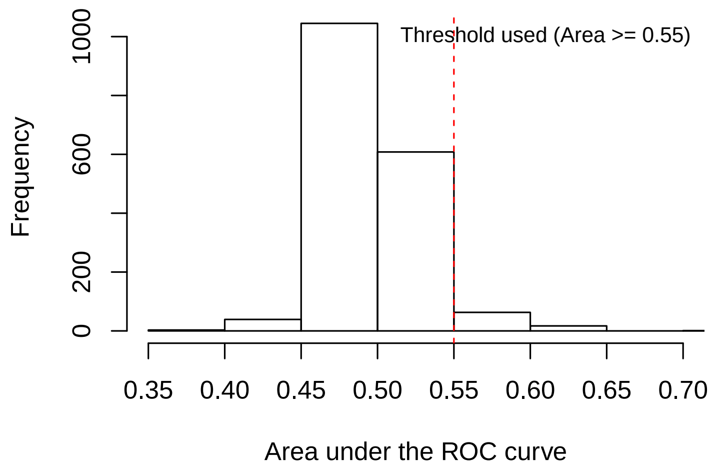


Figure 5.5: Frequency of predictors vs. their area under ROC curve

Figure 5.5 shows the frequency of predictors versus their area under ROC. Using the area greater than or equal to 0.55 as our threshold, only 81 predictors are selected to build our models.

5.5.3 Relief Algorithm

The Relief algorithm [43] is a generic method originally developed for classification problems with two classes. It attempts to estimate the quality of predictors according to how well their values distinguish between instances that are near to each other. For a randomly selected training instant *Ri*, the Relief algorithm finds its two nearest neighbors: one from the same class, called the nearest hit *H*, and the other from the different class, called the nearest miss *M*. It updates the quality estimation *W[P]* for all predictors *P* depending on their values for *Ri*, *M*, and *H*. If instances *Ri* and *H* have different values of the predictor *P* then the predictor *P* separates two instances with the same class which is not desirable so the quality estimation *W[P]* is decreased. On the other hand, if instances *Ri* and *M* have different values of the predictor *P* then the predictor *P* separates two instances with different class values which is desirable so the quality estimation *W[P]* is increased. The whole process is repeated for m times, where m is a user-defined parameter [71].

The ReliefF algorithm [44] is an improved version of the Relief algorithm that can be used for classification problems with more than two classes. It uses more than a single nearest neighbor, and can handle missing predictor values. The RReliefF algorithm [70] is another extension to handle regression problems [45].

In contrast to the majority of heuristic methods for estimating the quality of predictors, which assume the conditional independence of the predictors, relief algorithms can estimate the quality of the predictors with high dependencies between themselves [71].

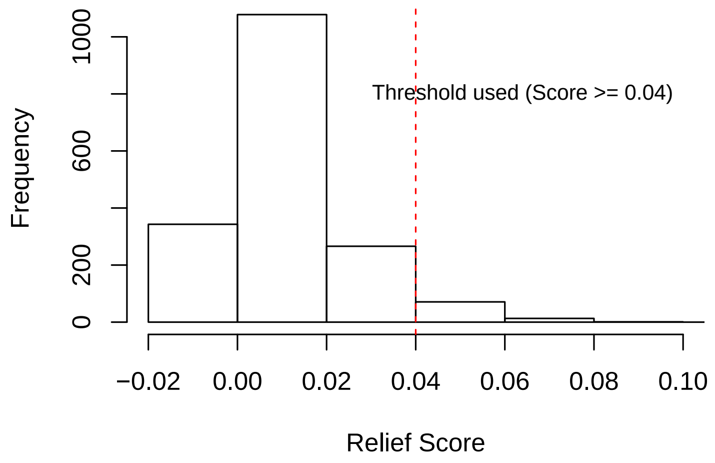


Figure 5.6: Frequency of predictors vs. their Relief score

Figure 5.6 shows the frequency of predictors versus their Relief score. Kira and Rendell [43] suggest that a threshold can be much smaller than $\frac{1}{\sqrt{\alpha m}}$, where α is the desired false-positive rate, and m is the number of randomly selected training instances used to calculate relief scores. For our dataset, with 5% false positive rate, and $m = 2626$ the suggested threshold is ≈ 0.087. However, this value seems to be inappropriate for our predictors' scores. Therefore, we decided to use a score greater than or equal to 0.04 as our threshold leading to the selection of 86 predictors.

5.6 Model Selection and Training

Many learning algorithms have important parameters called "hyper-parameters". These parameters can not be directly estimated from the data using analytical formulas. Most of these parameters control the complexity of the model, and hence, poor choices for their values can lead to under- or over-fitting [45].

Each learning algorithms used in this case study, Stochastic Gradient Boosting Machines (GBM), Support Vector Machine (SVM), Random Forest (RF), Flexible Discriminate Analysis (FDA), and K-Nearest Neighbors (KNN), contains, at least, one hyper-parameter that needs to be tuned.

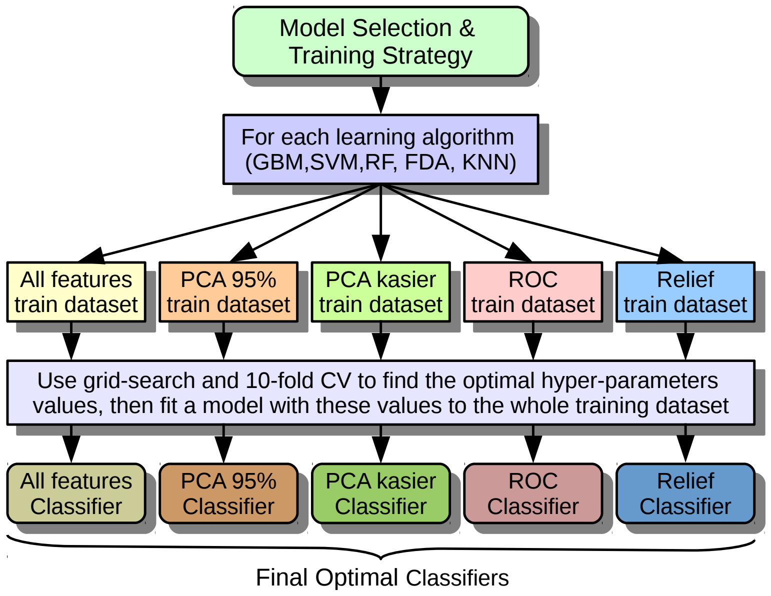

Model Selection & Training Strategy
For each learning algorithm (GBM,SVM,RF, FDA, KNN)
All features train dataset
PCA 95% train dataset
PCA kasier train dataset
ROC train dataset
Relief train dataset
Use grid-search and 10-fold CV to find the optimal hyper-parameters values, then fit a model with these values to the whole training dataset
All features Classifier
PCA 95% Classifier
PCA kasier Classifier
ROC Classifier
Relief Classifier
Final Optimal Classifiers

Figure 5.7: Model selection and training strategy

Figure 5.7 shows the model selection and training strategy. First, for each learning algorithm and feature selection approach, use grid search and 10-fold cross-validation, shown in Figure 5.8, to find the optimal hyper-parameters. Then, fit a classifier, with the optimal hyper-parameters, using the whole training dataset. Given that five learning algorithms and five feature selection approaches are used, there will be twenty-five final classifiers.

Model selection, for such a high-dimensional data, is unfeasible using a local PC. To hand the intensive computations, an instance of Amazon Elastic Cloud EC2, with 32 cores and 65 GB RAM, is used for model selection and training. The following sections present the results of the model selection and training strategy, shown in Figure 5.7.

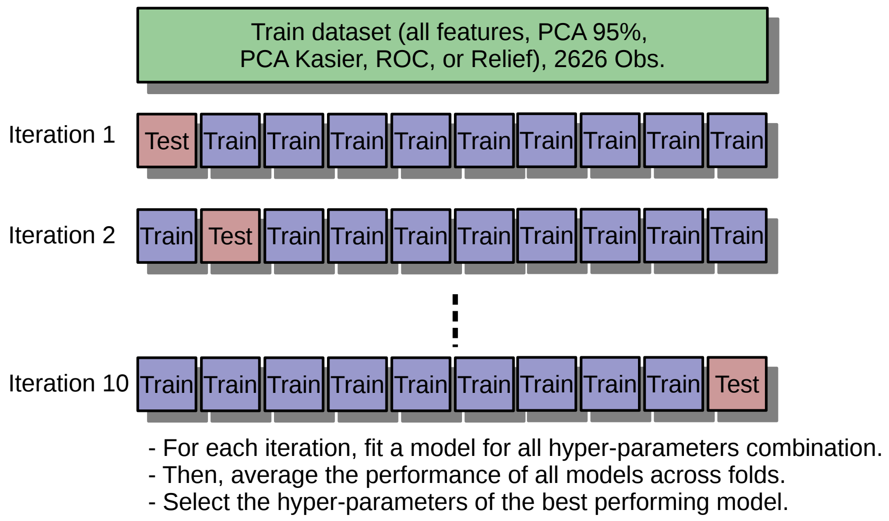

Train dataset (all features, PCA 95%,
PCA Kasier, ROC, or Relief), 2626 Obs.
Iteration 1
Iteration 2
Iteration 10
Test
Train
- For each iteration, fit a model for all hyper-parameters combination.
- Then, average the performance of all models across folds.
- Select the hyper-parameters of the best performing model.

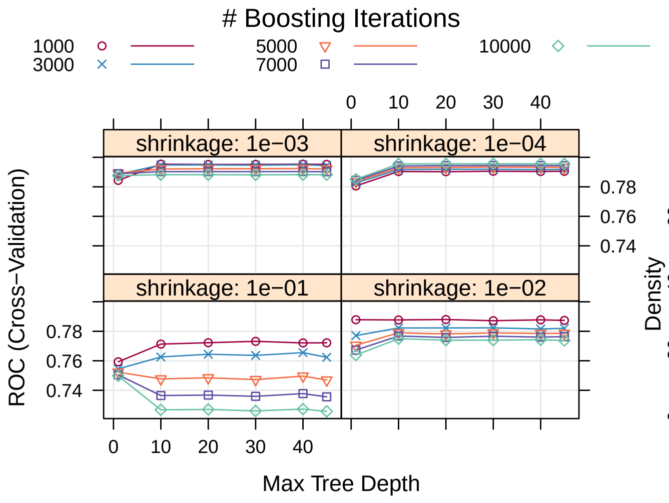
Figure 5.8: Model selection using grid-search and 10-fold CV

5.6.1 Stochastic Gradient Boosting Machine (GBM)

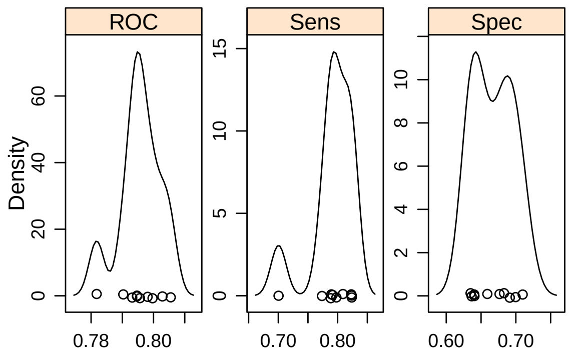
The Stochastic Tree-Based GBM learning algorithm [29] has five main hyper-parameters that control the complexity and performance of a GBM classifier. These are the loss function (distribution), the number of iterations (n.trees), the depth of each tree (interaction.depth), the shrinkage (or learning rate), and the subsampling rate (bag.fraction).

For classification problems, the *bernoulli* distribution, and $bag.fraction = 0.5$ are recommended for the loss function and subsampling rate respectively [69]. In our case study, the remaining hyper-parameters: n.trees, interaction.depth, and shrinkage are tuned over 5, 6, and 4 values respectively using grid-search and 10-fold cross validation. Five GBM optimal classifiers (one classifier per feature selection approach) are selected from a total of 6005 trained classifiers.

Figure 5.9 to Figure 5.13 present the results of the GBM model selection for each feature selection approach.

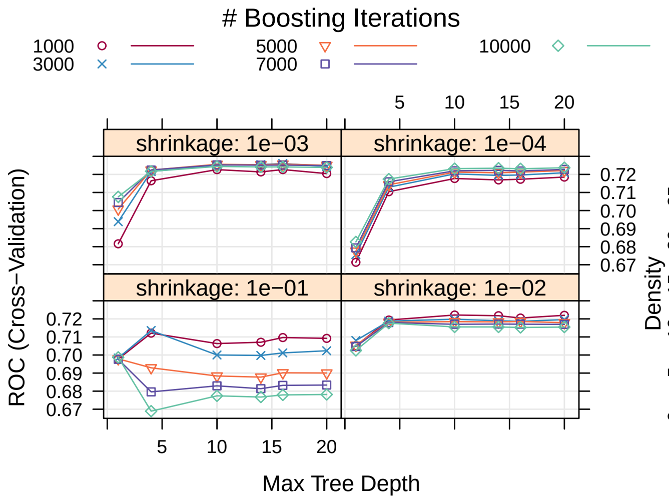


(a) Hyper-parameters grid search

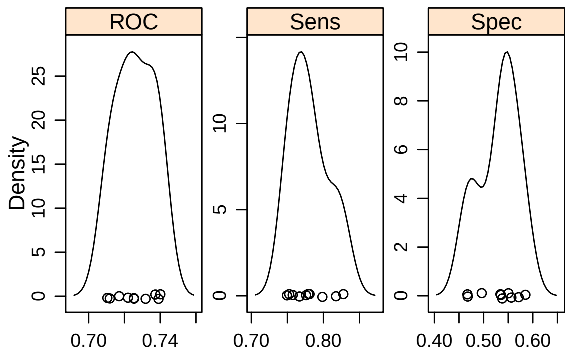


(b) Optimal classifier resampling accuracy

Figure 5.9: GBM model selection using all 1776 predictors

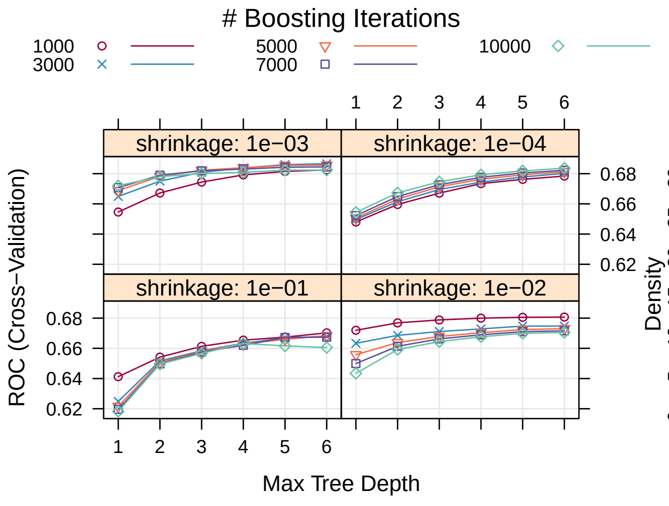


(a) Hyper-parameters grid search

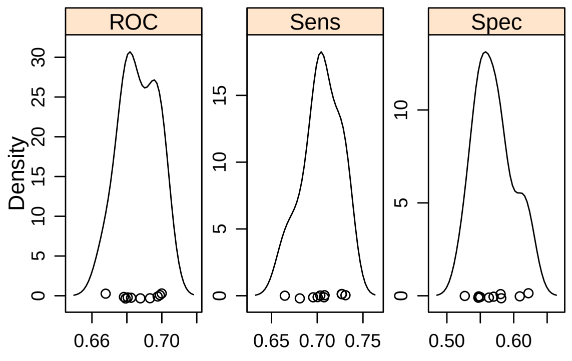


(b) Optimal classifier resampling accuracy

Figure 5.10: GBM model selection using 255 PCA components with 95% Var.

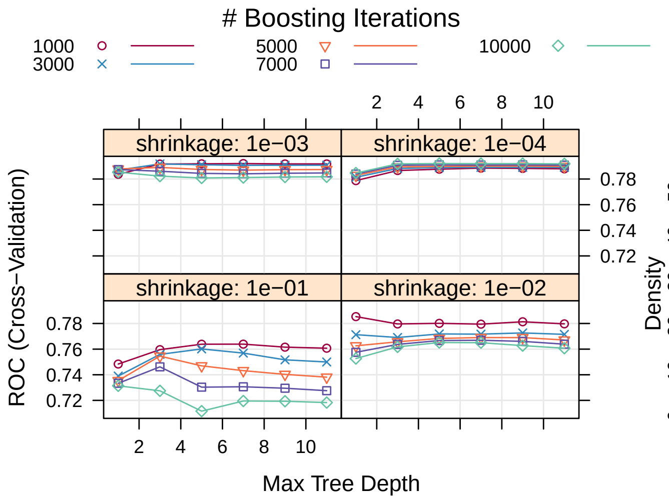


(a) Hyper-parameters grid search

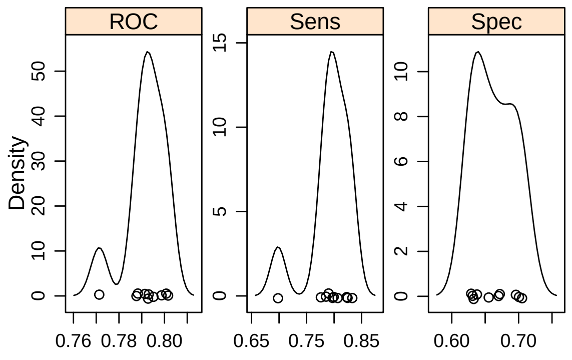


(b) Optimal classifier resampling accuracy

Figure 5.11: GBM model selection using 11 PCA components (Kaiser rule)

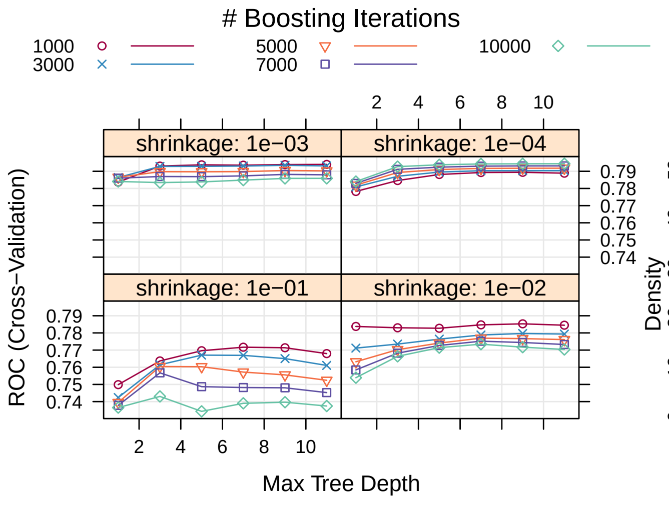


(a) Hyper-parameters grid search

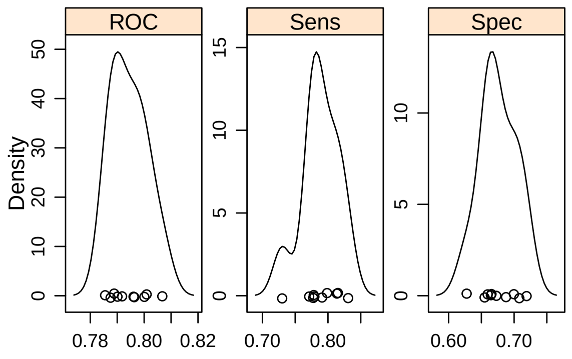


(b) Optimal classifier resampling accuracy

Figure 5.12: GBM model selection using 81 predictors (Area under ROC)



(a) Hyper-parameters grid search



(b) Optimal classifier resampling accuracy

Figure 5.13: GBM model selection using 86 predictors (Relief algorithm)

Table 5.1: GBM selected classifiers

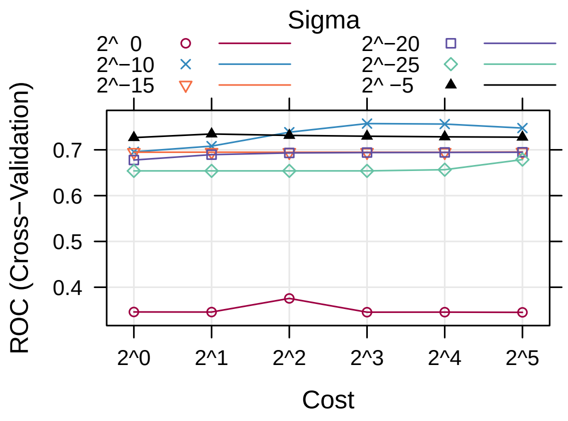
Model	Shrinkage	interaction.depth	n.trees	ROC(%)	ROC SD(%)
GBM_ALL	0.0001	30	10000	79.57	0.67
GBM_PCA_95	0.001	16	3000	72.60	1.09
GBM_PCA_Kasier	0.001	6	3000	68.66	1.07
GBM_ROC	0.0001	5	10000	79.21	0.87
GBM_Relief	0.0001	11	10000	79.44	0.68

5.6.2 Support Vector Machine (SVM)

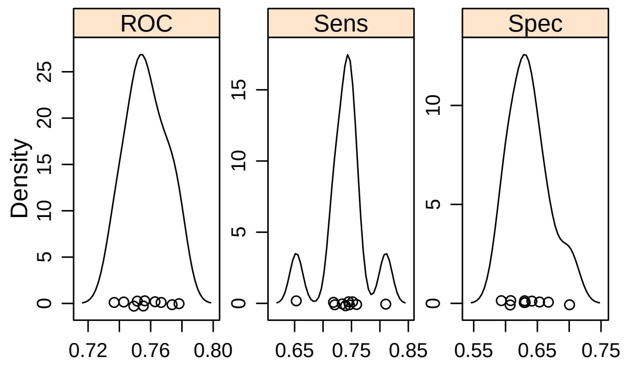
The kernel trick allows the SVM [91] classifier to produce extremely flexible decision boundaries. This complexity is controlled by the choice of the kernel function parameters and the cost value. For example, if the cost value is too low, the SVM classifier most likely will under-fit the data. Conversely, if the cost value is too high, the SVM classifier will probably over-fit the data, especially if the kernel parameter has a large value. Therefore, these hyper-parameters should be appropriately tuned (e.g., using 10-fold CV) to find a reasonable balance between under- and over-fitting [45].

For this classification study case, a SVM with a radial basis function is used to build five SVM base classifiers. Two hyper-parameters, σ and $Cost$, are tuned using grid-search and 10-fold CV to select the optimal model. For each feature selection approach, an optimal model is selected out of 36 unique models [6 (σ values) $\times$ 6 ($Cost$ values)] with total 361 trained models [36 (unique models) $\times$ 10 (10-fold CV) + 1 (complete dataset)]. The total number of trained models for all feature selection approaches is 1805 models [361 (models per feature selection) $\times$ 5 (number of feature selection approaches)].

Figure 5.14 to Figure 5.18 present the results of the SVM model selection for each feature selection approach.

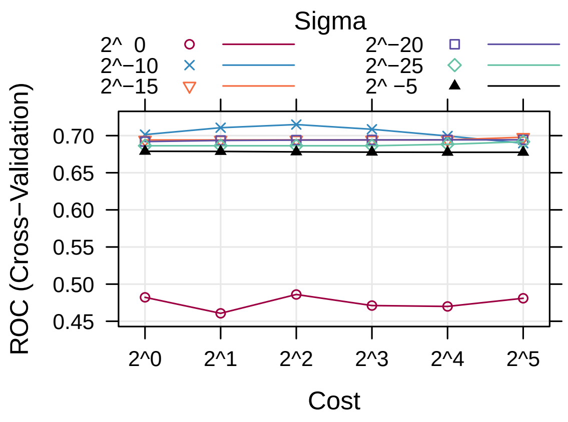


(a) Hyper-parameters grid search

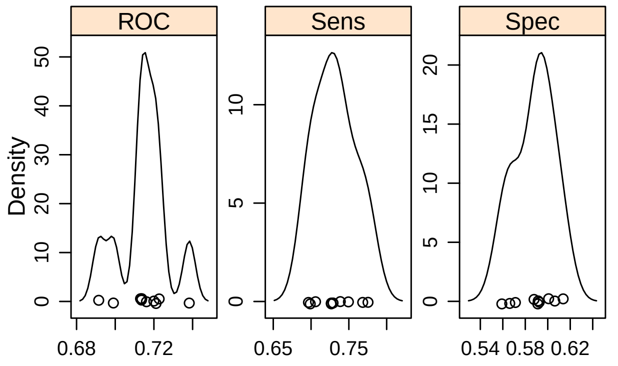


(b) Optimal classifier resampling accuracy

Figure 5.14: SVM model selection all 1776 predictors

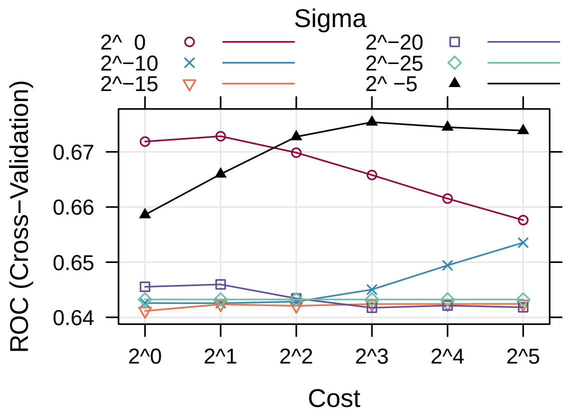


(a) Hyper-parameters grid search

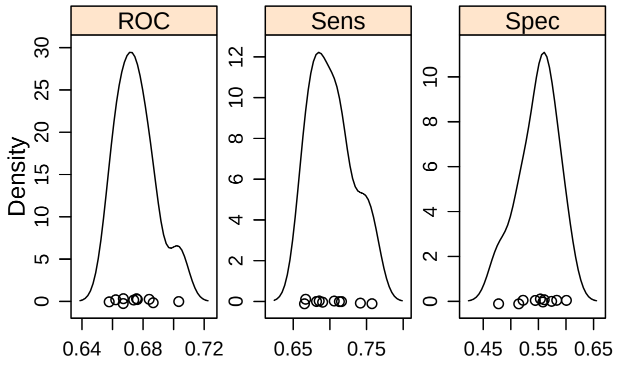


(b) Optimal classifier resampling accuracy

Figure 5.15: SVM model selection using 255 PCA components with 95% Var.

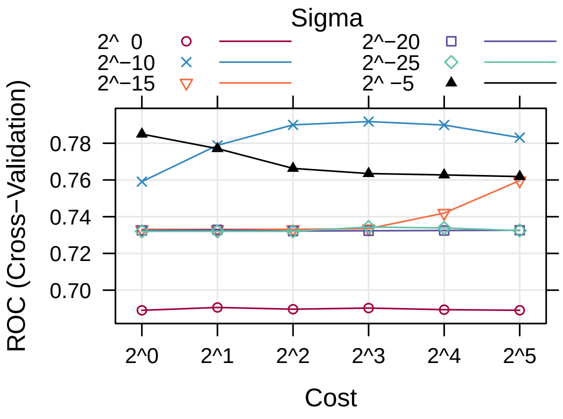


(a) Hyper-parameters grid search

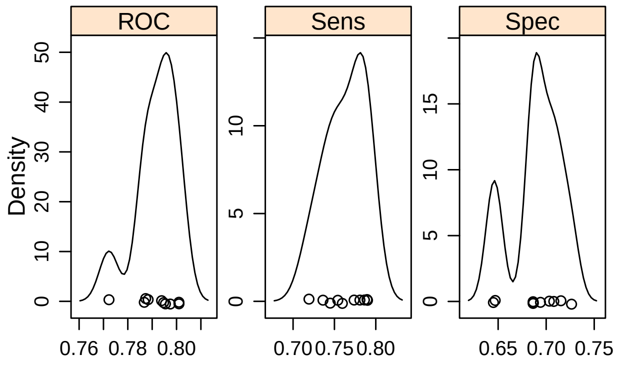


(b) Optimal classifier resampling accuracy

Figure 5.16: SVM model selection using 11 PCA components (Kasir rule)

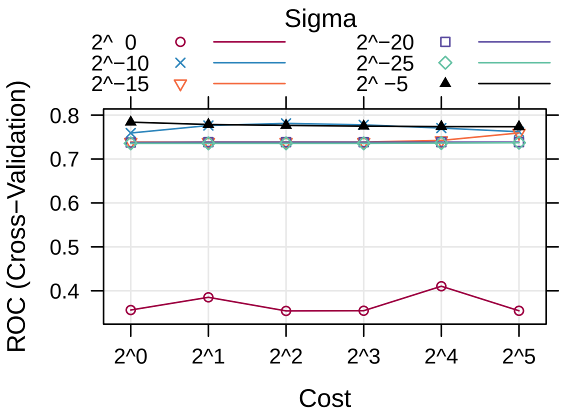


(a) Hyper-parameters grid search

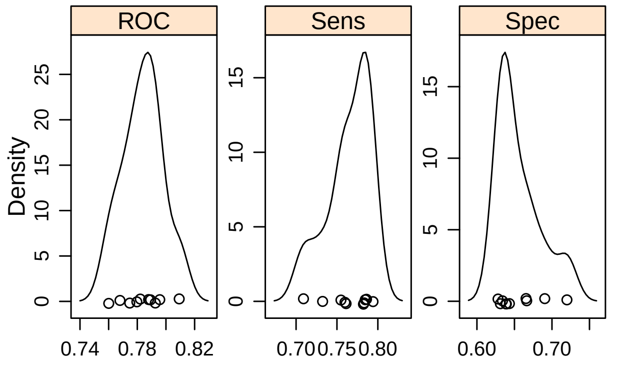


(b) Optimal classifier resampling accuracy

Figure 5.17: SVM model selection using 81 predictors (AUC)



(a) Hyper-parameters grid search



(b) Optimal classifier resampling accuracy

Figure 5.18: SVM model selection using 86 predictors (Relief algorithm)

Table 5.2: SVM selected classifiers

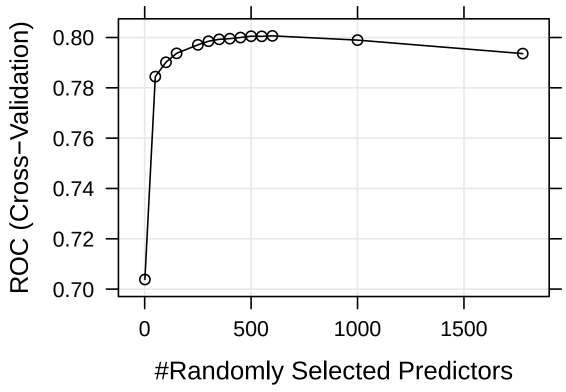
Model	Sigma	Cost	ROC(%)	ROC SD(%)
SVM_ALL	2^{-10}	8	75.73	1.32
SVM_PCA_95	2^{-10}	4	71.49	1.28
SVM_PCA_Kasier	2^{-05}	8	67.54	1.34
SVM_ROC	2^{-10}	8	79.18	0.86
SVM_Relief	2^{-05}	1	78.39	1.43

5.6.3 Random Forest (RF)

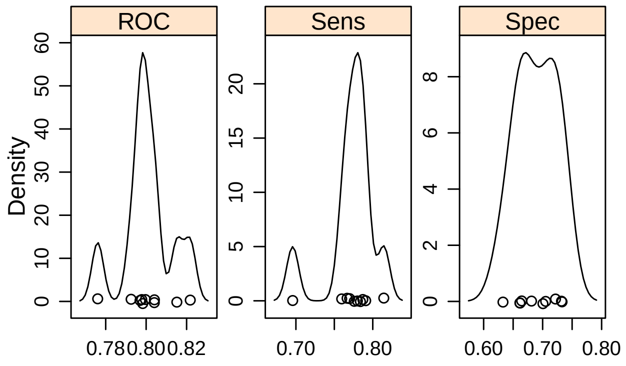
Random Forest [12] has one major tuning hyper-parameter called m_{try}, which is the number of randomly selected predictors to choose from at each split. Breiman [12] recommends setting m_{try} to the square root of the number of predictors. Although the RF model is relatively insensitive to the values of m_{try}, too large values may lead to over-fitting. Therefore, it is advisable to tune m_{try} using the grid-search and cross-validation approach. Kuhn and Johnson [45] recommend starting with values that are evenly spaced across the range from 2 to P (number of predictors in the dataset).

In our case study, the RF's m_{try} is tuned over several values using grid-search and 10-fold CV to select the optimal model. The number of trees per ensemble is kept constant at 5000 trees. Five RF optimal models (one model per feature selection approach) are selected from a total of 555 trained models.

Figure 5.19 to Figure 5.23 present the results of the RF model selection for each feature selection approach.

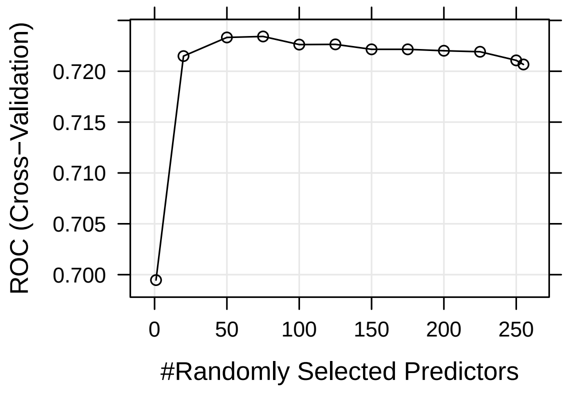


(a) Hyper-parameters grid search

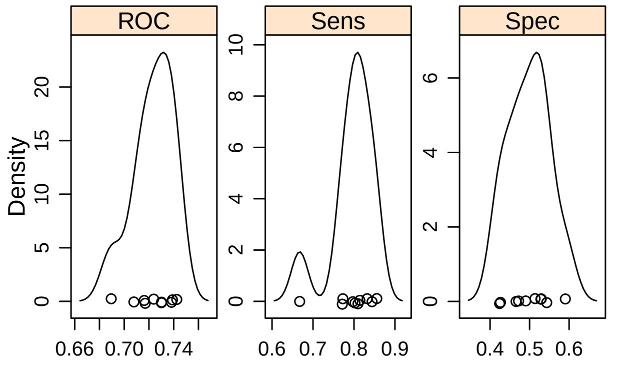


(b) Optimal classifier resampling accuracy

Figure 5.19: Random Forest model selection using all 1776 predictors

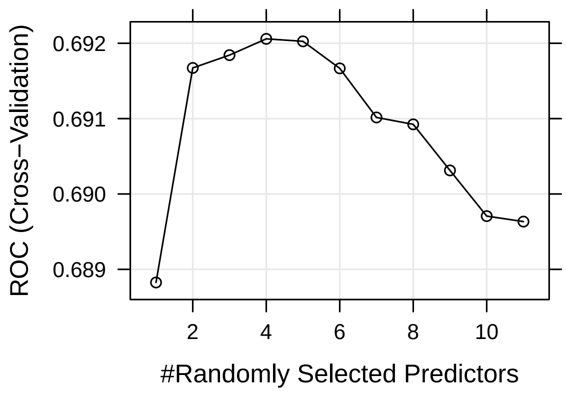


(a) Hyper-parameters grid search

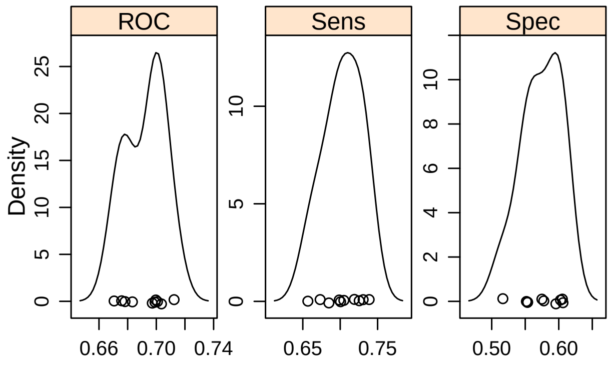


(b) Optimal classifier resampling accuracy

Figure 5.20: Random Forest model selection using 255 PCA components with 95% Var.

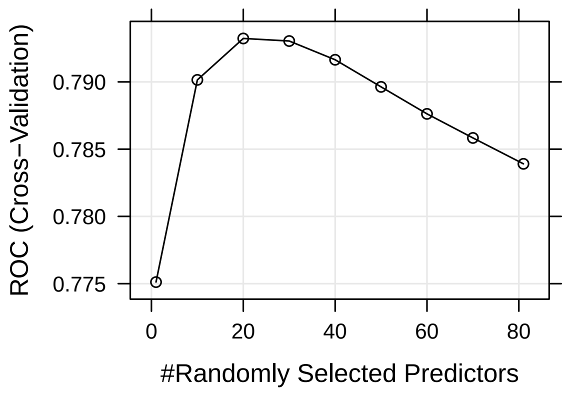


(a) Hyper-parameters grid search

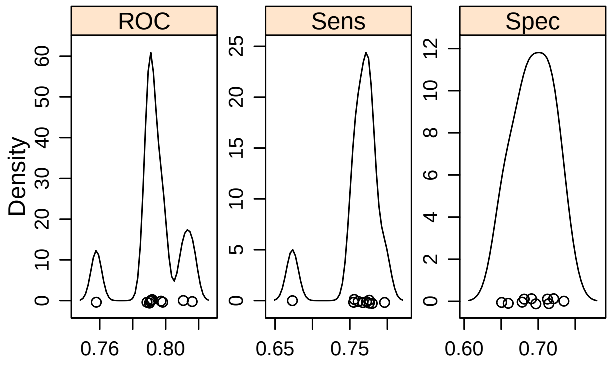


(b) Optimal classifier resampling accuracy

Figure 5.21: Random Forest model selection using 11 PCA components (Kasir rule)

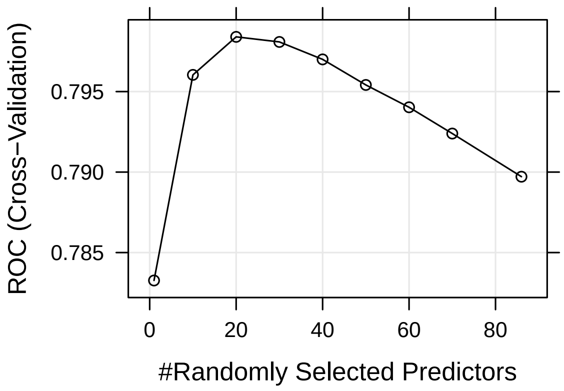


(a) Hyper-parameters grid search

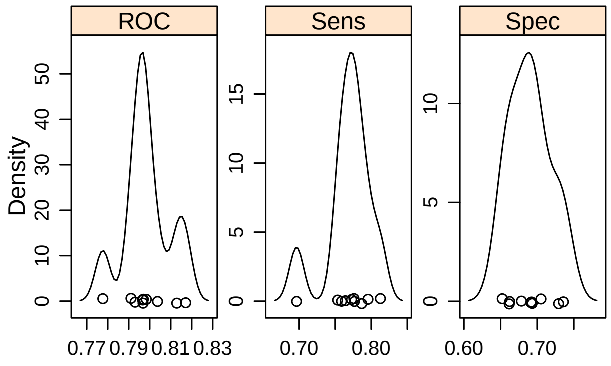


(b) Optimal classifier resampling accuracy

Figure 5.22: Random Forest model selection using 81 predictors (AUC)



(a) Hyper-parameters grid search



(b) Optimal classifier resampling accuracy

Figure 5.23: Random Forest model selection using 86 predictors (Relief algorithm)

Table 5.3: RF selected classifiers

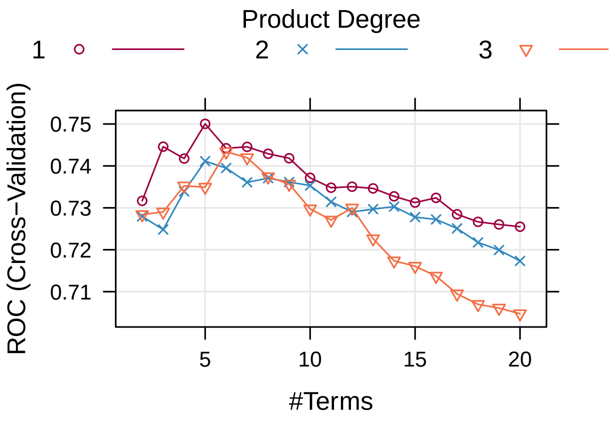
Model	mtry	n.trees	ROC(%)	ROC SD(%)
RF_ALL	600	5000	80.07	1.24
RF_PCA_95	75	5000	72.34	1.63
RF_PCA_Kasier	4	5000	69.21	1.40
RF_ROC	20	5000	79.32	1.55
RF_Relief	20	5000	79.84	1.11

5.6.4 Flexible Discriminate Analysis (FDA)

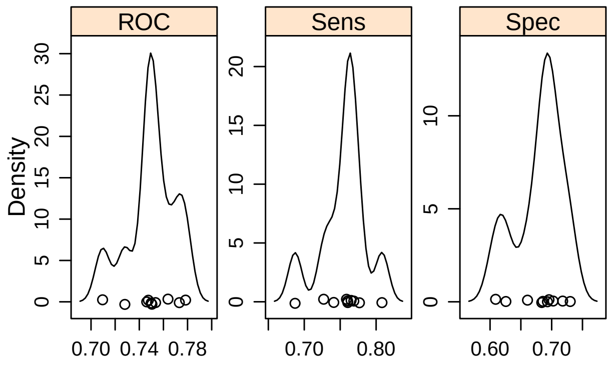
Flexible discriminant analysis [35] using MARS (Multivariate Adaptive Regression Splines) [28] has two hyper-parameters, the number of retained terms and the degree of predictors involved in the hinge functions. As the values of the two parameters increase, the probability of over-fitting increases.

In our case study, the number of retained terms and the degree of predictors is tuned (over nineteen and three values, respectively) using grid-search and 10-fold CV to select the optimal classifier. Five FDA optimal classifiers (one model per feature selection approach) are chosen from a total of 2855 trained models.

Figure 5.24 to Figure 5.28 present the results of the FDA model selection for each feature selection approach.

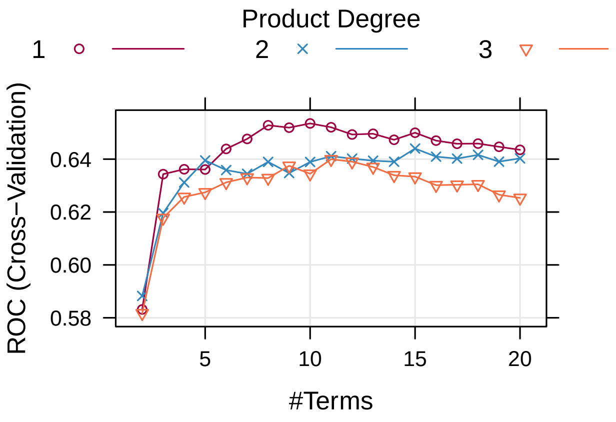


(a) Hyper-parameters grid search

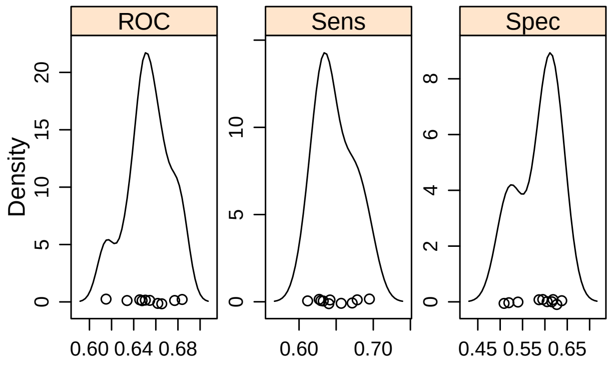


(b) Optimal classifier resampling accuracy

Figure 5.24: FDA model selection using all 1776 predictors

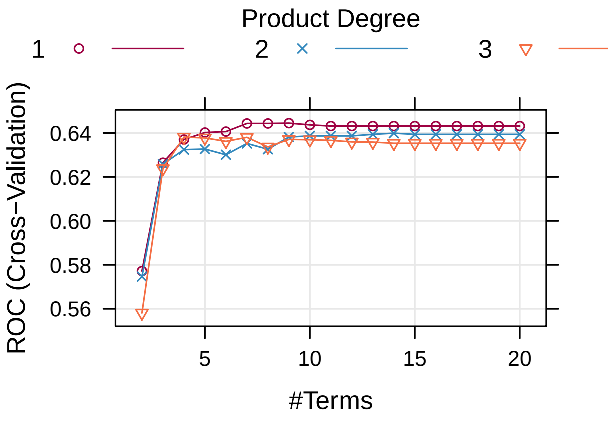


(a) Hyper-parameters grid search

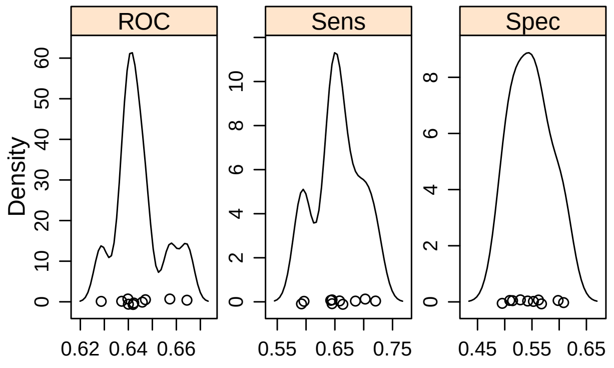


(b) Optimal classifier resampling accuracy

Figure 5.25: FDA model selection using 255 PCA components with 95% Var.

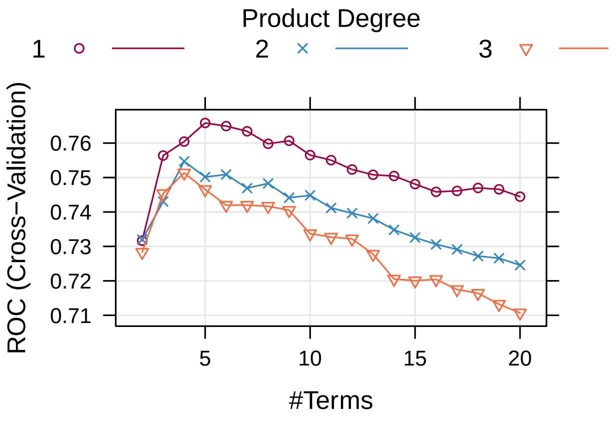


(a) Hyper-parameters grid search

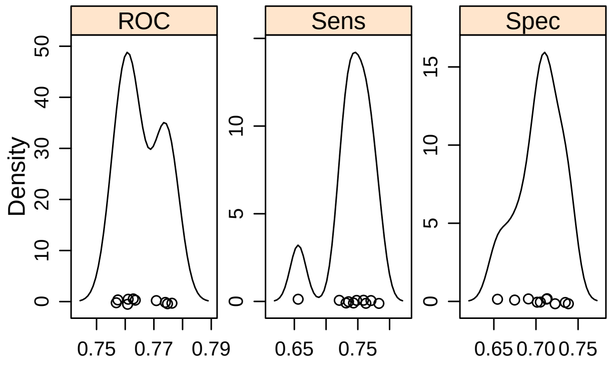


(b) Optimal classifier resampling accuracy

Figure 5.26: FDA model selection using 11 PCA components (Kasir rule)

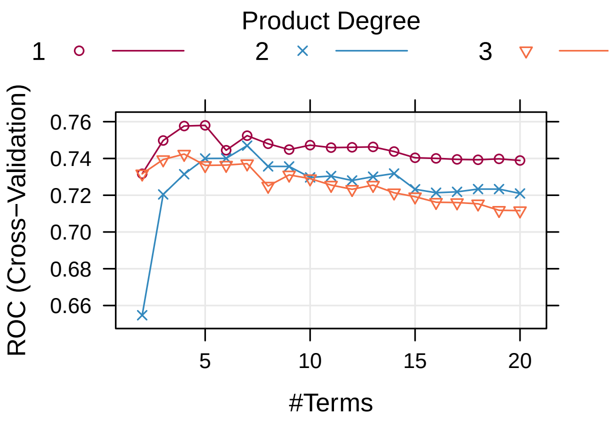


(a) Hyper-parameters grid search

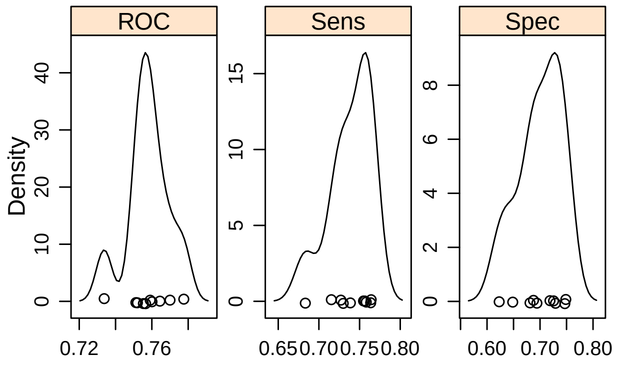


(b) Optimal classifier resampling accuracy

Figure 5.27: FDA model selection using 81 predictors (AUC)



(a) Hyper-parameters grid search



(b) Optimal classifier resampling accuracy

Figure 5.28: FDA model selection using 86 predictors (Relief algorithm)

Table 5.4: FDA selected classifiers

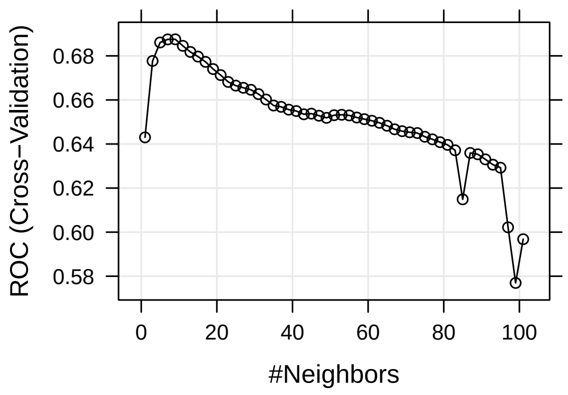
Model	Product.degree	n.terms	ROC(%)	ROC SD(%)
FDA_ALL	1	5	75.00	2.02
FDA_PCA_95	1	10	65.35	2.01
FDA_PCA_Kasier	1	9	64.45	1.01
FDA_ROC	1	5	76.58	0.74
FDA_Relief	1	5	75.80	1.18

5.6.5 K-Nearest Neighbors (KNN)

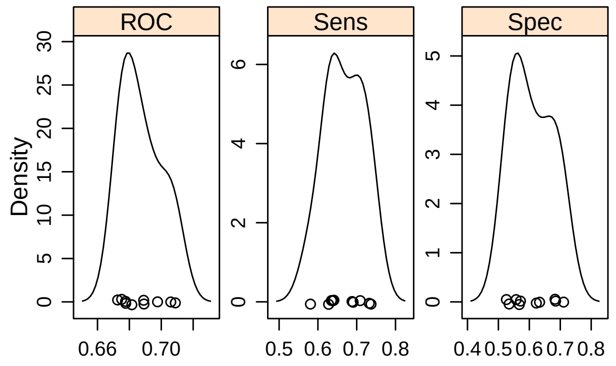
KNN [64] uses a sample's geographic neighborhood to predict the sample's class label. It uses the K-closest samples from the training dataset to predict a new sample. The K-closest training dataset samples are determined via the chosen distance metric. The hyper-parameter for KNN is the number of neighbors (K-closest samples) used to predict a new sample. Too few neighbors lead to over-fitting while too many neighbors result in under-fitting [45].

In our case study, the value of K is tuned over 51 values using grid-search and 10-fold CV to select the optimal classifier. Five KNN optimal classifiers (one classifier per feature selection approach) are selected from a total of 2555 trained classifiers.

Figure 5.29 to Figure 5.33 present the results of the KNN model selection for each feature selection approach.

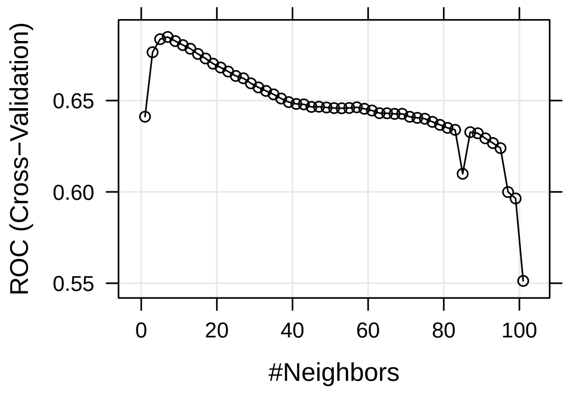


(a) Hyper-parameters grid search

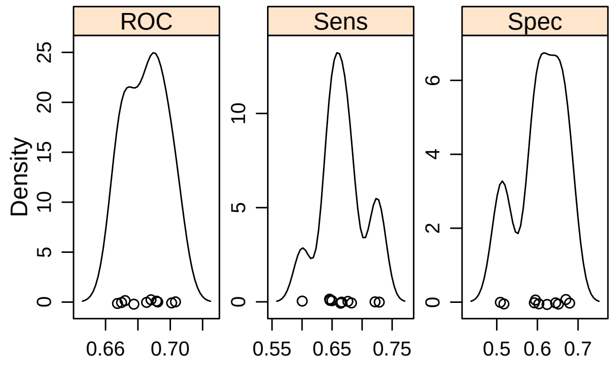


(b) Optimal classifier resampling accuracy

Figure 5.29: KNN model selection using all 1776 predictors

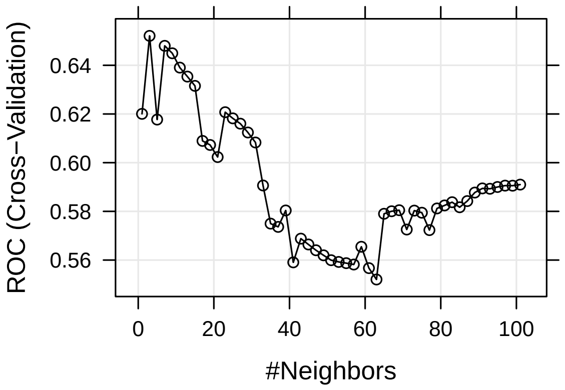


(a) Hyper-parameters grid search

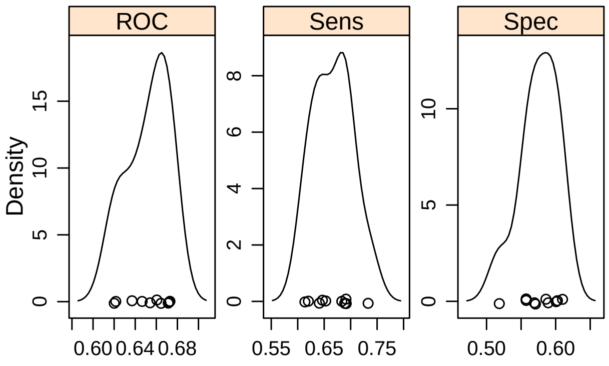


(b) Optimal classifier resampling accuracy

Figure 5.30: KNN model selection using 255 PCA components with 95% Var.

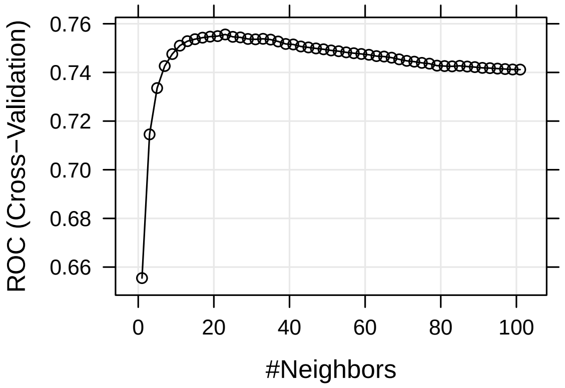


(a) Hyper-parameters grid search

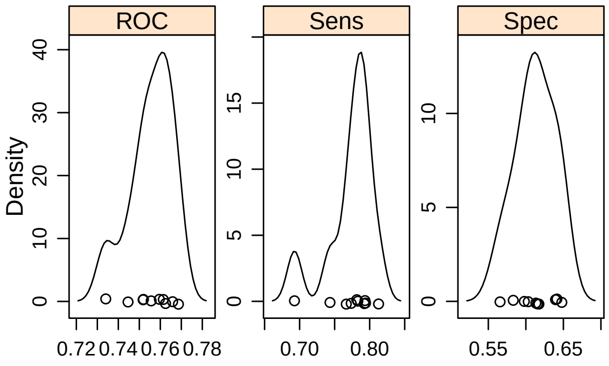


(b) Optimal classifier resampling accuracy

Figure 5.31: KNN model selection using 11 PCA components (Kasir rule)

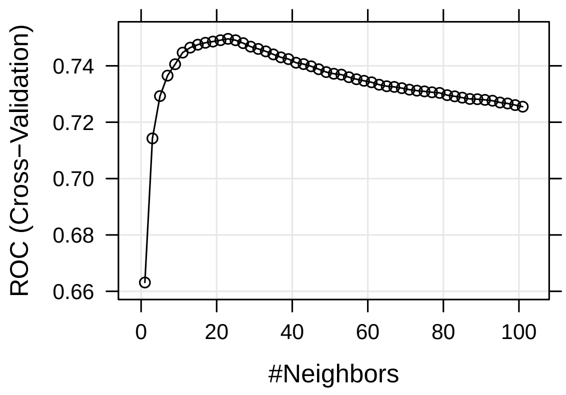


(a) Hyper-parameters grid search

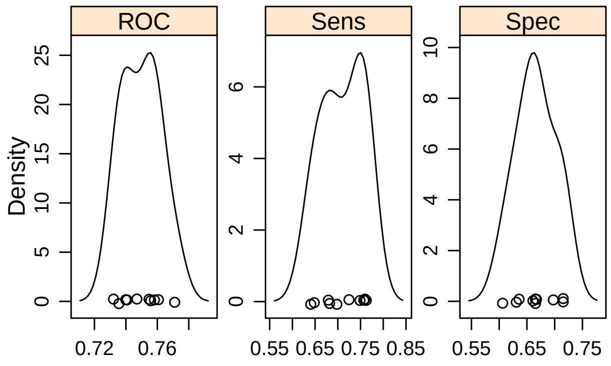


(b) Optimal classifier resampling accuracy

Figure 5.32: KNN model selection using 81 predictors (AUC)



(a) Hyper-parameters grid search



(b) Optimal classifier resampling accuracy

Figure 5.33: KNN model selection using 86 predictors (Relief algorithm)

Table 5.5: KNN selected classifiers

Model	n.neighbors	ROC(%)	ROC SD(%)
KNN_ALL	9	68.75	1.29
KNN_PCA_95	7	68.48	1.27
KNN_PCA_Kasier	3	65.21	2.02
KNN_ROC	23	75.56	1.05
KNN_Relief	23	74.96	1.25

5.7 Performance Evaluation

In this section, different metrics and approaches are employed to evaluate the classification performance of the twenty-five classifiers, developed in the previous section. The choice of a particular metric/method depends on the application and the designer's objectives. However, it is imperative that the designer of the ensemble system uses different metrics/approaches to evaluate the individual and relative performance of the base classifiers before combining them. As discussed in Section 3.4.3, the training data (used to train the classifiers) should never be used to evaluate the classification performance. Instead, the out-of-fold cross-validation and/or testing data (new data, never seen by the classifiers) should be employed. The 10-fold cross-validation (out-of-fold) and a testing dataset (with a number of observations $n = 1125$) are used to evaluate different aspects of classifiers' predictions using statistics and visualizations. As additional objectives, this section aims to investigate:

- **Algorithms scalability**: Experimentally evaluates the effect of "the curse of dimensionality" on the model selection, model performance, and computation cost of several learning algorithms.
- **Comparing feature selection algorithms**: Compares the performance of models built using several feature selection algorithms across different learning algorithms.
- **Learning algorithms' relative performance**: Compares the performance of several learning algorithms for a high-dimensional classification problem.

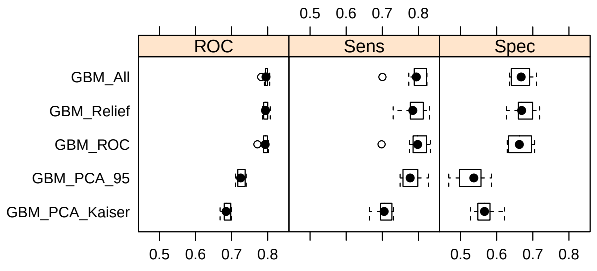
5.7.1 10-fold Cross-validation Resampling

In Section 5.6, the 10-fold cross-validation resampling is applied to the training dataset to tune classifiers' hyper-parameters and select the optimal models. During model selection, the classifier is trained on nine folds and tested on the 10th fold. This process is repeated ten times as previously explained in Section 3.4.2. So, for each optimal classifier, there are ten out-of-folds predictions. In this section, the out-of-folds performance (area under ROC, Sensitivity (Sens.), and Specificity (Spec.)) are compared based on three approaches. In the first approach, the performance of classifiers is compared per modeling algorithm. The second approach compares classifiers per feature selection approach. Finally, in the third approach all twenty-five classifiers' performances are compared. The performance metrics are compared in two categories: the metrics' distribution (using the box-whisker plot), and metrics' estimates with 95% Confidence Level (CL).

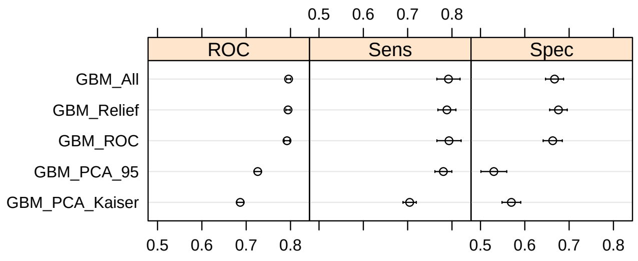
In the context of building ensemble systems, this comparison reveals the effect of injecting diversity at feature/structural and learning algorithm levels on the base classifiers' performance. Using out-of-folds cross-validation often gives robust estimates of classifiers' performance, especially in the absence of a large testing dataset. However, as discussed in Section 3.4.2, in order to compare apples-to-apples, the same folds should be used for training and validation across all learning algorithms. The stratified sampling should be used to generate the CV folds to maintain similar class's distribution across all folds. Besides, care should be taken to minimize the effect of randomness in the learning algorithm and distributed computing by properly setting the seed of the random numbers generator.

Comparing Classifiers per Learning Algorithm

The classifiers built using the same learning algorithm are grouped together, and their performances are compared. Since five learning algorithms (GBM, SVM, RF, FDA, and KNN) are used in this case study, there are five groups (one for each learning algorithm). Within each group, there are five classifiers (one for each feature selection approach). The objective here is to compare the relative performance of classifiers built using the same learning algorithm and cross-validation folds but different feature selection algorithms.



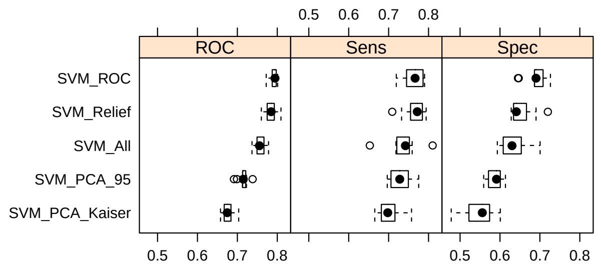
(a) Performance's distributions



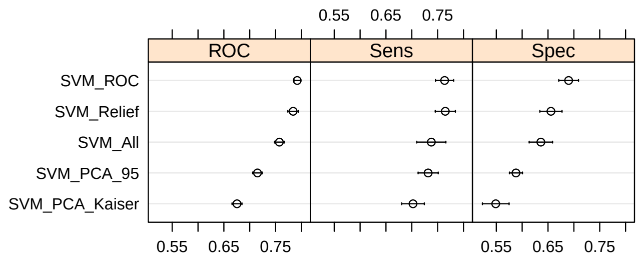
(b) Estimates with 95% CL

Figure 5.34: GBM classifiers' resampling performance

Figure 5.34, compares the performance of the five GBM classifiers. The performance's distributions of classifiers, shown in Figure 5.34a, are ordered by the median of the area under ROC, while the performance's estimates with 95% CL of classifiers, shown in Figure 5.34b, are ordered by the average of the area under ROC. The GBM classifiers built using all predictors (1776) (*GBM_All*), the Relief feature selection (*GBM_Relief*), and the area under ROC feature selection (*GBM_ROC*) algorithms produce similar ROC distribution, but slightly different sensitivity and specificity distributions. It should be noted that the high-dimensional feature space has no effect on the performance of the GBM learning algorithm (the *GBM_All* is the best performing classifier) because it has a built-in feature selection algorithm. The classifiers built using PCA (*GBM_PCA_95* and *GBM_PCA_Kaiser*) produce the worst performance.



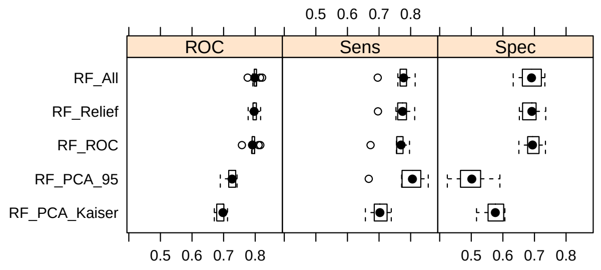
(a) Performance's distributions



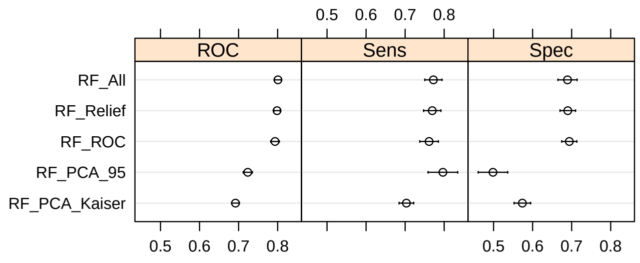
(b) Estimates with 95% CL

Figure 5.35: SVM classifiers' resampling performance

The performance of the five SVM classifiers are compared in Figure 5.35. The SVM classifiers built using the area under ROC feature selection (SVM_ROC), and the Relief feature selection (SVM_Relief) algorithms produce similar ROC distributions, but different sensitivity and specificity distributions. The SVM classifier built using all predictors (1776) (SVM_All) comes in the third place, which suggests that the performance of the SVM learning algorithms is negatively affected by the high-dimensional feature space. The classifiers built using PCA (GBM_PCA_95 and GBM_PCA_Kaiser) produce the worst performance.



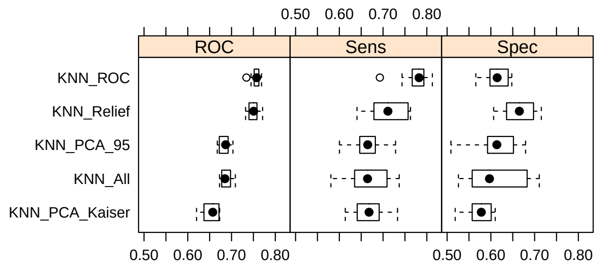
(a) Performance's distributions



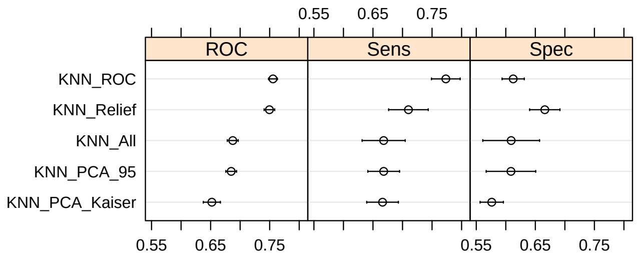
(b) Estimates with 95% CL

Figure 5.36: RF classifiers' resampling performance

Figure 5.36, compares the performance of the five RF classifiers. The RF classifiers built using all predictors (1776) (RF_All), the Relief feature selection (RF_Relief), and the area under ROC feature selection (RF_ROC) algorithms produce similar ROC distribution, but slightly different sensitivity and specificity distributions. Analogous to the GBM learning algorithm, the RF algorithm has a built-in feature selection algorithm, which eliminates the effect of the high-dimensional feature space on the classifier's performance (the *RF_All* is the best performing classifier). The classifiers built using PCA (*RF_PCA_*95 and *RF_PCA_Kaiser*) produce the worst performance.



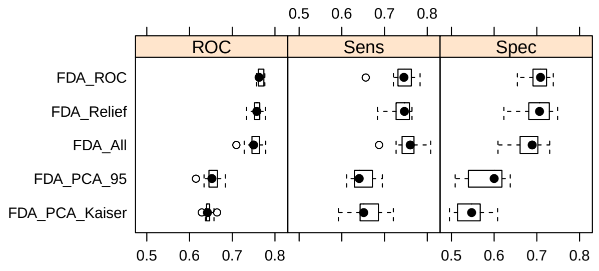
(a) Performance's distributions



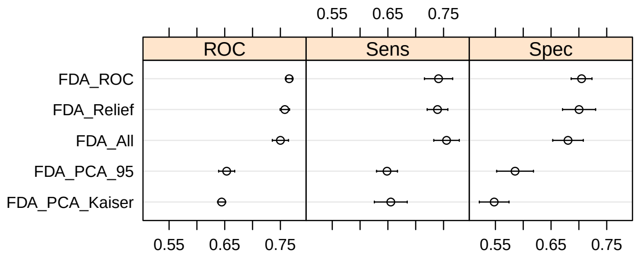
(b) Estimates with 95% CL

Figure 5.37: KNN classifiers' resampling performance

The performance of the five KNN classifiers are compared in Figure 5.37. The KNN classifiers built using the area under ROC feature selection (*KNN_ROC*), and the Relief feature selection (*KNN_Relief*) algorithms produce similar ROC distributions, but different sensitivity and specificity distributions. The KNN classifier built using all predictors (*KNN_All*) comes in the third place, which suggests that the performance of the KNN learning algorithms is negatively affected by the high-dimensional feature space. The classifiers built using PCA (*KNN_PCA_*95 and *KNN_PCA_Kaiser*) produce the worst performance. The results show that KNN classifiers have a high variance of performance.



(a) Performance's distributions



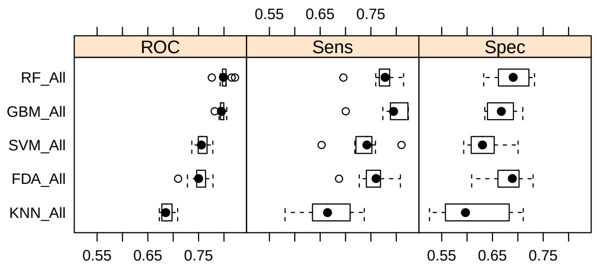
(b) Estimates with 95% CL

Figure 5.38: FDA classifiers' resampling performance

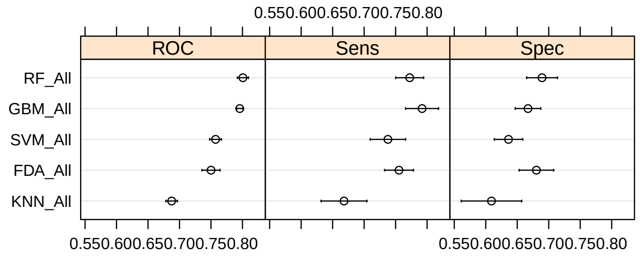
Finally, Figure 5.38, compares the performance of the five FDA classifiers. The FDA classifiers built using the area under ROC feature selection (*FDA_ROC*) algorithm, the Relief feature selection (*FDA_Relief*) algorithm, and all predictors (*FDA_All*) produce similar ROC distributions, but different sensitivity and specificity distributions. Since the FDA algorithm has a built-in feature selection algorithm, the high-dimensional feature space has a slight effect on its performance. The classifiers built using PCA (FDA_PCA_95 and FDA_PCA_Kaiser) produce the worst performance.

Comparing Classifiers per Feature Selection Algorithm

The classifiers built using the same feature selection algorithm are grouped together and their performances are compared. Since five feature selection algorithms/approaches (all features, PCA 95%, PCA Kaiser, area under ROC, and Relief) are used in this case study, there are five groups (one for each feature selection algorithm). Within each group, there are five classifiers (one for each learning algorithm). The objective here is to compare the relative performance of the five learning algorithms (GBM, SVM, RF, FDA, and KNN), given the exact same features and cross-validation folds.



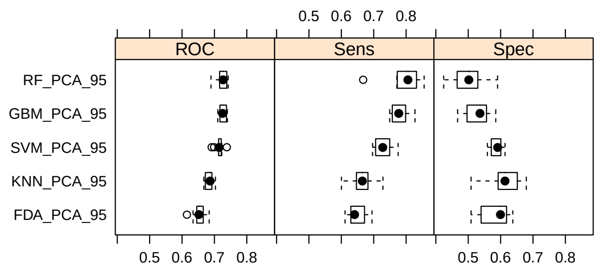
(a) Performance's distributions



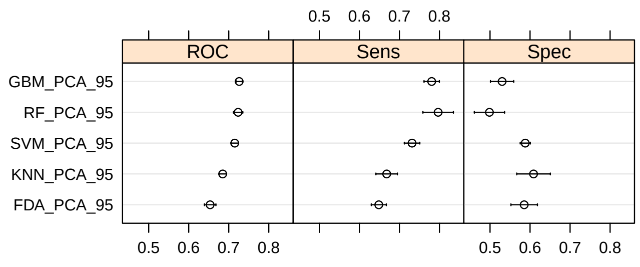
(b) Estimates with 95% CL

Figure 5.39: All predictors classifiers' resampling performance

The performances of the five classifiers built using all features (1776 feature) are compared in Figure 5.39. The performance's distributions of classifiers, shown in Figure 5.39a, are ordered by the median of the area under the ROC while the performance's estimates with 95% CL of classifiers, shown in Figure 5.39b are ordered by the average of the area under ROC. The *RF_All* classifier has a slight edge (in terms of ROC and specificity) over the *GBM_All* classifier, which comes in the second place. However, the *GBM_All* has slightly better sensitivity. As noted in the previous comparisons, the performance of the RF and GBM learning algorithms is not affected by the high-dimensionality because they have built-in feature selection algorithms. The *SVM_All* and *FDA_All* classifiers come respectively in the third and fourth positions. The *KNN_All* classifier has the worst performance. It should be noted that the order of classifiers based on their area under ROC does not agree with their order based on sensitivity and specificity.



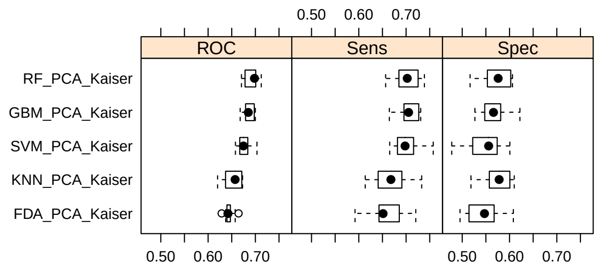
(a) Performance's distributions



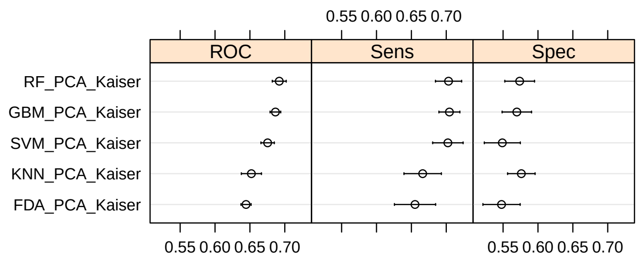
(b) Estimates with 95% CL

Figure 5.40: PCA 95% classifiers' (255 PCA) resampling performance

Figure 5.40 compares the performances of classifiers built using the PCA with 95% variance (255 PCA components). Based on the average area under ROC, the *GBM_PCA_95* classifier outperforms all classifiers. The *RF_PCA_95* classifier comes in the second place, and the classifiers *SVM_PCA_95*, *KNN_PCA_95*, and *FDA_PCA_95* follow respectively in the third, fourth, and fifth places. The order of classifiers based on the sensitivity agrees with the order based on the area under ROC except for the first two positions, where the *RF_PCA_95* classifier slightly outperforms the *GBM_PCA_95* classifier. On the other hand, the order of classifiers based on the specificity is reversed.



(a) Performance's distributions

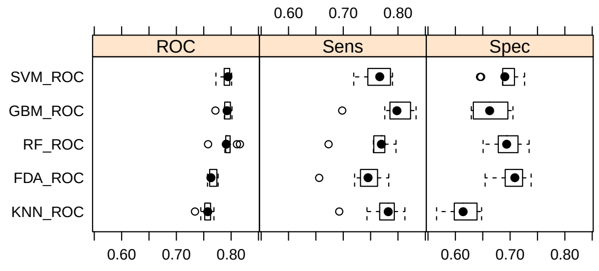


(b) Estimates with 95% CL

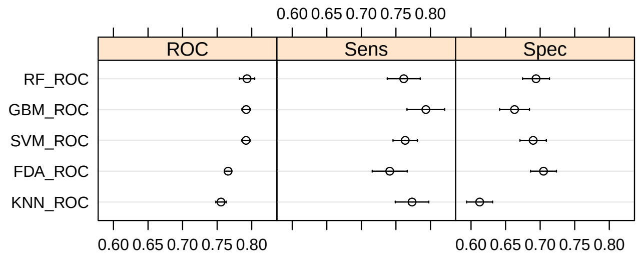
Figure 5.41: Kaiser classifiers' (11 PCA) resampling performance

The performances of the classifiers built using the PCA Kaiser rule (11 PCA components) are shown in Figure 5.41. The order of classifiers based on the average area under ROC is respectively as follows: *RF_PCA_kaiser*, *GBM_PCA_kaiser*, *SVM_PCA_kaiser*,

KNN_PCA_kaiser, and FDA_PCA_kaiser. This order agrees with the classifiers' order based on the sensitivity while the order based on the specificity slightly disagrees (the specificity of KNN_PCA_kaiser is better than SVM_PCA_kaiser).



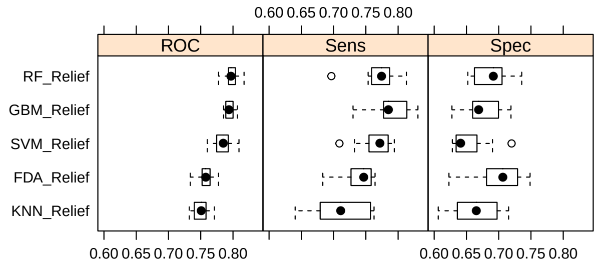
(a) Performance's distributions



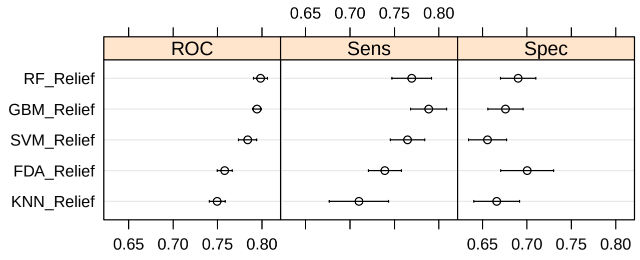
(b) Estimates with 95% CL

Figure 5.42: ROC classifiers' resampling performance

Figure 5.42 compares the performances of classifiers built using the area under ROC feature selection algorithm. Based on the average area under ROC, the order of classifiers is respectively RF_ROC, GBM_ROC, SVM_ROC, FDA_ROC, and KNN_ROC. However, this order substantially disagrees with the orders based on the sensitivity and specificity.



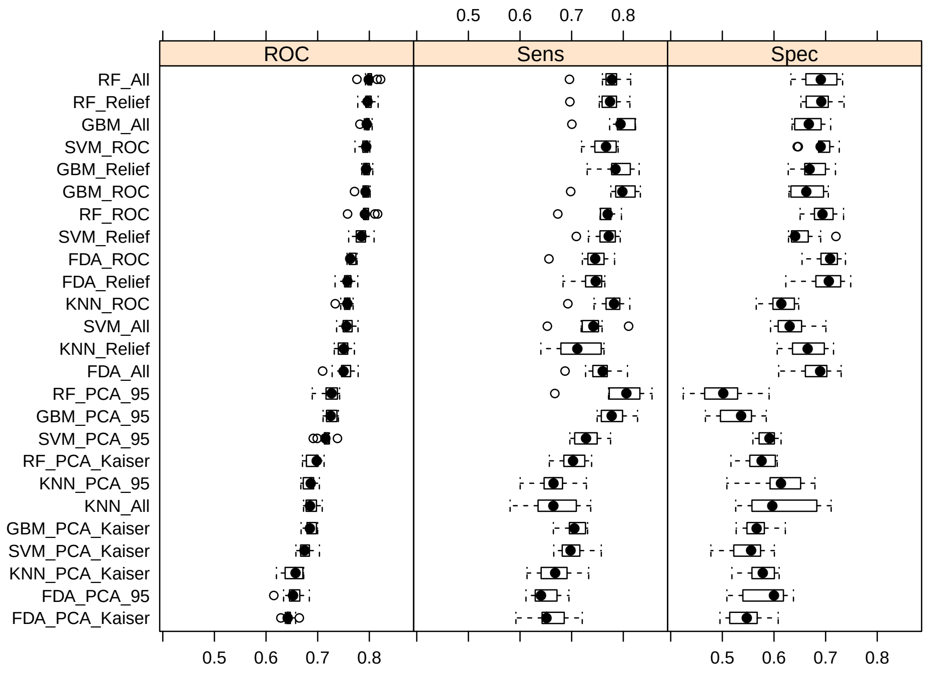
(a) Performance's distributions



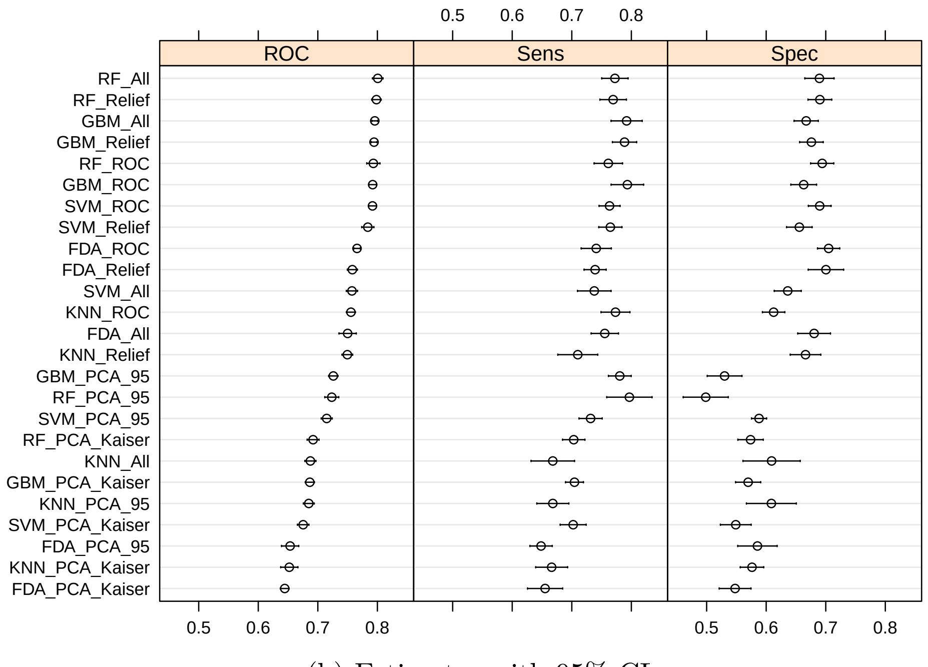
(b) Estimates with 95% CL

Figure 5.43: Relief classifiers' resampling performance

The performances of classifiers built using the Relief feature selection algorithm are compared in Figure 5.43. The order of classifiers, based on the average area under ROC, is respectively RF_Relief, GBM_Relief, SVM_Relief, FDA_Relief, and KNN_Relief. The classifiers' order based on the sensitivity and specificity greatly agrees with their order based on the average area under ROC.



(a) Performance's distributions



(b) Estimates with 95% CL

Figure 5.44: All 25 classifiers' resampling performance

Finally, Figure 5.44 compares the performances of all twenty-five classifiers. The comparison shows that injecting model diversity at the feature, structural, and learning algorithm levels produces a diverse area under ROC, sensitivity, and specificity.

5.7.2 Evaluating Class Probabilities

Using the test data set (with 1125 observations) and several types of plots, this section evaluates the class probabilities produced by the trained classifiers. These plots include: histograms of class probabilities, calibration plot, lift charts, and the Receiver Operating Characteristics (ROC) curves plot.

Histograms of Class Probabilities

For two classes, histograms of the predicted classes for each of the true outcomes reveals the strengths and weaknesses of a classifier. Figure 5.45 shows histograms of the test set class's probabilities for the SVM_Relief (left plot) and FDA_PCA_95 (right plot) classifiers (the panels indicate the true Active/Inactive status). Ideally, the probability of active (left panel) for molecules with an active response should have a left skewed distribution while the probability of inactive for molecules with an inactive response should have a right skewed distribution. In contrast, if the probabilities are flat (or uniformly distributed), as for the FDA_PCA_95, this reflects the classifier's inability to distinguish active/inactive molecules.

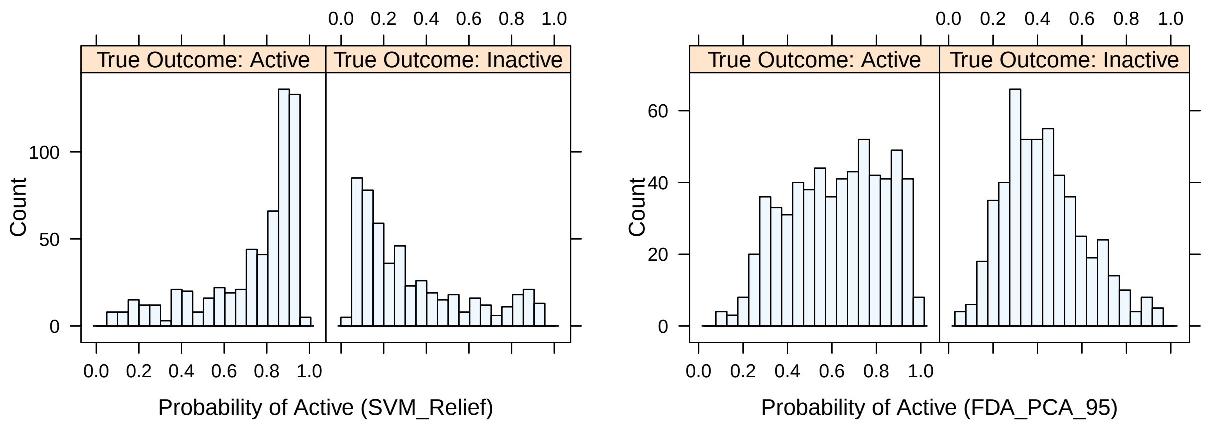


Figure 5.45: Histograms of class probabilities

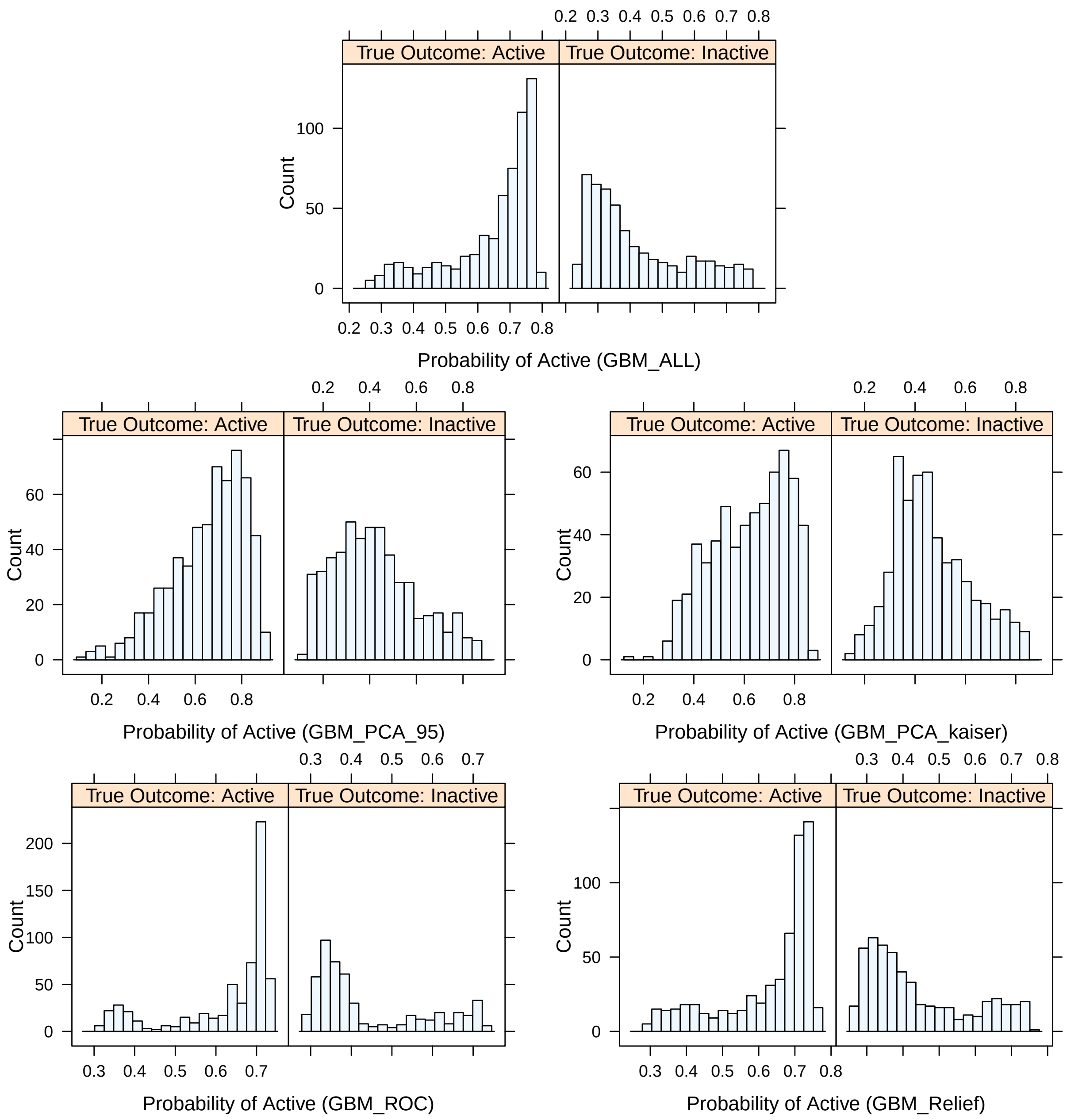


Figure 5.46: GBM classifiers' histograms of class probabilities

Figures 5.46 to 5.50 present the histograms of class probabilities for the twenty-five classifiers grouped by the learning algorithm. Most GBM, RF, SVM, and FDA classifiers have good skewed distributions of class probabilities except the classifiers built using the PCA features. On the other hand, all KNN classifiers have near flat/uniform distributions.

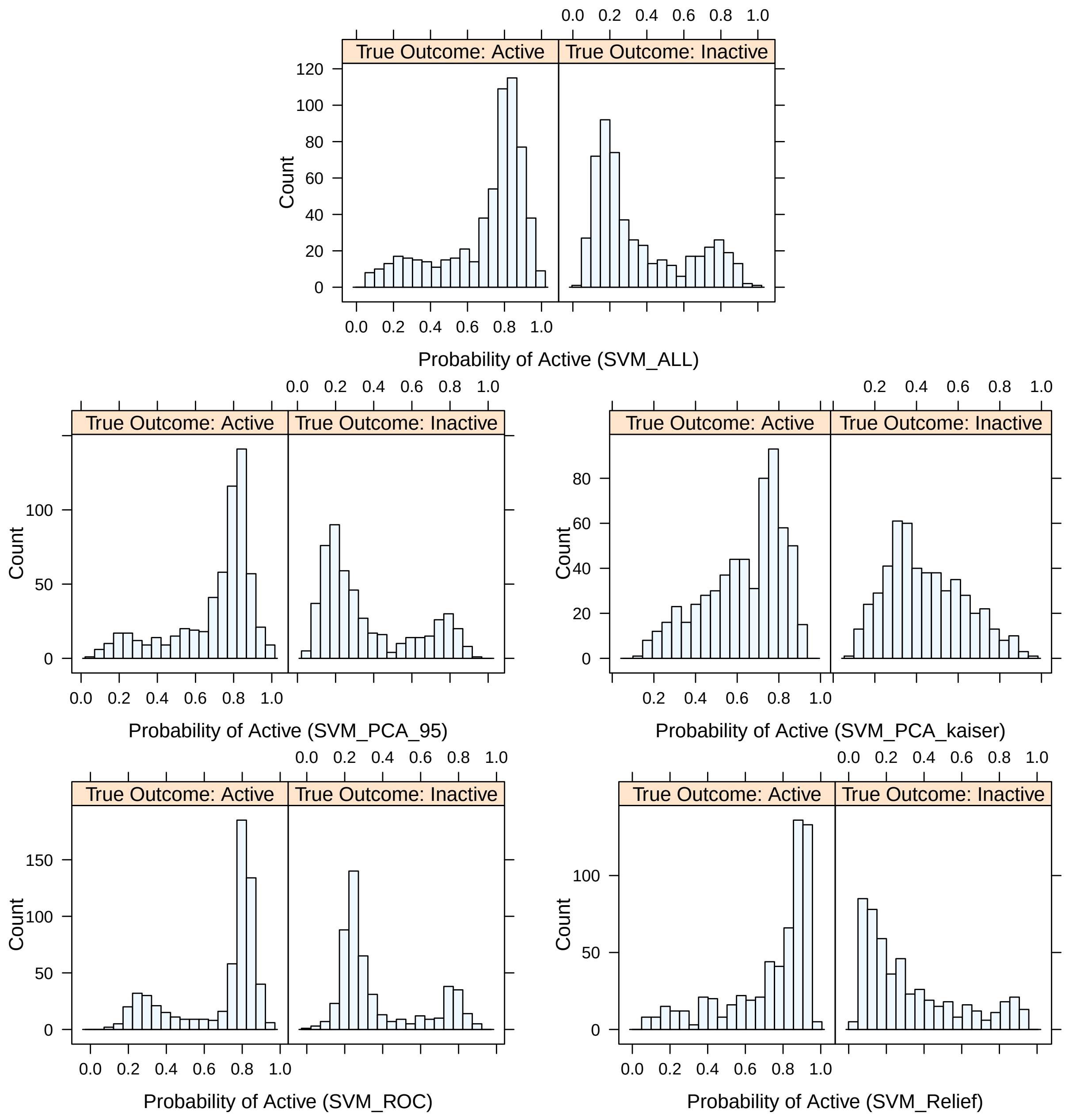


Figure 5.47: SVM classifiers' histograms of class probabilities

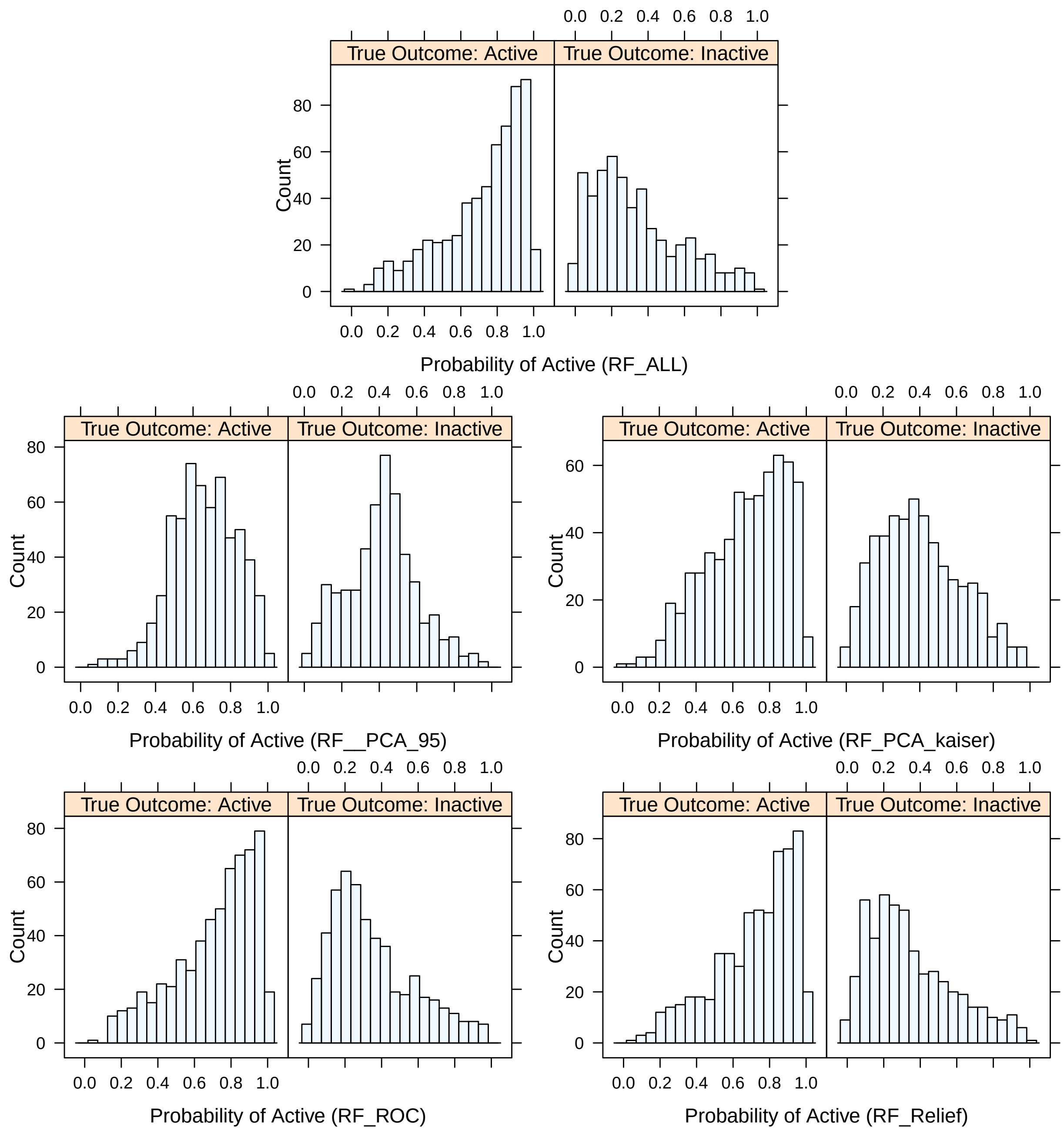


Figure 5.48: RF classifiers' histograms of class probabilities

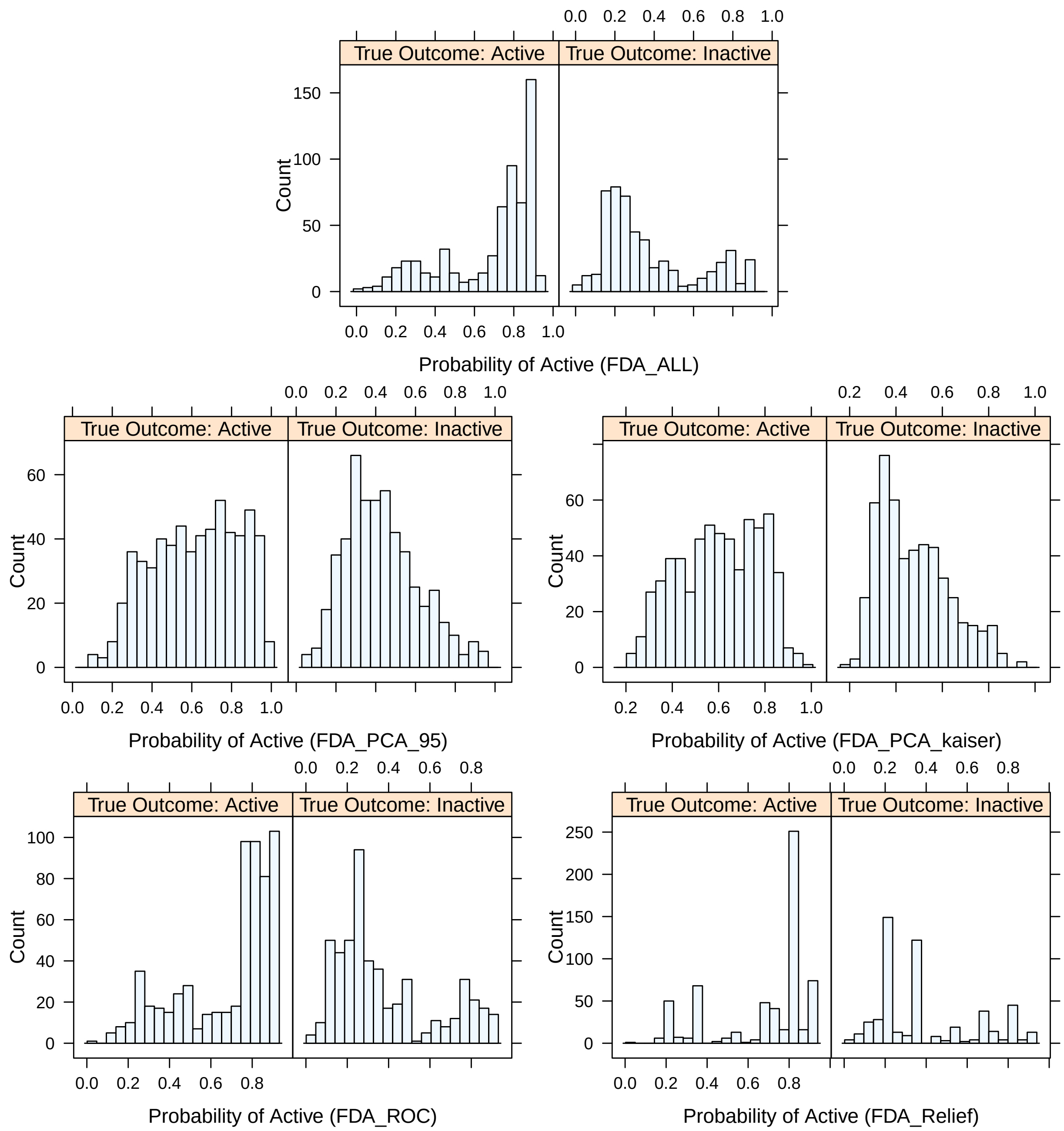


Figure 5.49: FDA classifiers' histograms of class probabilities

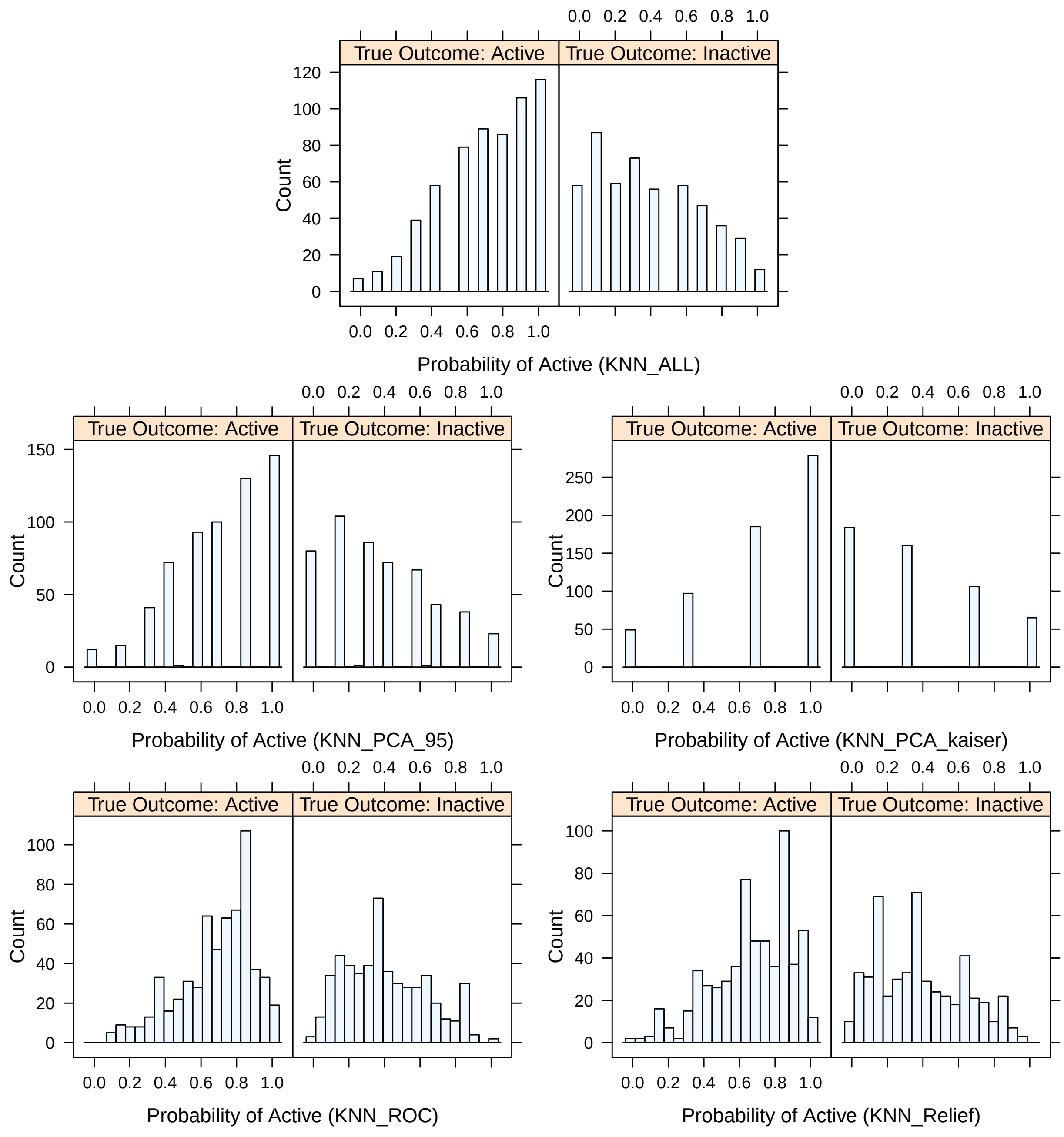
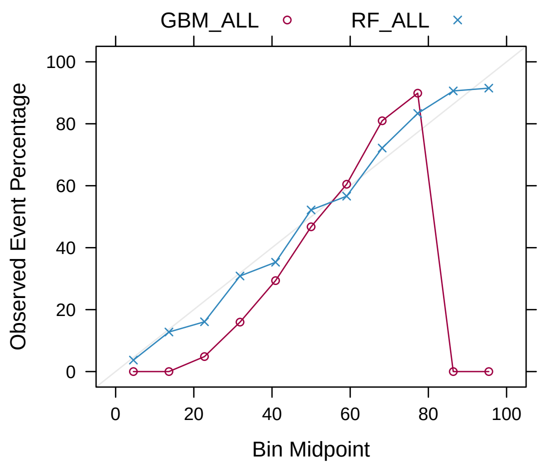


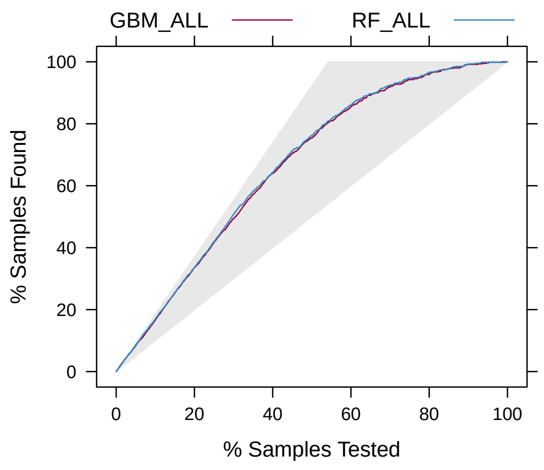
Figure 5.50: KNN classifiers' histograms of class probabilities

Calibration and Lift Plots

Regardless of the application's domain, one desirable feature of a classifier is that the predicted class probabilities are well-calibrated. That is, they must reflect the true likelihood of the event of interest in a given sample. The calibration plot is one approach to assessing the quality of the predicted class probabilities. It measures the observed probability of an event versus the predicted class probability. The calibration plot is constructed by applying the classifier to the test data set. Next, the data is binned into groups based on their class probabilities (e.g. [0, 10%], (10%, 20%],..., (90%, 100%]). Then, for each bin, the observed event rate is determined. The classifier's predicted probabilities are well-calibrated if the points fall along the 45° line [45]. For example, Figure 5.51a shows the calibration plots for the *GBM_ALL* and *RF_ALL* classifiers. The *RF_ALL* produces good-calibrated probabilities. On the other hand, the *GBM_ALL* classifier produces poorly calibrated probabilities, where no samples with active response were predicted with a probability above 80%.



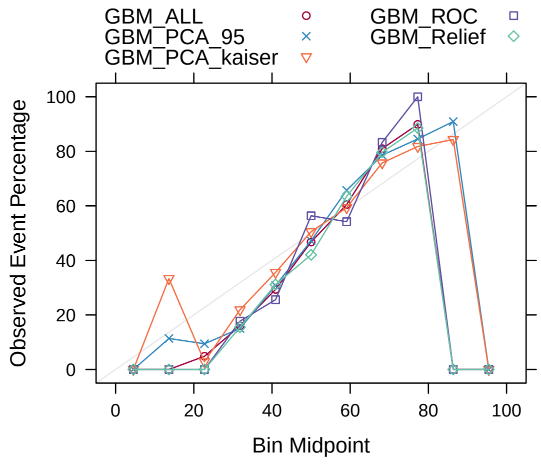
(a) Calibration plot



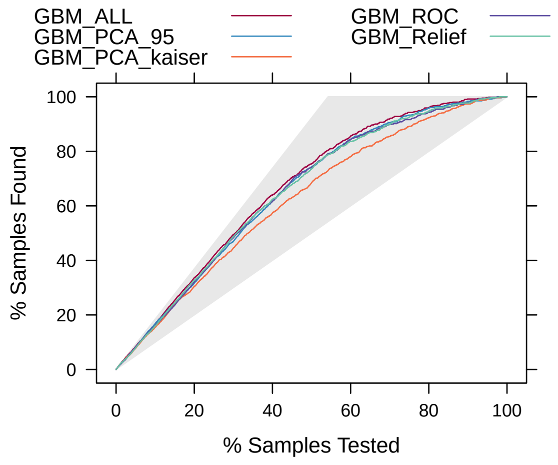
(b) Lift plot

Figure 5.51: Calibration and Lift plots

Lift charts [53] are used to assess the ability of a classifier to detect the events in a binary classification problem. The lift charts rank the samples by their event class probability and determine the cumulative event rate as more samples are evaluated. Optimally, the N highest-ranked samples contain all N events in the data set. The number of samples detected by a classifier above an entirely random selection of samples is called the lift. So, the lift chart plots the cumulative lift against the cumulative percentage of the samples tested. Figure 5.51b shows the lift charts for the *GBM_ALL* and *RF_ALL* classifiers. The left top corner of the triangle represents the event rate in the test data set (in our case, the event (active) rate is 54.2%). A perfect classifier would have a curve that goes into the top left corner of the triangle, which indicates that all of the events have been captured by the classifier. On the other hand, a non-informative classifier would have a curve close to the 45° line. Lift charts can be used to compare the performance of different classifiers [45].

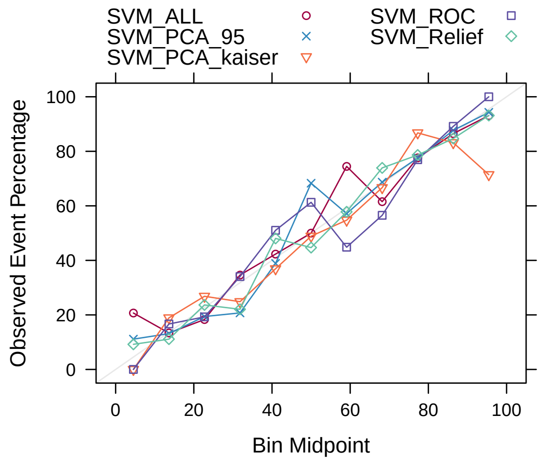


(a) Calibration plot

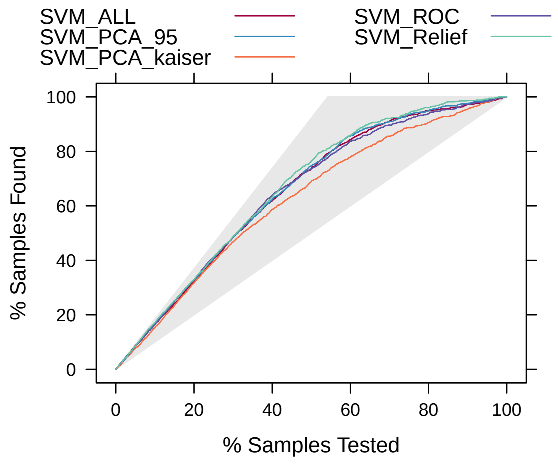


(b) Lift plot

Figure 5.52: Calibration and lift plots of the GBM classifiers

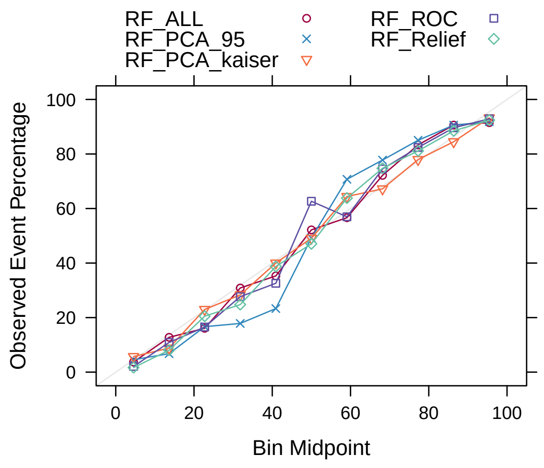


(a) Calibration plot

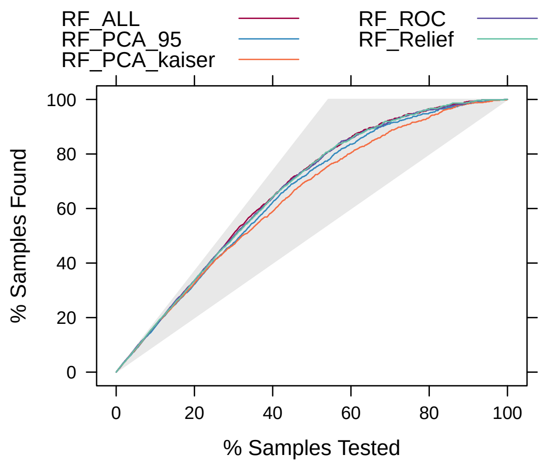


(b) Lift plot

Figure 5.53: Calibration and lift plots of the SVM classifiers

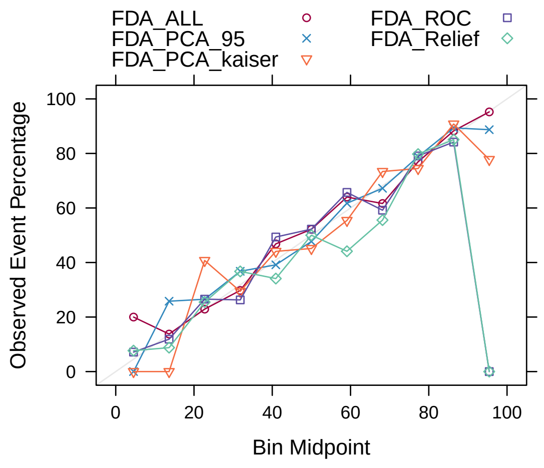


(a) Calibration plot

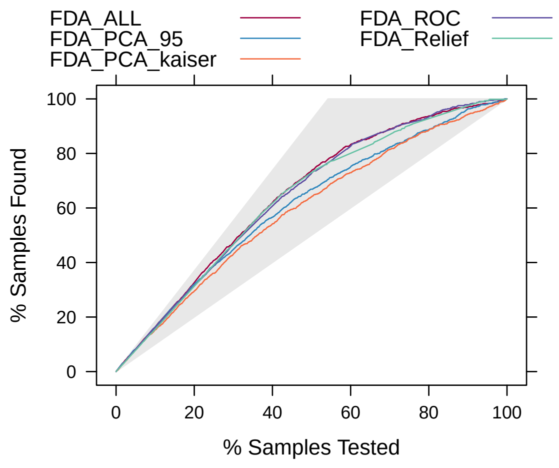


(b) Lift plot

Figure 5.54: Calibration and lift plots of the RF classifiers

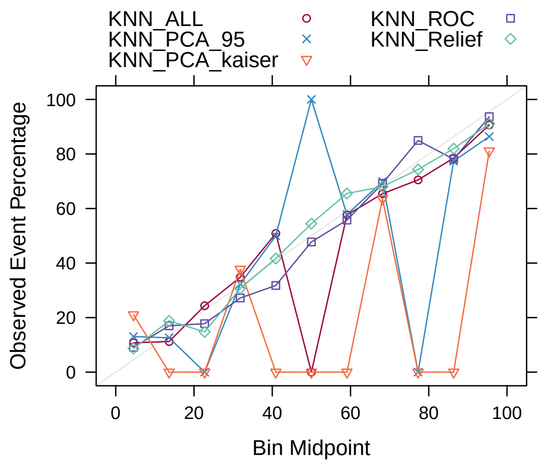


(a) Calibration plot

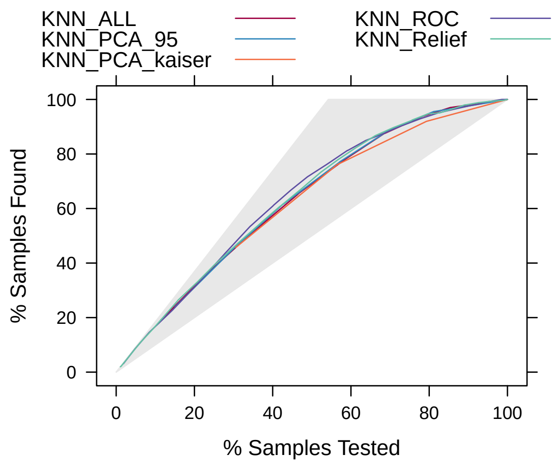


(b) Lift plot

Figure 5.55: Calibration and lift plots of the FDA classifiers



(a) Calibration plot



(b) Lift plot

Figure 5.56: Calibration and lift plots of the KNN classifiers

Figures 5.52 to 5.56 present the calibration and lift plots for the twenty-five classifiers grouped by the learning algorithm. Most SVM, RF, and FDA classifiers have good calibration of class probabilities while GBM and KNN classifiers have poorly-calibrated probabilities. The class probabilities produced by a classifier can be re-calibrated, to reflect the likelihood of the event seen in the data set, by training an additional classifier (such as Logistic Regression or Naive Bayes) to adjust the predicted probabilities [45]. Having well-calibrated class probabilities might prove to be worthy of consideration, especially when using these probabilities as predictors in the model fusion by model stacking, discussed in Section 5.8.3. However, for this case study, the predicted probabilities will be used in the model fusion without re-calibrating. Investigating the effect of re-calibrating class probabilities on the overall performance of ensembles is considered a future work.

The ranks of classifiers (per learning algorithm) based on the lift charts are consistent with the ranks in Section 5.7.1 (based on the average CV area under ROC).

Receiver Operating Characteristic (ROC) Curves

The ROC curve is a common quantitative measure of classifiers' performance in binary classification problems. It can also be used to compare visually the performance of different classifiers (e.g. classifiers built using the same learning algorithm, but different feature selection algorithms and/or different hyper-parameters).

Figure 5.57 shows an example of a ROC curve for the *GBM_ALL* classifier. A perfect classifier would have 100% sensitivity and specificity (defined in Section 5.7.3). The ROC curve for such a classifier would be a single step between (0,0) and (0,1) and remain constant from (0, 1) to (1, 1), and its Area Under ROC Curve (AUC) would be 1.0. On the other hand, an entirely ineffective classifier would have an ROC curve alone the 45° line, and AUC of approximately 0.50. When comparing several classifiers, the optimal classifier would be the one with an ROC curve shifted towards the upper left corner of the plot, or, the classifier with the largest AUC [45].

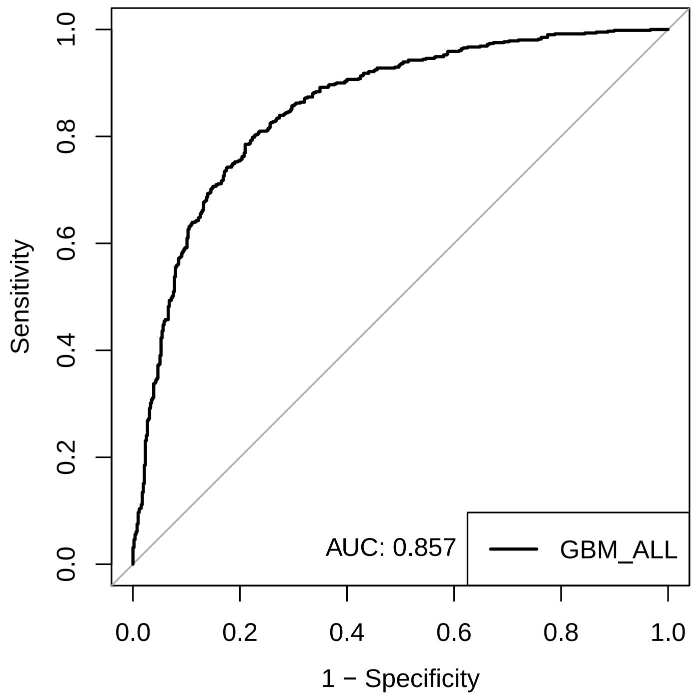


Figure 5.57: An example of a classifier's ROC Curve

This Section presents the ROC curves of the trained twenty-five classifiers grouped by the learning algorithm. The classifiers' area under ROC curve (AUC), and their 95% Confidence Interval (CI) are also presented.

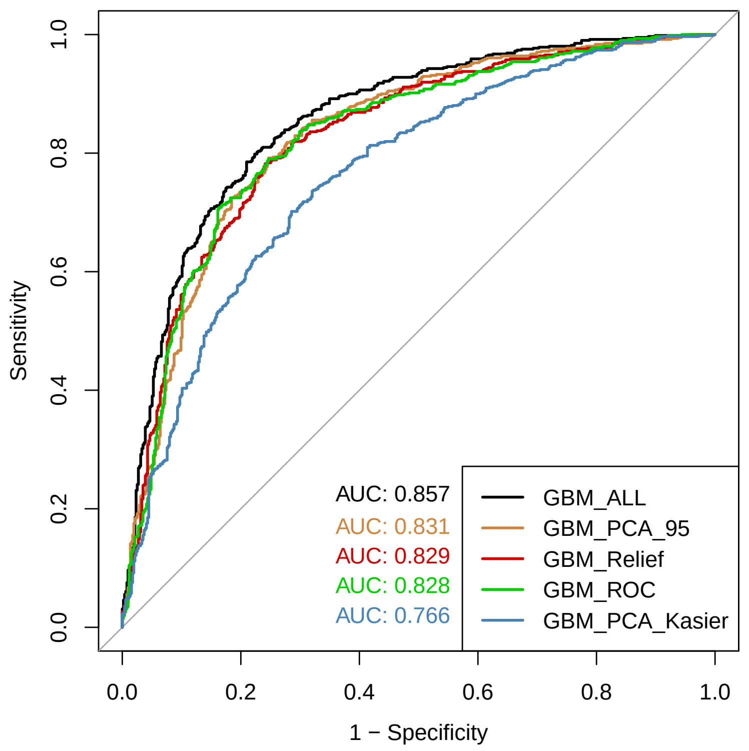


Figure 5.58: ROC Curves of the GBM classifiers

Figure 5.58 compares the ROC Curves of the GBM classifiers, while Table 5.6 ranks the GBM classifiers based on their AUC. The order of the classifiers (for the test dataset) is slightly different from the order previously obtained using the average 10-fold CV AUC (Figure 5.34b). Besides, the magnitudes of classifiers' AUC are more optimistic than the average 10-fold CV AUC.

Table 5.6: Area Under ROC Curve (AUC) of the GBM classifiers

Rank	Model	AUC (%)	AUC 95% CI (%)
1	GBM_ALL	85.65	(83.44 - 87.86)
2	GBM_PCA_95	83.15	(80.73 - 85.57)
3	GBM_Relief	82.90	(80.47 - 85.32)
4	GBM_ROC	82.83	(80.38 - 85.28)
5	GBM_PCA_Kasier	76.59	(73.82 - 79.36)

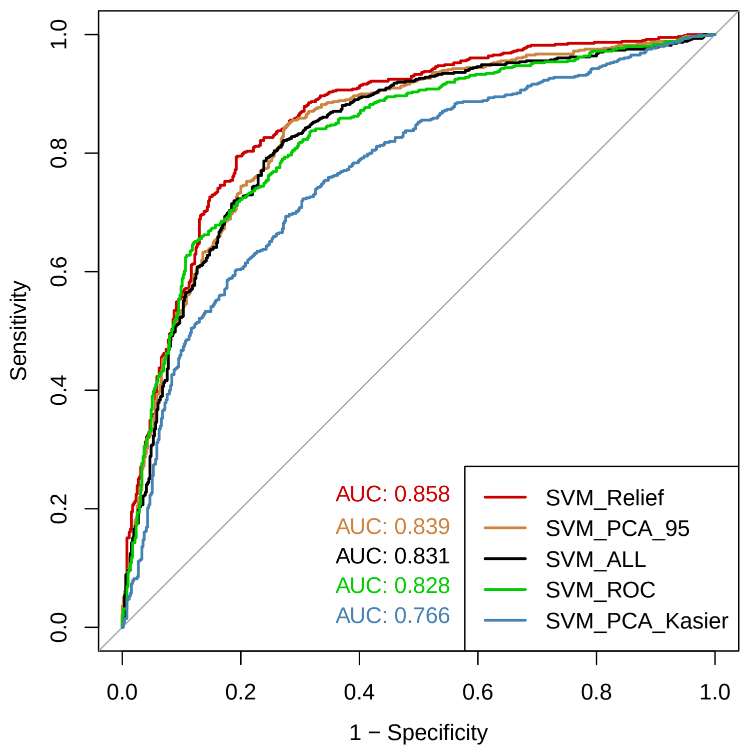


Figure 5.59: ROC Curves of the SVM classifiers

The ROC Curves of the SVM classifiers are compared in Figure 5.59. The classifiers' order based on their AUC, presented in Table 5.7, is slightly different and more optimistic than their order based on the average 10-fold CV AUC (Figure 5.35b).

Table 5.7: Area Under ROC Curve (AUC) of the SVM classifiers

Rank	Model	AUC (%)	AUC 95% CI (%)
1	SVM_Relief	85.79	(83.57 - 88.01)
2	SVM_PCA_95	83.85	(81.48 - 86.22)
3	SVM_ALL	83.08	(80.64 - 85.52)
4	SVM_ROC	82.84	(80.41 - 85.27)
5	SVM_PCA_Kasier	76.58	(73.80 - 79.37)

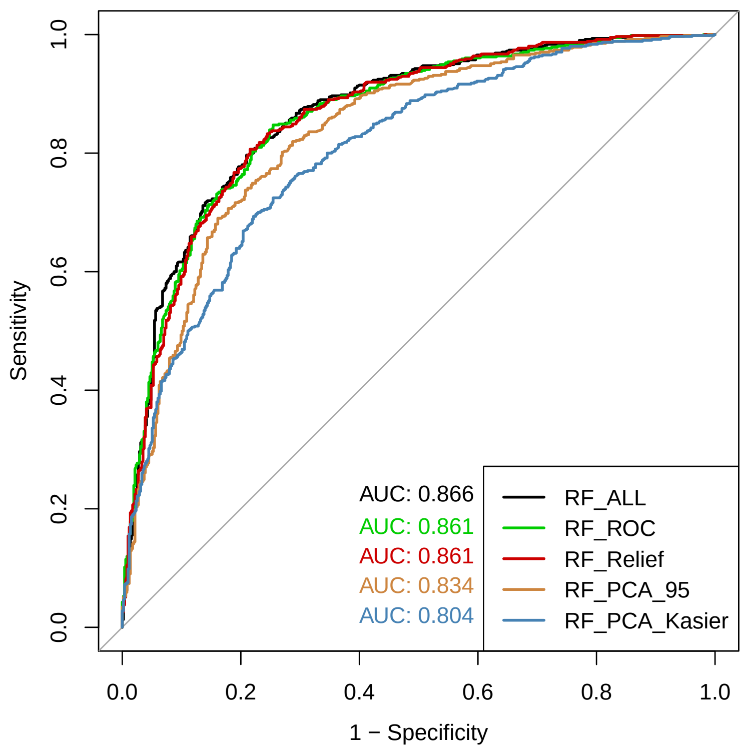


Figure 5.60: ROC Curves of the RF classifiers

Figure 5.60 compares the ROC Curves of the RF classifiers while Table 5.6 ranks the RF classifiers based on their AUC. This order is consistent with the order of RF classifiers based on the average 10-fold CV AUC (Figure 5.36b). However, the magnitudes of the AUC are more optimistic.

Table 5.8: Area Under ROC Curve (AUC) of the RF classifiers

Rank	Model	AUC (%)	AUC 95% CI (%)
1	RF_ALL	86.56	(84.42 - 88.69)
2	RF_ROC	86.14	(83.98 - 88.30)
3	RF_Relief	86.07	(83.90 - 88.24)
4	RF_PCA_95	83.36	(80.97 - 85.75)
5	RF_PCA_Kasier	80.41	(77.88 - 82.94)

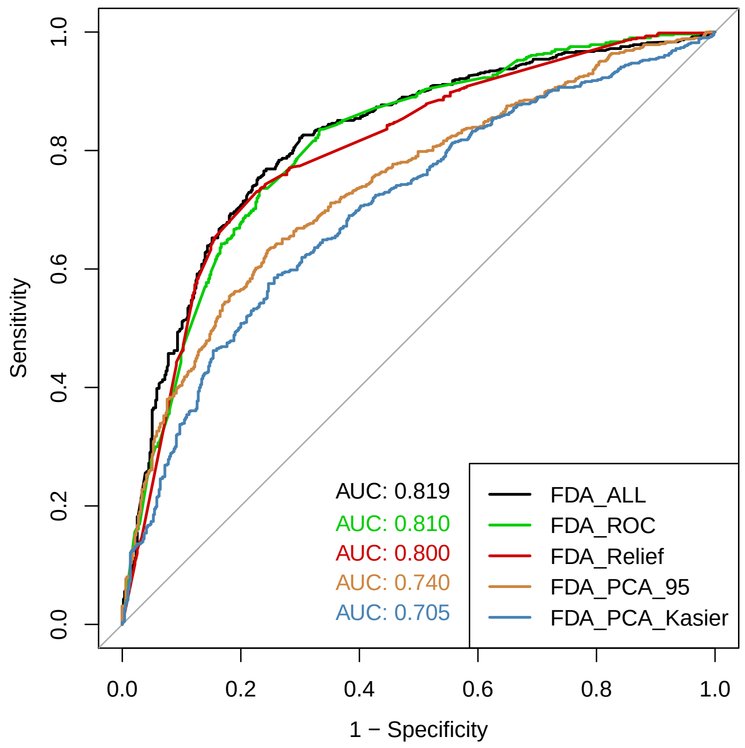


Figure 5.61: ROC Curves of the FDA classifiers

Figure 5.61 compares the ROC Curves of the FDA classifiers. The classifiers are ranked in Table 5.9 based on their AUC. This order is slightly different from the order of classifiers based on the average 10-fold CV AUC (Figure 5.38b). Moreover, the magnitudes of the AUC are more optimistic.

Table 5.9: Area Under ROC Curve (AUC) of the FDA classifiers

Rank	Model	AUC (%)	AUC 95% CI (%)
1	FDA_ALL	81.85	(79.35 - 84.36)
2	FDA_ROC	80.96	(78.42 - 83.51)
3	FDA_Relief	79.96	(77.36 - 82.56)
4	FDA_PCA_95	74.00	(71.13 - 76.87)
5	FDA_PCA_Kasier	70.45	(67.43 - 73.47)

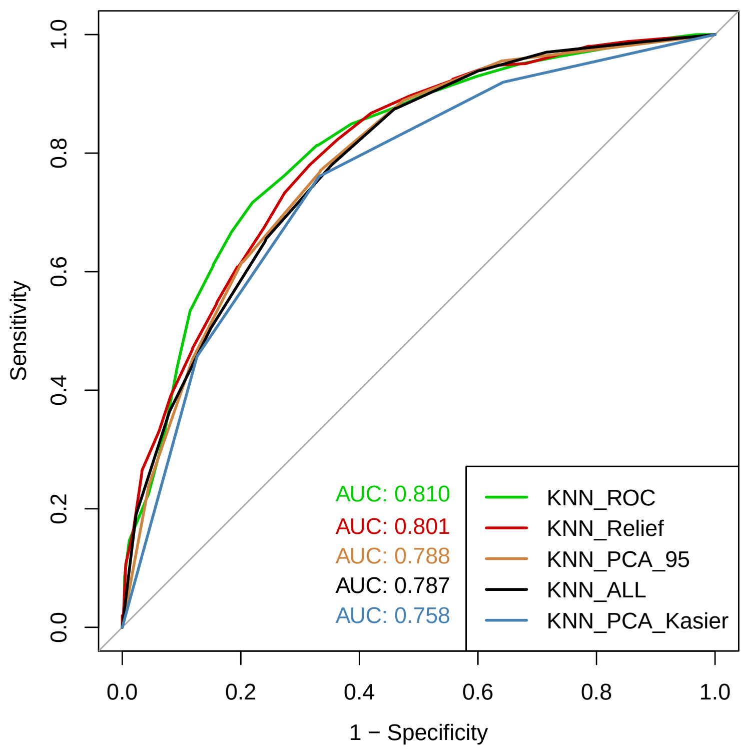


Figure 5.62: ROC Curves of the KNN classifiers

Finally, Figure 5.62 compares the ROC Curves of the KNN classifiers. The order of classifiers based on the AUC, presented in Table 5.10, is consistent with the order of classifiers based on the average 10-fold CV AUC (Figure 5.37b). Moreover, the magnitudes of the AUC are more optimistic.

Table 5.10: Area Under ROC Curve (AUC) of the KNN classifiers

Rank	Model	AUC (%)	AUC 95% CI (%)
1	KNN_ROC	81.00	(78.47 - 83.54)
2	KNN_Relief	80.15	(77.59 - 82.71)
3	KNN_PCA_95	78.79	(76.15 - 81.43)
4	KNN_ALL	78.73	(76.11 - 81.36)
5	KNN_PCA_Kasier	75.82	(73.80 - 79.37)

Table 5.11 ranks the twenty-five classifiers based on their area under ROC (using the test data set with 1125 samples). The order of the top performing classifiers based on the test dataset is consistent with the order based on the average 10-fold cross-validation AUC (Figure 5.44b), which indicates the robustness of the trained classifiers.

Table 5.11: All 25 models Area Under ROC Curve (AUC)

Rank	Model	AUC (%)	AUC 95% CI (%)
1	RF_ALL	86.56	(84.42 - 88.69)
2	RF_ROC	86.14	(83.98 - 88.30)
3	RF_Relief	86.07	(83.90 - 88.24)
4	SVM_Relief	85.79	(83.57 - 88.01)
5	GBM_ALL	85.65	(83.44 - 87.86)
6	SVM_PCA_95	83.85	(81.48 - 86.22)
7	RF_PCA_95	83.36	(80.97 - 85.75)
8	GBM_PCA_95	83.15	(80.73 - 85.57)
9	SVM_ALL	83.08	(80.64 - 85.52)
10	GBM_Relief	82.90	(80.47 - 85.32)
11	SVM_ROC	82.84	(80.41 - 85.27)
12	GBM_ROC	82.83	(80.38 - 85.28)
13	FDA_ALL	81.85	(79.35 - 84.36)
14	KNN_ROC	81.00	(78.47 - 83.54)
15	FDA_ROC	80.96	(78.42 - 83.51)
16	RF_PCA_Kasier	80.41	(77.88 - 82.94)
17	KNN_Relief	80.15	(77.59 - 82.71)
18	FDA_Relief	79.96	(77.36 - 82.56)
19	KNN_PCA_95	78.79	(76.15 - 81.43)
20	KNN_ALL	78.73	(76.11 - 81.36)
21	GBM_PCA_Kasier	76.59	(73.82 - 79.36)
22	SVM_PCA_Kasier	76.58	(73.80 - 79.37)
23	KNN_PCA_Kasier	75.82	(73.80 - 79.37)
24	FDA_PCA_95	74.00	(71.13 - 76.87)
25	FDA_PCA_Kasier	70.45	(67.43 - 73.47)

5.7.3 Class Predictions

The *confusion matrix* is one of the standard methods to quantify the performance of a classifier. Table 5.12 shows an example of the confusion matrix for a binary classification problem. The diagonal cells (True Positive (TP) + True Negative (TN)) represent the correctly predicted classes while the incorrectly predicted classes (False Positive (FP) + False Negative (FN)) are represented by the off-diagonals. The number of samples in the test dataset is $N = TP + TN + FP + FN$.

Table 5.12: An example of the confusion matrix

Predicted		Observed
	Active	Inactive
Active	TP	FP
Inactive	FN	TN

Several performance metrics are derived from the confusion matrix. Some of these metrics include [45]:

$$Accuracy = \frac{TP + TN}{N} \tag{5.1}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.2}$$

$$Specificity = \frac{TN}{TN + FP} \tag{5.3}$$

Another important metrics is the Kappa statistic [18], which takes into account the class distributions of the samples. It assesses the agreement between two raters.

$$Kappa = \frac{O - E}{1 - E} \tag{5.4}$$

Where O is the observed accuracy and E is the expected accuracy based on the marginal totals of the confusion matrix. Kappa takes values between -1 and 1. If there is no agreement between the observed and predicted classes, the Kappa statistic would have a value of 0.

A value of 1 indicates a perfect agreement while negative values indicate that the classifier's predictions are in the opposite direction of the truth. Generally, a reasonable agreement between the classifier's predictions and the observed classes is indicated by Kappa values within 0.30 to 0.50 [45].

Using these metrics, the rest of this section presents the performance of the trained twenty-five classifiers, grouped by the learning algorithm. For each group of classifiers, the total and shared miss-classified samples are visually examined to assess the diversity of the ensemble.

Table 5.13: Performance of GBM's classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	GBM_ALL	78.58	(76.06-80.94)	83.44	72.82	56.59
2	GBM_PCA_95	77.33	(74.77-79.75)	82.79	70.87	54.03
3	GBM_ROC	77.33	(74.77-79.75)	83.61	69.90	53.96
4	GBM_Relief	76.53	(73.94-78.98)	81.97	70.10	52.41
5	GBM_PCA_Kasier	70.67	(67.91-73.31)	75.08	65.44	40.68

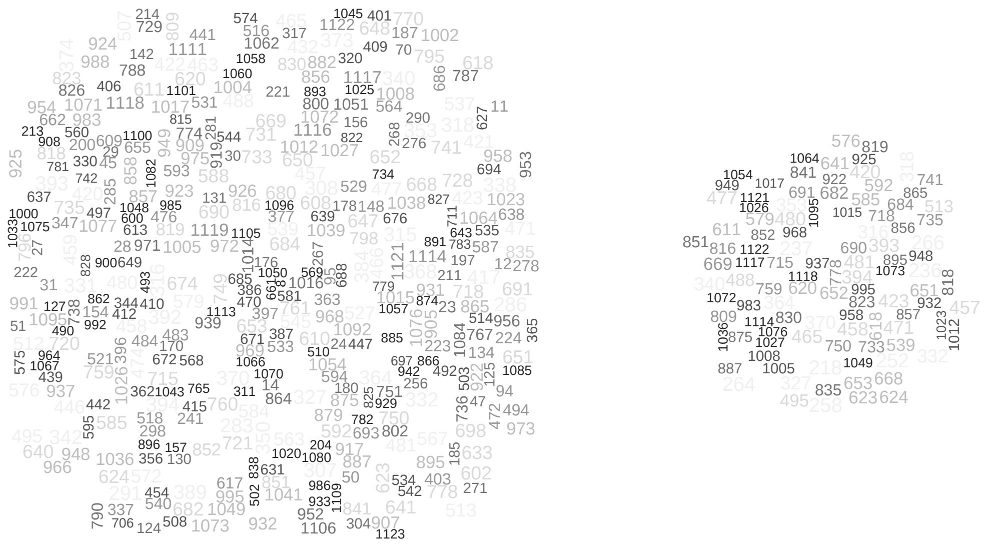


Figure 5.63: GBM's 5 classifiers, Left: Total miss-classifications (467 sample), Right: Shared miss-classifications (119 sample)

Table 5.13 ranks the GBM classifiers based on their accuracy. This order is consistent with the 5 GBM classifiers' order based on their area under ROC (Table 5.6). The total unique and shared miss-classified test samples are shown in Figure 5.63. The five classifiers share 25.48% of miss-classified samples, which indicates that injecting diversity at the feature and structural levels resulted in GBM classifiers that make reasonably different errors.

Table 5.14: Performance of SVM's classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	SVM_Relief	79.56	(77.08-81.88)	82.46	76.12	58.72
2	SVM_PCA_95	78.31	(75.79-80.69)	82.95	72.82	56.07
3	SVM_ALL	77.51	(74.96-79.92)	80.82	73.59	54.57
4	SVM_ROC	75.82	(73.21-78.30)	76.89	74.56	51.37
5	SVM_PCA_Kasier	71.02	(68.27-73.66)	75.90	65.24	41.34

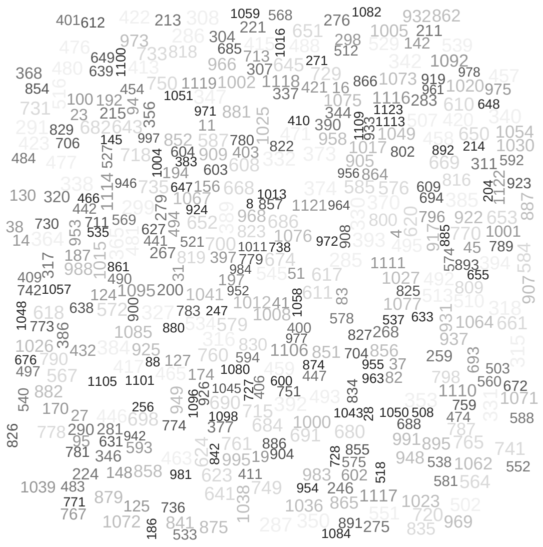
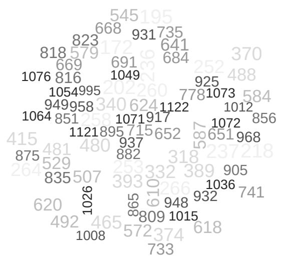



Figure 5.64: SVM's 5 classifiers, Left: Total miss-classifications (485 sample), Right: Shared miss-classifications (96 sample)

The five SVM classifiers are ranked based on their accuracy in Table 5.14, which exactly the same order based on their area under ROC (Table 5.7). Figure 5.64 shows that out of the 485 total miss-classified test samples only 19.79% are shared by the five SVM classifiers.

Table 5.15: Performance of RF's classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	RF_Relief	79.56	(77.08-81.88)	83.28	75.15	58.65
2	RF_ALL	79.29	(76.80-81.62)	81.97	76.12	58.20
3	RF_ROC	79.11	(76.62-81.45)	81.64	76.12	57.85
4	RF_PCA_95	76.98	(74.40-79.41)	81.80	71.26	53.36
5	RF_PCA_Kasier	73.16	(70.46-75.73)	76.89	68.74	45.76

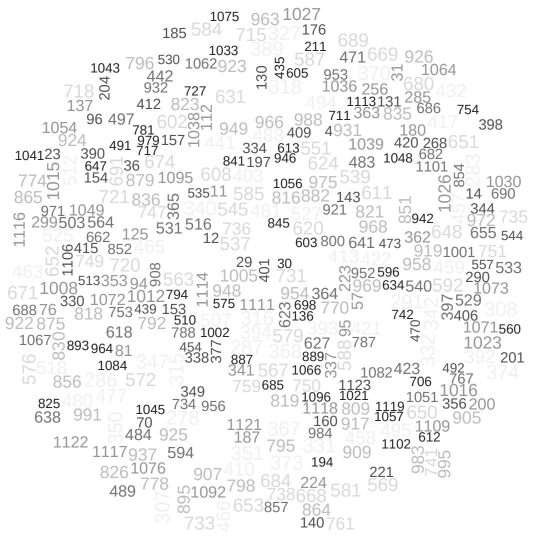
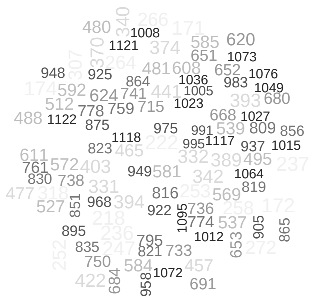



Figure 5.65: RF's 5 classifiers, Left: Total miss-classifications (411 sample), Right: Shared miss-classifications (124 sample)

Table 5.15 presents performance of the five RF classifiers based on their class predictions. The RF classifiers' accuracy based ranks is slightly different from their AUC based ranks (Table 5.8). Moreover, Figure 5.65 shows that 30.17% of the total unique miss-classified test samples are shared by the five RF classifiers, which indicates that injecting diversity at feature and structural levels for the RF learning algorithm did not result in the classifiers making substantially different errors.

Table 5.16: Performance of FDA's classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	FDA_ALL	76.27	(73.67-78.73)	76.39	76.12	52.34
2	FDA_ROC	74.76	(72.11-77.27)	73.61	76.12	49.44
3	FDA_Relief	74.58	(71.93-77.10)	76.72	72.04	48.77
4	FDA_PCA_95	68.00	(65.19-70.72)	67.70	68.35	35.87
5	FDA_PCA_Kasier	65.51	(62.65-68.29)	70.33	59.81	30.25

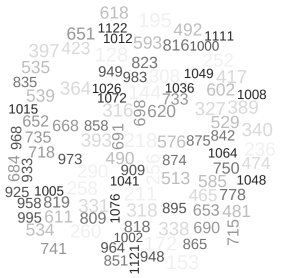


Figure 5.66: FDA's 5 classifiers, Left: Total miss-classifications (598 sample), Right: Shared miss-classifications (113 sample)

Table 5.16 ranks the five FDA classifiers based on their test accuracy. This classifiers' order is precisely the same order based on their AUC (Table 5.9). Despite that the total number of miss-classified test samples, by the FDA classifiers, is the largest among all the five learning algorithm based ensembles, FDA classifiers share only 18.90% of the miss-classifications. This indicates that with diversity injected at the feature and structural levels, FDA classifiers make substantially different errors, which would intuitively result in an ensemble's performance higher than the performance of any individual base classifier as we will see in Section 5.8.2.

Table 5.17: Performance of KNN's classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	KNN_ROC	74.76	(72.11-77.27)	81.31	66.99	48.71
2	KNN_Relief	73.60	(70.92-76.16)	78.03	68.35	46.59
3	KNN_PCA_95	72.27	(69.55-74.87)	77.05	66.60	43.86
4	KNN_ALL	71.91	(69.19-74.52)	78.03	64.66	43.01
5	KNN_PCA_Kasier	71.82	(69.09-74.44)	76.07	66.80	43.02

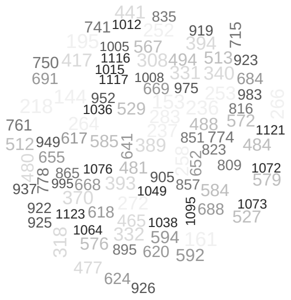


Figure 5.67: KNN's 5 classifiers, Left: Total miss-classifications (562 sample), Right: Shared miss-classifications (112 sample)

Finally, the five KNN classifiers are ranked based on their accuracy in Table 5.17. The KNN classifiers maintain the same order based on their AUC (Table 5.10). Similar to FDA classifiers, Figure 5.67 shows that the KNN classifiers produce substantially different errors by sharing only 19.93% of the total number of unique miss-classified test samples. As shown in Section 5.8.2, this diversity would result in a better overall ensemble's performance than the performance of the individual base classifiers.

Table 5.18: Performance of the 25 classifiers based on class predictions (N=1125)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	SVM_Relief	79.56	(77.08-81.88)	82.46	76.12	58.72
2	RF_Relief	79.56	(77.08-81.88)	83.28	75.15	58.65
3	RF_ALL	79.29	(76.80-81.62)	81.97	76.12	58.20
4	RF_ROC	79.11	(76.62-81.45)	81.64	76.12	57.85
5	GBM_ALL	78.58	(76.06-80.94)	83.44	72.82	56.59
6	SVM_PCA_95	78.31	(75.79-80.69)	82.95	72.82	56.07
7	SVM_ALL	77.51	(74.96-79.92)	80.82	73.59	54.57
8	GBM_PCA_95	77.33	(74.77-79.75)	82.79	70.87	54.03
9	GBM_ROC	77.33	(74.77-79.75)	83.61	69.90	53.96
10	RF_PCA_95	76.98	(74.40-79.41)	81.80	71.26	53.36
11	GBM_Relief	76.53	(73.94-78.98)	81.97	70.10	52.41
12	FDA_ALL	76.27	(73.67-78.73)	76.39	76.12	52.34
13	SVM_ROC	75.82	(73.21-78.30)	76.89	74.56	51.37
14	FDA_ROC	74.76	(72.11-77.27)	73.61	76.12	49.44
15	KNN_ROC	74.76	(72.11-77.27)	81.31	66.99	48.71
16	FDA_Relief	74.58	(71.93-77.10)	76.72	72.04	48.77
17	KNN_Relief	73.60	(70.92-76.16)	78.03	68.35	46.59
18	RF_PCA_Kasier	73.16	(70.46-75.73)	76.89	68.74	45.76
19	KNN_PCA_95	72.27	(69.55-74.87)	77.05	66.60	43.86
20	KNN_ALL	71.91	(69.19-74.52)	78.03	64.66	43.01
21	KNN_PCA_Kasier	71.82	(69.09-74.44)	76.07	66.80	43.02
22	SVM_PCA_Kasier	71.02	(68.27-73.66)	75.90	65.24	41.34
23	GBM_PCA_Kasier	70.67	(67.91-73.31)	75.08	65.44	40.68
24	FDA_PCA_95	68.00	(65.19-70.72)	67.70	68.35	35.87
25	FDA_PCA_Kasier	65.51	(62.65-68.29)	70.33	59.81	30.25

The twenty-five trained classifiers are ranked based on their accuracy in Table 5.18. The top ten performing classifiers are SVM, RF, and GBM based classifiers. Generally, the worst performing classifiers are built using PCA based features.

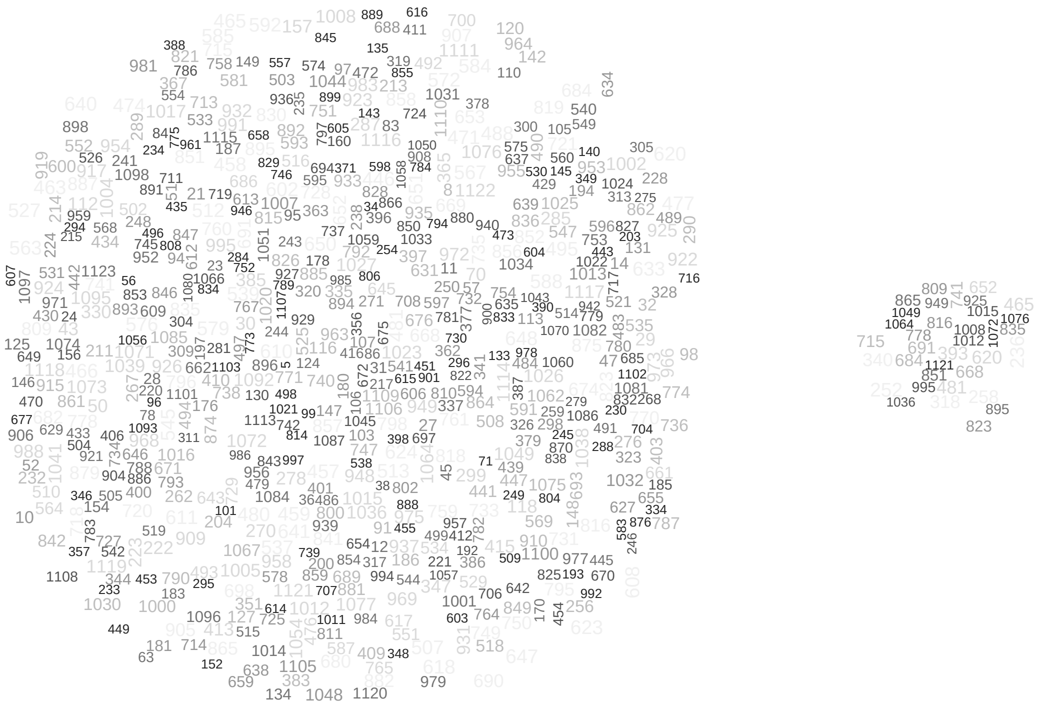


Figure 5.68: All 25 models, Left: Total miss-classifications (762 sample), Right: Shared miss-classifications (40 sample)

Table 5.19: Miss-classified samples by ensembles (test dataset, n=1125)

Rank	Models	Total	Shared	Shared(%)
1	All 25 classifiers	762	40	05.25
2	5 PCA 95 classifiers	562	102	18.15
3	5 FDA classifiers	598	113	18.90
4	5 SVM classifiers	485	96	19.79
5	5 KNN classifiers	562	112	19.93
6	5 PCA Kasier classifiers	596	124	20.81
7	5 All Pred. classifiers	494	112	22.67
8	5 Relief classifiers	457	114	24.95
9	5 GBM classifiers	467	119	25.48
10	5 ROC classifiers	430	125	29.07
11	5 RF classifiers	411	124	30.17

Out of 762 unique miss-classified samples, the twenty-five classifiers share only 05.25% of miss-classifications. This low percentage clearly indicates that injecting diversity at the feature, structural, and learning algorithm levels, would result in the ensemble's base classifiers make ultimately different errors. This diversity would intuitively lead to a the ultimate ensemble performance as shown in Section 5.8.2. Table 5.19 ranks feature, learning algorithm based ensembles and the 25 classifiers ensemble based on their percentage of shared miss-classified test samples. The overall performance of ensembles will be examined in the next section.

Table 5.20: Performance of the 25 classifiers based on class predictions (test dataset N=337)

Rank	Model	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
1	SVM_Relief	81.60	(77.05-85.59)	81.42	81.82	63.04
2	RF_ALL	80.71	(76.09-84.79)	80.87	80.52	61.24
3	RF_ROC	80.42	(75.77-84.52)	80.33	80.52	60.66
4	GBM_ROC	80.12	(75.45-84.25)	81.42	78.57	59.96
5	RF_Relief	80.12	(75.45-84.25)	81.97	77.92	59.92
6	SVM_PCA_95	80.12	(75.45-84.25)	82.51	77.27	59.88
7	GBM_ALL	79.82	(75.13-83.98)	81.42	77.92	59.34
8	SVM_ALL	79.23	(74.50-83.43)	79.78	78.57	58.23
9	GBM_PCA_95	78.34	(73.55-82.62)	80.87	75.32	56.29
10	FDA_ALL	78.04	(73.24-82.35)	73.77	83.12	56.25
11	GBM_Relief	78.04	(73.24-82.35)	80.33	75.32	55.71
12	SVM_ROC	77.74	(72.92-82.07)	75.41	80.52	55.50
13	RF_PCA_95	77.45	(72.61-81.80)	79.23	75.32	54.56
14	KNN_Relief	75.96	(71.04-80.43)	78.69	72.73	51.50
15	KNN_ROC	75.96	(71.04-80.43)	80.33	70.78	51.34
16	FDA_ROC	75.37	(70.41-79.88)	70.49	81.17	51.00
17	FDA_Relief	75.07	(70.10-79.60)	75.41	74.68	49.93
18	KNN_ALL	74.48	(69.47-79.05)	79.23	68.83	48.31
19	KNN_PCA_95	74.18	(69.16-78.77)	75.96	72.08	48.01
20	FDA_PCA_95	73.29	(68.23-77.94)	71.58	75.32	46.57
21	RF_PCA_Kasier	73.29	(68.23-77.94)	75.41	70.78	46.19
22	GBM_PCA_Kasier	72.40	(67.30-77.11)	74.86	69.48	44.37
23	SVM_PCA_Kasier	71.81	(66.68-76.55)	74.86	68.18	43.11
24	KNN_PCA_Kasier	71.22	(66.06-75.99)	71.58	70.78	42.21
25	FDA_PCA_Kasier	69.73	(64.52-74.59)	69.95	69.48	39.26

5.7.4 Computational Cost

Table 5.21: Computational cost for the twenty-five classifiers

Model	Model selection & training time	Prediction time
	All 1776 features	
GBM_ALL	5.2245 hours	1.9372 secs
SVM_ALL	1.7044 mins	2.2771 secs
RF_ALL	1.9791 hours	3.5183 secs
FDA_ALL	21.2017 mins	1.0911 secs
KNN_ALL	36.1009 mins	1.2461 mins
	PCA with 95% variance (255 components)	
GBM_PCA_95	22.3885 mins	0.4363 secs
SVM_PCA_95	45.8724 secs	0.6348 secs
RF_PCA_95	11.0254 mins	1.0178 secs
FDA_PCA_95	3.2669 mins	0.0752 secs
KNN_PCA_95	2.0230 mins	10.4531 secs
	PCA Kasier rule (11 components)	
GBM_PCA_Kasier	1.2937 mins	0.2612 secs
SVM_PCA_Kasier	28.4544 secs	0.4183 secs
RF_PCA_Kasier	1.0283 mins	0.8327 secs
FDA_PCA_Kasier	10.6081 secs	0.0091 secs
KNN_PCA_Kasier	8.2679 secs	0.0814 secs
	Predictors area under ROC curve (81 features)	
GBM_ROC	7.4156 mins	0.7164 secs
SVM_ROC	32.1860 secs	0.4335 secs
RF_ROC	3.8801 mins	0.9033 secs
FDA_ROC	1.1197 mins	0.0348 secs
KNN_ROC	12.5632 secs	0.4962 secs
	Relief algorithm (86 features)	
GBM_Relief	10.2612 mins	1.0299 secs
SVM_Relief	34.1287 secs	0.5358 secs
RF_Relief	3.9246 mins	0.8309 secs
FDA_Relief	1.1300 mins	0.0305 secs
KNN_Relief	13.3004 secs	0.5382 secs

Table 5.21 lists the computational cost for the twenty-five classifiers (e.g., model selection, training, and prediction times). The results show significant differences in training time between the learning algorithms. It should be noted that the model selection and training were performed on an Amazon EC2 instance with 32 cores and 65 GB RAM, while the predictions were performed on a local PC with 6 cores and 8 GB RAM.

5.8 Model Fusion

The choice of an appropriate fusion strategy may further improve the performance of a *diverse* ensemble [46, 48, 68, 82]. In this section, three non-linear fusion strategies: majority vote, stacked generalization, and fusion of intelligently selected classifiers are applied to several ensembles. Each ensemble is constructed using a different approach to injecting classifier diversity at the feature, structural, and/or learning algorithm levels. The performances of ensembles are compared to each other and the performances of the individual classifiers on a test data set of 337 samples, shown in Table 5.20. Also, in this section, the relationship between ten theoretical diversity measures and the empirical performance of the ensembles is assessed for each diversity injection approach.

To avoid over-fitting during training a second layer of fusion classifiers, in Section 5.8.3 and Section 5.8.4, the test dataset, with 1125 samples, is split into two data sets. The first data set, with 337 samples, is used as a final test dataset to evaluate the performance of ensembles and individual classifiers. The second data set, with 788 samples, is used as a training data set to train the fusion classifiers.

5.8.1 Diversity Measures

The theoretical diversity measures/metrics assume a relationship between diversity and the ensemble performance. If such a correlation exists, the classifiers to include in the ensemble could be selected in advance by using diversity (measured by these metrics) as a criterion [16, 46]. Although this research does not use the diversity measured by such metrics as a selection criterion, this section measures the ensemble diversity using ten diversity metrics (four pairwise and six non-pairwise) to investigate whether a high correlation exists between the diversity metrics and ensembles performance. These metrics are Yule's Q statistic ($\downarrow Q_{av}$), correlation ($\downarrow \rho_{av}$), Disagreement ($\uparrow D_{av}$), Double Fault ($\downarrow DF_{av}$), Kohavi-Wolpert Variance ($\uparrow KW$), interrater agreement ($\downarrow k$), Entropy ($\uparrow E$), measure of difficulty ($\downarrow \theta$), Generalized Diversity ($\uparrow GD$), and Coincident Failure Diversity ($\uparrow CFD$).

The down arrow (↓) means a lower metric value indicates a better diversity while the up arrow (↑) means a higher value indicates a better diversity. Complete descriptions of these diversity measures can be found in [46, 68].

Measurements of ensemble diversity at the feature and structural levels are shown in Table 5.22. Each row represents an ensemble of five classifiers built using the same learning algorithm (GBM, SVM, RF, FDA, or KNN) and five feature selection approaches (all features, PCA with 95%, PCA Kasier, ROC, and Relief).

Table 5.22: Diversity measures (at the feature and structural levels, $N = 1125$)

Models	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
GBM	.8850	.6085	.1465	.1659	.0586	.5974	.2116	.1233	.3063	.5300
SVM	.8545	.5623	.1618	.1547	.0647	.5508	.2316	.1154	.3434	.5670
RF	.9244	.6573	.1223	.1627	.0489	.6480	.1782	.1248	.2732	.4842
FDA	.7486	.4950	.2117	.1759	.0847	.4769	.3138	.1177	.3757	.5874
KNN	.8180	.4964	.1991	.1717	.0796	.4964	.2978	.1180	.3670	.5712

Table 5.23 shows the measurements of ensemble diversity at the learning algorithm level. Each row represents an ensemble of five classifiers built using the same feature selection approach (all features, PCA with 95%, PCA Kasier, ROC, or Relief) and five learning algorithms (GBM, SVM, RF, FDA, and KNN).

Table 5.23: Diversity measures (at the learning algorithm level, $N = 1125$)

Models	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
All	.8733	.5656	.1584	.1537	.0634	.5567	.2262	.1153	.3401	.5870
PCA_95	.8218	.5064	.1927	.1579	.0771	.4918	.2773	.1125	.3790	.6139
PCA_k	.8250	.5119	.2048	.2048	.0819	.5083	.3022	.1263	.3464	.5524
ROC	.9222	.6458	.1282	.1724	.0513	.6450	.1849	.1293	.2711	.4767
Relief	.9020	.5921	.1477	.1585	.0591	.5859	.2169	.1193	.3179	.5350

Finally, Table 5.24 shows the measurements of ensemble diversity at the feature, structural and learning algorithm levels. The ensemble is constructed using all 25 classifiers.

Table 5.24: Diversity measures (at the feature, structural, and learning algorithm levels, N = 1125)

Models	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
`All_25`	.8196	.5059	.1896	.1555	.0910	.4949	.2701	.0967	.3786	.6567

The three tables, Table 5.22, Table 5.23, and Table 5.24 show that most of the diversity metrics do not agree with each other on the most diverse ensemble even within the same diversity injection approach. In the next section, the theoretical assumption that there exists a correlation between diversity measured by the ten metrics and ensembles performance is empirically examined.

5.8.2 Fusion By Majority Vote

Majority vote, discussed in Section 2.5.1, is a simple but an effective fusion approach, where the class label of a new sample is assigned by the ensemble based on the majority vote of its base classifiers. In this section, the majority vote is used as a fusion function for eleven ensembles. These ensembles are constructed to inject diversity at feature, structural, or/and learning algorithm levels.

At the feature and structural levels, five ensembles are formed. Each ensemble consists of five classifiers. These classifiers are built using one learning algorithm (GBM, SVM, RF, FDA, or KNN), and five feature selection approaches (all features, PCA 95%, PCA Kasier, ROC, and Relief algorithm). Diversity at the learning algorithm is achieved by constructing five ensembles. Each ensemble consists of five classifiers built using one feature selection approach (all features, PCA 95%, PCA Kasier, ROC, or Relief algorithm) and five learning algorithms (GBM, SVM, RF, FDA, and KNN). Finally, the diversity at the feature, structural, and learning algorithm levels is achieved by grouping all twenty-five models built using the five learning algorithms and five feature selection approaches. Since the base classifiers are independent of each other, the parallel topology is chosen as the fusion topology for all ensembles.

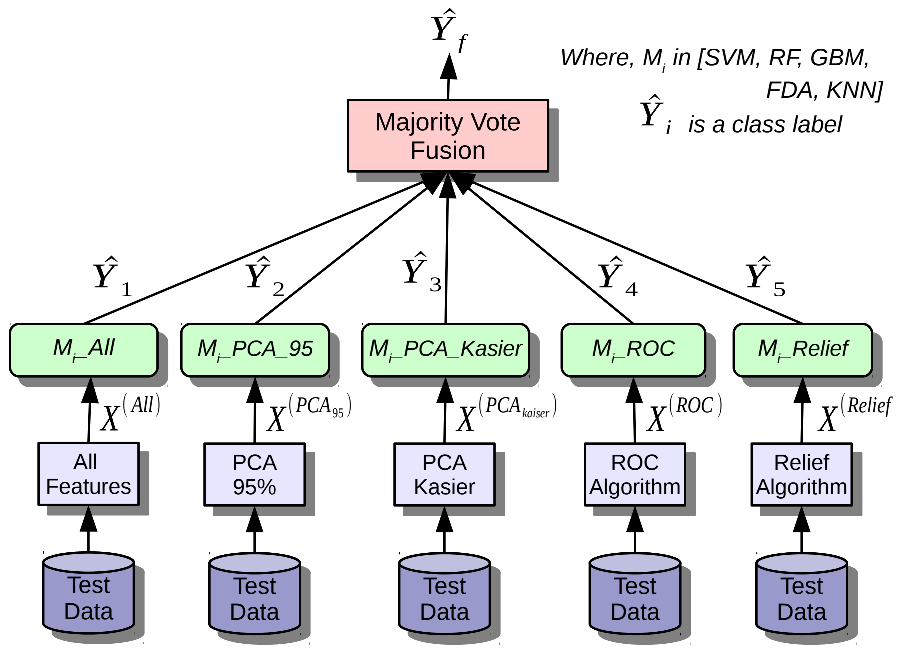


Figure 5.69: Fusion by majority vote topology (diversity at the feature and structural levels)

Figure 5.69 shows the majority vote fusion topology of ensembles with diversity at the feature and structural levels. The performance of the five ensembles on the test dataset is presented in Table 5.25. Although the SVM ensemble outperformed the other four ensembles with 80.71% accuracy, it failed to outperform the best performing individual classifier *SVM_Relief* with 81.6% accuracy.

Table 5.25: Model fusion by majority vote (diversity at the feature and structural levels, N=337)

Ensemble	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
SVM	80.71	(76.09, 84.79)	81.42	79.87	61.20
RF	80.42	(75.77, 84.52)	81.42	79.22	60.58
GBM	79.82	(75.13, 83.98)	82.51	76.62	59.26
FDA	77.45	(72.61, 81.80)	73.22	82.47	55.07
KNN	76.26	(71.35, 80.70)	79.78	72.08	52.02

Table 5.26, presents the ensembles ranks based on the ten diversity metrics. Although the double fault (DF_{av}) metric was able to predict the best performing ensemble (SVM

ensemble), it failed to rank the remaining ensembles based on their accuracy. The other nine diversity metrics failed to rank all ensembles.

Table 5.26: Ensembles ranks based on the diversity measures (at the feature and structural levels, N=337)

Ensemble	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
SVM	3rd	3rd	3rd	1st	3rd	3rd	3rd	3rd	3rd	3rd
RF	5th	5th	5th	5th	5th	5th	5th	5th	5th	5th
GBM	4th	4th	4th	4th	4th	4th	4th	4th	4th	4th
FDA	1st	1st	1st	2nd	1st	1st	1st	1st	1st	1st
KNN	2nd	2nd	2nd	3rd	2nd	2nd	2nd	2nd	2nd	2nd

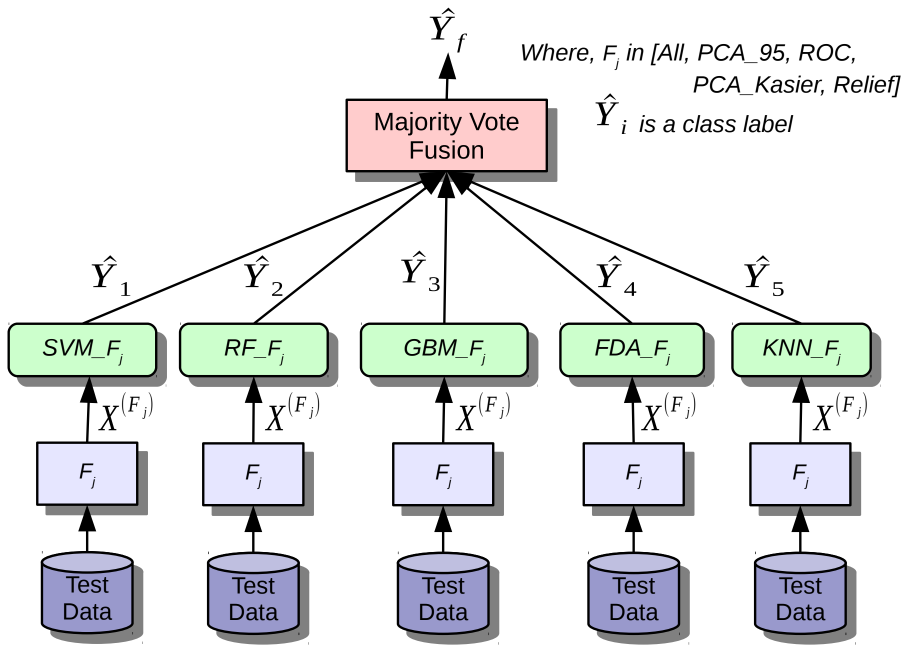


Figure 5.70: Fusion by majority vote topology (diversity at the learning algorithm level)

The majority vote fusion topology of ensembles with diversity at the learning algorithm level is shown in Figure 5.70. The performance of the five ensembles is presented in Table 5.27. The most accurate ensemble, all features ensemble with 81.31% accuracy, failed to outperform the best individual classifier *SVM_Relief* with 81.6% accuracy.

Table 5.27: Model fusion by majority vote (diversity at the learning algorithm level, $N = 337$)

Ensemble	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
All	81.31	(76.73, 85.33)	81.42	81.17	62.43
Relief	79.82	(75.13, 83.98)	80.87	78.57	59.38
PCA_95	79.82	(75.13, 83.98)	80.87	78.57	59.38
ROC	78.34	(73.55, 82.62)	78.14	78.57	56.51
PCA_k	73.29	(68.23, 77.94)	75.96	70.13	46.13

Ensembles with diversity at the learning algorithm level are ranked based on the ten diversity metrics in Table 5.28. The double fault (DF_{av}) metric ranked almost all ensembles correctly based on their accuracy. Ensembles ranks by the remaining metrics did not match their performance.

Table 5.28: Ensembles ranks based on the diversity measures (at the learning algorithm level, $N = 337$)

Ensemble	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
All	3rd	3rd	3rd	1st	3rd	3rd	3rd	2nd	3rd	2nd
Relief	4th	4th	4th	3rd	4th	4th	4th	3rd	4th	3rd
PCA_95	1st	1st	2nd	2nd	2nd	2nd	1st	1st	1st	1st
ROC	5th	5th	5th	4th	5th	5th	5th	5th	5th	5th
PCA_k	2nd	2nd	1st	5th	1st	1st	2nd	4th	2nd	4th

Finally, the majority vote fusion topology of the ensemble with diversity at the feature, structural, and learning algorithm levels is shown in Figure 5.71. The performance of the ensemble, presented in Table 5.29, which combines all twenty-five base classifiers (with 82.20% accuracy, 83.06% sensitivity, and 64.16% kappa) outperforms the best single classifier, *SVM_Relief* (with 81.6% accuracy, 81.42% sensitivity, and 63.04% kappa). However, the ensemble's specificity (81.17%) was slightly lower than the specificity of the *SVM_Relief* classifier (81.82%).

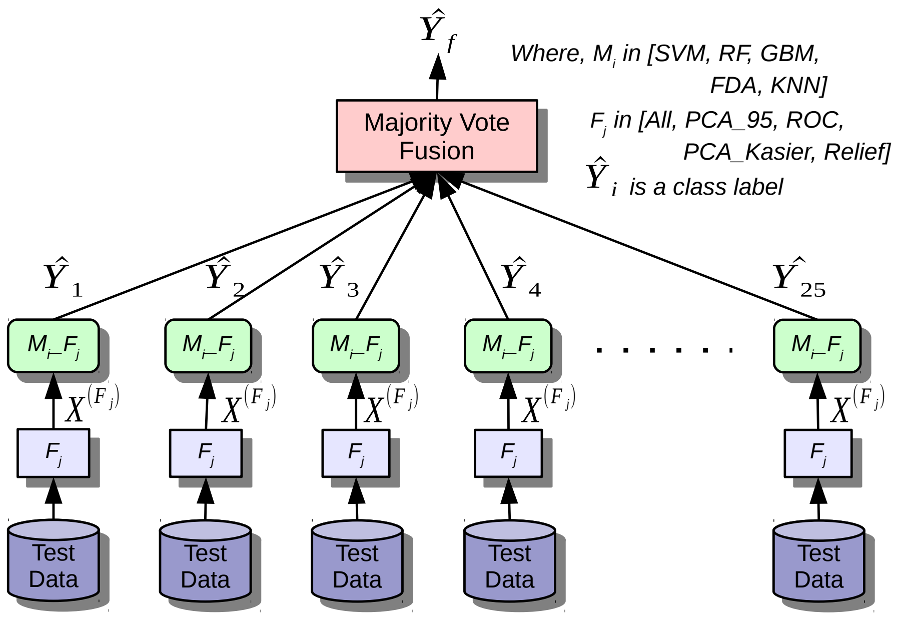


Figure 5.71: Fusion by majority vote topology (diversity at the feature, structural, and learning algorithm levels, test dataset $n = 337$)

Table 5.29: Model fusion by majority vote (diversity at the feature, structural, and learning algorithm levels)

Ensemble	Acc.(%)	95% CI(%)	Sens.(%)	Spec.(%)	Kappa(%)
All_25	82.20	(77.69, 86.13)	83.06	81.17	64.16

Table 5.30: Ensemble's rank based on the diversity measures (at the feature, structural, and learning algorithm levels, $N = 337$)

Ensemble	$\downarrow Q_{av}$	$\downarrow \rho_{av}$	$\uparrow D_{av}$	$\downarrow DF_{av}$	$\uparrow KW$	$\downarrow k$	$\uparrow E$	$\downarrow \theta$	$\uparrow GD$	$\uparrow CFD$
All_25	4th	4th	5th	3rd	1st	4th	5th	1st	4th	1st

Three of the ten diversity metrics, shown in Table 5.30, were able to predict that the ensemble with twenty-five base classifiers (diversity injected at the feature, structural, and learning algorithm levels) is the best performing ensemble compared to the other ensembles with diversity injected at one level.

In this section, the simple majority vote was used as the fusion function. The ensemble with diversity at all levels (at the feature, structural, and learning algorithm levels) outperforms the best single classifier and other ensembles with diversity injected at one level. The experimental results clearly indicate a weak correlation between the diversity, measured by the ten most used theoretical diversity metrics, and ensembles' performance. Therefore, these theoretical metrics should not be used as a guide to constructing the ensembles. In the next section, a more sophisticated fusion strategy, model stacking, is employed to try further improve the performance of the ensemble with diversity injected at all levels.

5.8.3 Fusion by Model Stacking

In the previous section, the majority vote was used to combine the class labels predicted by the twenty-five base classifiers to produce a final class label. In this section, the model stacking fusion strategy, discussed in Section 2.5.4, is employed. In this sophisticated fusion strategy, a new classifier is trained using the class probabilities produced by the base classifiers as predictors (independent variables) and the actual class label as the response. The fusion by model stacking topology is shown in Figure 5.72.

As discussed in Section 3.5.2, to avoid over-fitting, the class probabilities produced by the twenty-five base classifiers must be for a new data set, which was not used to train the base classifiers, or cross-validation's out-of-fold predictions. In our case, the test data set (1125 samples) is split into two parts, a training dataset (788 samples) to produce the class probabilities and train the fusion classifier, and a final test dataset (337 samples) to evaluate and compare ensembles' performance with the performance of the best single classifier (SVM_Relief).

To compensate for the relatively small training data set, the bootstrap resampling (25 bootstraps) is employed for model selection. Another important factor in model selection, is the loss function being optimized. Three different loss function, the logistic loss or cross-entropy function [1] (minimize), Area Under ROC (AUC) (maximize), and Kappa (maximize), are used to select the optimal fusion classifier. Finally, three different learning algorithms, Support Vector Machine (SVM), Random Forest (RF), and Stochastic Gradient Boosting Machine (GBM) are used to build the fusion classifiers (a total of 9 ensembles). The goal, in addition to obtaining the most accurate prediction, is to compare the effect of the loss function and learning algorithm on the performance of the ensemble.

[1] $logloss = -\frac{1}{N}\sum_1^N (y_i log(p_i) + (1-y_i) log(1-p_i))$, where $y_i \Rightarrow$ true class, and $p_i \Rightarrow$ predicted probability.

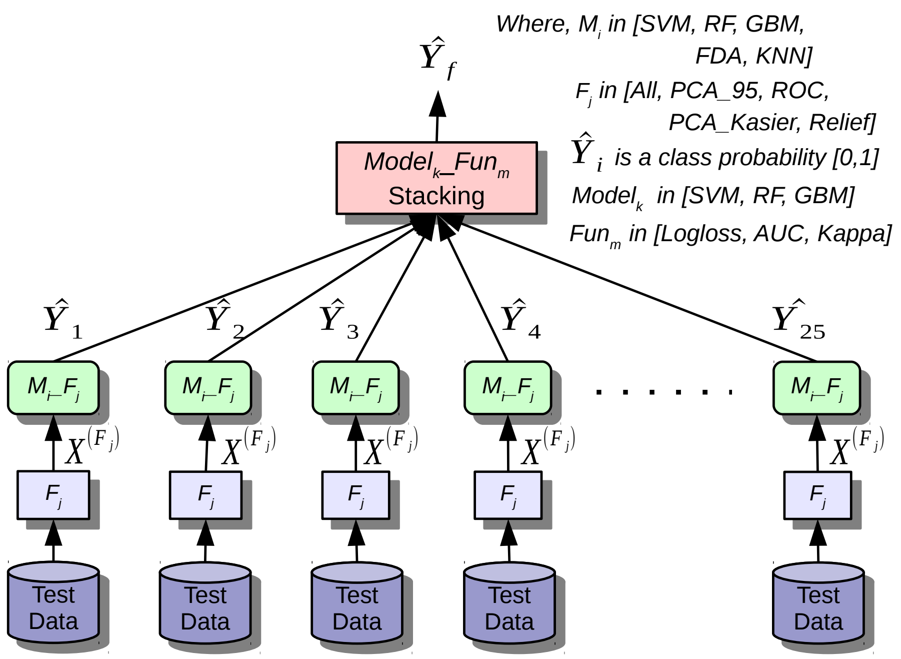

Where, M_i in [SVM, RF, GBM, FDA, KNN]
F_j in [All, PCA_95, ROC, PCA_Kasier, Relief]
$\hat{Y}_i$ is a class probability [0,1]
$Model_k$ in [SVM, RF, GBM]
Fun_m in [Logloss, AUC, Kappa]
$\hat{Y}_f$
$Model_k\_Fun_m$ Stacking
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
$\hat{Y}_4$
$\hat{Y}_{25}$
$M_i\_F_j$
$X^{(F_j)}$
F_j
Test Data

Figure 5.72: Fusion by stacking topology (diversity at the feature, structural, and learning algorithm levels)

Table 5.31: Model fusion by model stacking (on top of 25 base models) to minimize log loss function, $N = 337$

Model	Acc.(%)	95% CI(%)	AUC.(%)	Sens.(%)	Spec.(%)	Kappa(%)
SVM_Log	81.90	(77.37, 85.86)	88.17	83.61	79.87	63.51
RF_Log	81.31	(76.73, 85.33)	88.66	81.97	80.52	62.39
GBM_Log	81.01	(76.41, 85.06)	88.28	83.61	77.92	61.66

The results of model stacking fusion built by minimizing the log loss function are presented in Table 5.31. The SVM fusion classifier produced the best ensemble's performance in terms of accuracy (81.90%), sensitivity (83.61%), and Kappa (63.51%). In terms of AUC and specificity, the RF fusion classifiers produced the best performance (88.66% and 80.52% respectively). The performance of the SVM fusion classifier also outperforms the best single classifier, *SVM_Relief*, in terms of accuracy (81.6%), sensitivity (81.42%), and Kappa (63.04%), but failed to outperform its specificity (81.82%). The achieved improvement also failed to outperform the performance of the ensemble produced by the simple majority vote fusion strategy.

Table 5.32: Model fusion by model stacking (on top of 25 base models) to maximize AUC, $N = 337$

Model	Acc.(%)	95% CI(%)	AUC.(%)	Sens.(%)	Spec.(%)	Kappa(%)
GBM_AUC	81.60	(77.05, 85.59)	88.17	83.06	79.87	62.93
RF_AUC	81.31	(76.73, 85.33)	88.66	81.97	80.52	62.39
SVM_AUC	80.12	(75.45, 84.25)	88.27	80.87	79.22	60.00

The results of model stacking fusion built by maximizing the AUC are presented in Table 5.32. The GBM fusion classifier produced the best ensemble's performance in terms of accuracy (81.60%), sensitivity (83.06%), and Kappa (62.93%). In terms of AUC and specificity, the RF fusion classifiers produced the best performance (88.66% and 80.52% respectively). The performance of the GBM fusion classifier also outperforms the best single classifier, *SVM_Relief*, in terms of sensitivity (81.42%) with similar accuracy (81.6%), but failed to outperform its specificity (81.82%) and Kappa (63.04%). The achieved improvement also failed to outperform the performance of the ensemble produced by the simple majority vote fusion strategy.

Table 5.33: Model fusion by model stacking (on top of 25 base models) to maximize Kappa, $N = 337$

Model	Acc.(%)	95% CI(%)	AUC.(%)	Sens.(%)	Spec.(%)	Kappa(%)
RF_Kappa	81.90	(77.37, 85.86)	88.90	83.06	80.52	63.55
SVM_Kappa	81.60	(77.05, 85.59)	88.16	83.61	79.22	62.89
GBM_Kappa	80.71	(76.09, 84.79)	88.03	84.15	76.62	61.00

Finally, the results of model stacking fusion built by maximizing the Kappa measure are presented in Table 5.33. The RF fusion classifier produced the best ensemble's performance in terms of accuracy (81.90%), AUC (88.90%), specificity (80.52%), and Kappa (63.55%). In terms of sensitivity, the GBM fusion classifiers produced the best performance (84.15%). The performance of the RF fusion classifier also outperforms the best single classifier, *SVM_Relief*, in terms of accuracy (81.6%), sensitivity (81.42%), and Kappa (63.04%), but failed to outperform its specificity (81.82%). The achieved improvement also failed to outperform the performance of the ensemble produced by the simple majority vote fusion strategy.

In this section, three loss functions and three learning algorithms were used to build fusion classifiers by model stacking. It is shown that different optimization loss functions produce different ensemble's performances for the same learning algorithm. Although better ensemble's performance than the best single classifier was achieved by the complex model stacking fusion strategy, the achieved performance failed to outperform the simple majority vote fusion strategy.

5.8.4 Fusion of Intelligently Selected Models

To take into account the fact that some of the twenty-five classifiers perform better than the others, the model stacking fusion strategy, Section 5.8.3, is adjusted to combine the class probabilities of only intelligently selected classifiers (e.g using Genetic Algorithms GAs), as shown in Figure 5.73.

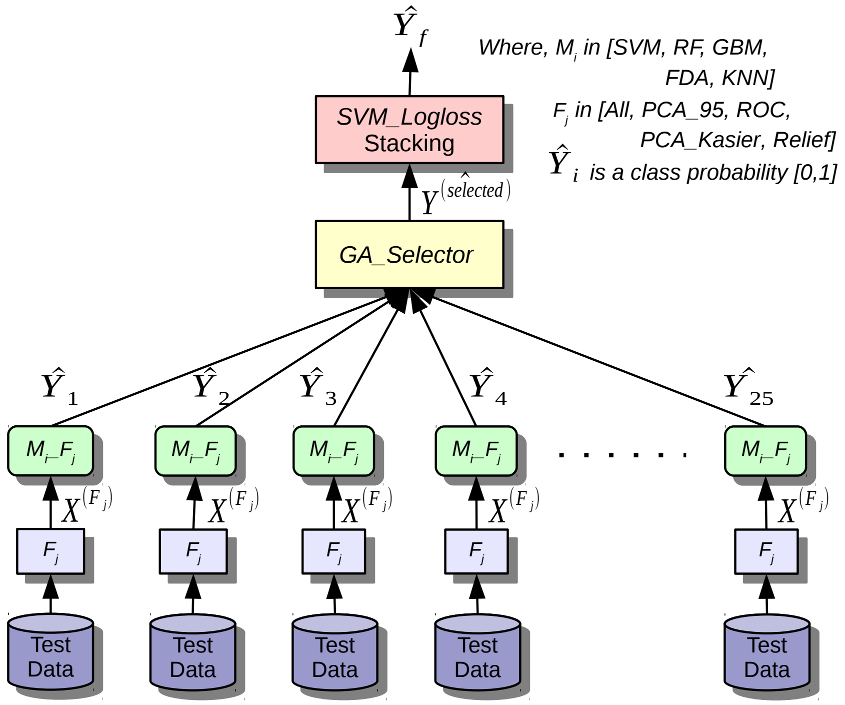


Figure 5.73: Fusion of intelligently selected models topology (diversity at the feature, structural, and learning algorithm levels)

To reduce the risk of over-fitting, any extensive search based algorithm, such GAs, must be combined with a proper resampling procedure. For this case study, GA was used to conduct the search of the feature space (the 25 class probabilities produced by the base classifiers) repeatedly within 10-fold cross-validation resampling. The entire GA is carried out ten separate times. For example, for the first fold, nine tenths of the data are used in the search while the remaining tenth is used to estimate the external performance. Another important aspect of fusion by model stacking is tuning the hyper-parameters (model selection) of the fusion classifier using another resampling procedure. Therefore, inside of the external resampling loop, the fusion classifier (SVM) is tuned using bootstrapping resampling (25 bootstraps). The logloss function is used to guide (internally) and to know if the GA has over-fitted the features (externally). After the external resampling is completed, the optimal number of generations for the GA and SVM's hyper-parameters are determined. Finally, the entire data set is used in the last execution of the GA search, and the final fusion classifier is built on the subset of the base classifiers that is associated with the optimal number of generations determined by resampling.

Needless to say, using GA with two resampling producers substantially increases the computational intensity of this fusion strategy, which requires parallel computing and large RAM to make it feasible to execute within an acceptable time. For this case study (with a small fusion dataset), using a compute-optimized instant of Amazon EC2 with 32 cores and 65 Gigabyte RAM, the fusion strategy took approximately 12 hours to execute.

Table 5.34 presents the results of combining intelligently selected classifiers using a SVM as the fusion function (meta-learner). The results indicate that this fusion strategy has significantly outperformed previous fusion strategies in all metrics.

Table 5.34: Fusion of intelligently selected classifiers to minimize Logloss function, $N = 337$

Model	Acc.(%)	95% CI(%)	AUC.(%)	Sens.(%)	Spec.(%)	Kappa(%)
SVM_GA_Log	83.38	(78.97, 87.20)	88.97	83.06	83.77	66.62

5.8.5 Computational Cost

Table 5.35 lists the computational cost for the fusion strategies (e.g., model selection, training, and prediction times). The results show significant differences in training time between the fusion strategies, especially the fusion of intelligently selected models. It should be noted that the model selection and training were performed on an Amazon EC2 instance with 32 cores and 65 GB RAM, while the predictions were performed on a local PC with 6 cores and 8 GB RAM.

Table 5.35: Computational cost for the fusion strategies

Model	Model selection & training time	Prediction time
	Model stacking to minimize the logloss function	
SVM_logloss	25.3077 secs	0.1673 secs
RF_logloss	4.3950 mins	0.1798 secs
GBM_logloss	21.9762 mins	0.0675 secs
	Model stacking to maximize the AUC	
SVM_AUC	25.6098 secs	0.1186 secs
RF_AUC	4.3835 mins	0.1705 secs
GBM_AUC	21.9071 mins	0.0482 secs
	Model stacking to maximize the Kappa	
SVM_Kappa	24.9003 secs	0.1035 secs
RF_Kappa	4.3189 mins	0.1627 secs
GBM_Kappa	21.8565 mins	0.0411 secs
	Fusion of intelligently selected models	
SVM_logloss	11.8385 hours	0.1854 secs

5.9 Summary

In this chapter, the proposed framework for ensemble predictive modeling was successfully applied to a high-dimensional classification case study, to predict the biological response of molecules from their chemical properties. The effectiveness of model diversity approaches on the performance of ensembles were thoroughly evaluated. In addition, the relative performance of several feature selection approaches and learning algorithms were empirically compared. The next chapter presents the conclusion and future work.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Ensemble systems, which build a predictive system by combining the predictions of multiple classifiers/models, are used to improve the prediction performance, and mitigate the risk of selecting the wrong classifier/model for the task. Researchers from various disciplines such as statistics, pattern recognition, and machine learning have explored different aspects of ensemble systems, especially from the system's designer point of view. However, there are still research gaps need to be addressed. First, a comprehensive framework for developing ensemble systems is not yet available. Second, several approaches have been proposed to inject model diversity in the ensemble; however, there is a shortage of empirical studies that compare the effectiveness of these approaches. Third, most of the ensemble systems research has concentrated on simple applications and relatively small/low-dimensional data sets. Further experimental research is required to investigate the application of ensemble systems to real-world, large and/or high-dimensional, data sets. This thesis attempts to fill these gaps.

The primary contribution of this thesis is a framework for ensemble predictive modeling, shown in Figure 3.1. The proposed framework addressed seven fundamental design issues:

- **Data preparation**: Despite its importance, data preparation is overlooked by the ensemble design methods reported in the literature. The proposed framework incorporates data preparation as one of the main phases of developing ensemble systems. A typical data preparation phase may involve exploratory data analysis, feature engineering, data pre-processing, and data partition.

- **Injecting model diversity**: Model diversity is considered to be an essential design feature for any ensemble system. To maximize the ensemble performance, the proposed framework calls for the systematic injection of model diversity at the data, feature, and learning algorithm levels.

- **Model selection**: in the context of ensemble predictive modeling, model selection may be classified into three types. First, the selection of the ensemble size, how many base models/classifiers should be trained. Second, the selection of the optimal hyper-parameters for the learning algorithms over a set of candidate values. Third, the selection of the learning algorithms to use over others. The proposed framework suggests building small to medium (e.g., from 3 to 50 classifiers) heterogeneous ensembles of strong base classifiers/models. For hyper-parameters, it recommends using a quantitative approach, such as grid-search and n-fold cross-validation, to select the optimal values. It advises the use of visualizations to better understand the effect of different hyper-parameters combinations, which helps refine the grid-search, and estimate the uncertainty of the selected values. Finally, the proposed framework stresses the important of using a fair quantitative approach, such n-fold cross-validation, to select the learning algorithms.

- **Model evaluation**: model evaluation is of paramount importance in any predictive modeling task. It becomes even more important in ensemble predictive modeling, where the relative performance and diversity of models/classifiers must be thoroughly evaluated. The proposed framework calls for using several performance metrics and visualizations to evaluate models/classifiers and understand their strengths and weaknesses. Therefore, the proposed framework does not use these metrics as criteria to select the ensemble classifiers.

- **Model fusion**: once a set of diverse and accurate models/classifiers are built, an appropriate fusion strategy should be selected to obtain the optimal ensemble performance. To obtain the optimal ensemble, the proposed framework calls for investigating different fusion topologies and functions.

- **Fusion evaluation**: in addition to the predictive performance of the final ensemble(s), the proposed framework calls for evaluating other aspects, such as required computing resources, model selection and training time (needed to build all base models/classifiers and the fusion model/classifier), and prediction time (required to produce a final prediction by the ensemble for a new sample).

- **Experimental design**: in order to have an accurate, and fair, comparison during model selection, the proposed framework stresses the importance of taking into account different sources of variation, such as the choice of the testing/training datasets, the internal randomness of the training algorithm, the random classification error, and the randomness in parallel computing (if the model selection is performed on a cluster of computers/cores).

As secondary contributions, this thesis performed the following empirical studies:

- **The effectiveness of model diversity approaches**: the thesis investigated the effect of injecting model diversity at data, feature, or/and learning algorithm levels, on the performance of ensembles. To maximize the performance of an ensemble, the empirical results suggest the need to inject model diversity at all levels.
- **The application of ensemble systems to large/high-dimensional data applications**: the thesis validated the effectiveness of the proposed framework using two real-world, large/high-dimensional, regression and classification case studies. Ensembles were built in the regression case study to predict the stock market's short-term behavior following liquidity shocks, while in the classification case, to predict the biological response of molecules from their chemical properties.
- **Investigating the relationship between diversity metrics and ensembles performance**: in the classification case study, the thesis experimentally investigated the relationship of ten diversity metrics and the ensembles performance. Experimental results suggest a weak, and sometimes contradictory, relationship.
- **Evaluating the scalability of learning algorithms**: scalability is one of the key issues in large-scale data analysis and modeling. The thesis experimentally evaluated the scalability, based performance and computational cost, of several learning algorithms, applied to regression and classification tasks. Empirical results indicate significant differences between the algorithms, especially in term of computational cost.

6.2 Future Work

Data science is currently a hot area not only in the academia but also across many industries, ranging from government to biotech. Cutting-edge startups, as well as established tech companies and universities, are pioneering new, novel, and exciting ways to apply data science/machine learning to interesting problems [34]. In the context of this research, several issues should be investigated in the future, such as:

- **Combining the proposed framework with Spark framework:** Spark is an open source cluster computing framework, initially developed at the University of California, Berkeley [95], which can be used to apply machine learning algorithms to petabytes of data [1]. As shown by the experimental results, developing ensemble systems for large/high-dimensional data requires very powerful computing resources to handle the intensive computation involved in the model building phase (e.g., model selection and training). Having such a computing resource allows the application of ensemble systems to commercial big data applications in many industries.

- **Re-calibrating class probabilities before model fusion:** as shown in Section 5.7.2, some trained classifiers have poorly-calibrated class probabilities. The class probabilities produced by a classifier can be re-calibrated, to reflect the likelihood of the event seen in the data set, by training an additional classifier (such as Logistic Regression or Naive Bayes) to adjust the predicted probabilities [45]. Having well-calibrated class probabilities might prove to be worthy of consideration, especially when using these probabilities as predictors in the model fusion by model stacking, discussed in Section 5.8.3.

- **Applying ensemble systems to novel applications:** the concept of ensemble systems naturally lends itself to many novel applications, such as combining machine learning, behavioral analytics, and big data. Examples of such applications include assigning an insurance's risk index to a customer based on several factors, or real-time threat detection against insider and targeted outsider cyber attacks to protect intellectual property, trade secrets, and/or classified information [2].

[1]A petabyte (PB) is 10^{15} bytes of data, 1,000 terabytes (TB) or 1,000,000 gigabytes (GB)

[2]A commercial real-time threat detection platform is developed by the Canadian startup Interset https://www.interset.com/

References

[1] Tarek Abdunabi and Otman Basir. Architectural considerations for multi-asset, multi-strategy algorithmic trading systems. In *proceedings of the 22nd ISCA International Conference on Software Engineering and Data Engineering (SEDE 2013), September 25-27, 2013, Los Angeles, USA*. IJCA, 2013.

[2] Tarek Abdunabi and Otman Basir. An integrated multi-asset, multi-strategy algorithmic trading system. In *proceedings of the 22nd ISCA International Conference on Software Engineering and Data Engineering (SEDE 2013), September 25-27, 2013, Los Angeles, USA*. IJCA, 2013.

[3] Tarek Abdunabi and Otman Basir. Building diverse and optimized ensembles of gradient boosted trees for high-dimensional data. In *Cloud Computing and Intelligence Systems (CCIS), 2014 IEEE 3rd International Conference on*, pages 351–356. IEEE, 2014.

[4] Tarek Abdunabi and Otman Basir. Holonic intelligent multi-agent algorithmic trading system. *International Journal of Computers and Their Applications (IJCA)*, 21(1):1–8, 2014.

[5] Tarek Abdunabi and Otman Basir. Predicting a biological response of molecules from their chemical properties using diverse and optimized ensembles of stochastic gradient boosting machine. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 10–17. IEEE, 2014.

[6] Donald J Abraham. *Burger's medicinal chemistry and drug discovery.* Wiley Online Library, 2003.

[7] Douglas G Altman and J Martin Bland. Diagnostic tests 3: receiver operating characteristic plots. *BMJ: British Medical Journal*, 309(6948):188, 1994.

[8] Yaxin Bi. Analyzing the relationship between diversity and evidential fusion accuracy. In *Multiple Classifier Systems*, pages 249–258. Springer, 2011.

[9] Christopher M Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995.

[10] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[11] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Christopher D Brown and Herbert T Davis. Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 80(1):24–38, 2006.

[14] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[15] Lorenzo Bruzzone, Roberto Cossu, and Gianni Vernazza. Detection of land-cover transitions by combining multidate classifiers. *Pattern Recognition Letters*, 25(13):1491–1500, 2004.

[16] Harris K Butler. The relationship between diversity and accuracy in multiple classifier systems. Technical report, Defense Technical Information Center (DTIC), USA, 2012.

[17] Anne Magaly de Paula Canuto. *Combining neural networks and fuzzy logic for applications in character recognition.* PhD thesis, University of Kent at Canterbury, 2001.

[18] Jacob Cohen et al. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[19] David A Dickey. Introduction to predictive modeling with examples. *Raleigh, N. Carolina State U., NC*, 2012.

[20] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[21] Thomas G Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

[22] Robert PW Duin. The combining classifier: to train or not to train? In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 765–770. IEEE, 2002.

[23] Yuval Elovici, Bracha Shapira, and Paul B Kantor. Using the information structure model to compare profile-based information filtering systems. *Information Retrieval*, 6(1):75–97, 2003.

[24] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[25] M Fieschi et al. Context-sensitive medical information retrieval. *MedInfo*, page 282, 2004.

[26] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[27] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[28] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[29] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[30] Francesco Gargiulo, Claudio Mazzariello, and Carlo Sansone. Multiple classifier systems: theory, applications and tools. In *Handbook on Neural Information Processing*, pages 335–378. Springer, 2013.

[31] Jean Gordon and Edward H Shortliffe. The dempster-shafer theory of evidence. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, 3:832–838, 1984.

[32] Louis Guttman. Some necessary conditions for common-factor analysis. *Psychometrika*, 19(2):149–161, 1954.

[33] Stefan T Hadjitodorov, Ludmila I Kuncheva, and Ludmila P Todorova. Moderate diversity for better cluster ensembles. *Information Fusion*, 7(3):264–275, 2006.

[34] Laura Hamilton. Six novel machine learning applications, 2014. http://www.forbes.com/sites/85broads/2014/01/06/six-novel-machine-learning-applications/#2b9396f467bf. Accessed: 2016-03-11.

[35] Trevor Hastie, Robert Tibshirani, and Andreas Buja. Flexible discriminant analysis by optimal scoring. *Journal of the American statistical association*, 89(428):1255–1270, 1994.

[36] Douglas M Hawkins, Subhash C Basak, and Denise Mills. Assessing model fit by cross-validation. *Journal of chemical information and computer sciences*, 43(2):579–586, 2003.

[37] Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[38] Tin Kam Ho. Data complexity analysis for classifier combination. In *Multiple Classifier Systems*, pages 53–67. Springer, 2001.

[39] Ajay N Jain, Kimberle Koile, and David Chapman. Compass: predicting biological activities from molecular surface properties. performance comparisons on a steroid benchmark. *Journal of Medicinal Chemistry*, 37(15):2315–2327, 1994.

[40] Kaggle.com. Competition: Algorithmic trading challenge, 2011. https://www.kaggle.com/c/AlgorithmicTradingChallenge. Accessed: 2016-02-11.

[41] Kaggle.com. Competition: Predicting a biological response, 2012. http://www.kaggle.com/c/bioresponse. Accessed: 2016-02-11.

[42] Mohamed S Kamel and Nayer M Wanas. Data dependence in combining classifiers. In *Multiple Classifier Systems*, pages 1–14. Springer, 2003.

[43] Kenji Kira and Larry A Rendell. The feature selection problem: Traditional methods and a new algorithm. In *AAAI*, pages 129–134, 1992.

[44] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *Machine Learning: ECML-94*, pages 171–182. Springer, 1994.

[45] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, 2013.

[46] Ludmila Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, second edition, 2014.

[47] Ludmila I Kuncheva. That elusive diversity in classifier ensembles. mallorca, spain, 2003. In *Proceedings of 1st Iberian Conference on Pattern Recognition and Image Analysis*.

[48] Ludmila I Kuncheva. A theoretical study on six classifier fusion strategies. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):281–286, 2002.

[49] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[50] Louisa Lam. Classifier combinations: implementations and theoretical issues. In *Multiple classifier systems*, pages 77–86. Springer, 2000.

[51] William Leigh, Russell Purvis, and James M Ragusa. Forecasting the nyse composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: a case study in romantic decision support. *Decision Support Systems*, 32(4):361–377, 2002.

[52] Hwei-Jen Lin, Yang-Ta Kao, Fu-Wen Yang, and Patrick SP Wang. Content-based image retrieval trained by adaboost for mobile application. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(04):525–541, 2006.

[53] Charles X Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *KDD*, volume 98, pages 73–79, 1998.

[54] Yi Lu. Knowledge integration in a multiple classifier system. *Applied Intelligence*, 6(2):75–86, 1996.

[55] Oded Maimon and Lior S Rokach. Data mining by attribute decomposition with semiconductor manufacturing case study. In *Data mining for design and manufacturing*, pages 311–336. Springer, 2001.

[56] Paul Mangiameli, David West, and Rohit Rampal. Model selection for medical diagnosis decision support systems. *Decision Support Systems*, 36(3):247–259, 2004.

[57] J Kent Martin and Daniel S Hirschberg. *Small sample statistics for classification error rates I: Error rate measurements*. Information and Computer Science, University of California, Irvine, 1996.

[58] Eitan Menahem, Lior Rokach, and Yuval Elovici. Troika–an improved stacking schema for classification tasks. *Information Sciences*, 179(24):4097–4122, 2009.

[59] Christian Merkwirth, Harald Mauser, Tanja Schulz-Gasch, Olivier Roche, Martin Stahl, and Thomas Lengauer. Ensemble methods for classification in cheminformatics. *Journal of chemical information and computer sciences*, 44(6):1971–1978, 2004.

[60] Annette M Molinaro, Richard Simon, and Ruth M Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005.

[61] Robert Moskovitch, Yuval Elovici, and Lior Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics & Data Analysis*, 52(9):4544–4566, 2008.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[63] Pedro R Peres-Neto, Donald A Jackson, and Keith M Somers. How many principal components? stopping rules for determining the number of non-trivial axes revisited. *Computational Statistics & Data Analysis*, 49(4):974–997, 2005.

[64] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[65] Robi Polikar. Ensemble based systems in decision making. *Circuits and systems magazine, IEEE*, 6(3):21–45, 2006.

[66] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2014.

[67] Romesh Ranawana and Vasile Palade. A neural network based multi-classifier system for gene identification in dna sequences. *Neural Computing & Applications*, 14(2):122–131, 2005.

[68] Romesh Ranawana and Vasile Palade. Multi-classifier systems: Review and a roadmap for developers. *Int. J. Hybrid Intell. Syst.*, 3(1):35–61, 2006.

[69] Greg Ridgeway. Generalized boosted models: A guide to the gbm package. *Update*, 1(1), 2007.

[70] Marko Robnik-Šikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICMLÂŠ97)*, pages 296–304, 1997.

[71] Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, 53(1-2):23–69, 2003.

[72] Lior Rokach. Mining manufacturing data using genetic algorithm-based feature set decomposition. *International journal of intelligent systems technologies and applications*, 4(1-2):57–78, 2008.

[73] Lior Rokach. Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography. *Computational Statistics & Data Analysis*, 53(12):4046–4072, 2009.

[74] Lior Rokach and Oded Maimon. Data mining for improving the quality of manufacturing: a feature set decomposition approach. *Journal of Intelligent Manufacturing*, 17(3):285–299, 2006.

[75] Lior Rokach, Oded Maimon, and Mordechai Averbuch. Information retrieval system for medical narrative reports. In *Flexible Query Answering Systems*, pages 217–228. Springer, 2004.

[76] Lior Rokach, Roni Romano, and Oded Maimon. Negation recognition in medical narrative reports. *Information Retrieval*, 11(6):499–538, 2008.

[77] Fabio Roli, Giorgio Giacinto, and Gianni Vernazza. Methods for designing multiple classifier systems. In *Multiple Classifier Systems*, pages 78–87. Springer, 2001.

[78] Dymitr Ruta and Bogdan Gabrys. Classifier selection for majority voting. *Information fusion*, 6(1):63–81, 2005.

[79] Alon Schclar, Alexander Tsikinovsky, Lior Rokach, Amnon Meisels, and Liat Antwarg. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems*, pages 261–264. ACM, 2009.

[80] AMANDA J C SHARKEY. On combining artificial neural nets. *Connection Science*, 8(3-4):299–314, 1996.

[81] Amanda JC Sharkey. Types of multinet system. In *Multiple Classifier Systems*, pages 108–117. Springer, 2002.

[82] Amanda JC Sharkey. *Combining artificial neural nets: ensemble and modular multi-net systems*. Springer Science & Business Media, 2012.

[83] Amanda JC Sharkey and Noel E Sharkey. Combining diverse neural nets. *The Knowledge Engineering Review*, 12(03):231–247, 1997.

[84] Amanda JC Sharkey, Noel E Sharkey, Uwe Gerecke, and Gopinath Odayammadath Chandroth. The âĂIJtest and selectâĂİ approach to ensemble combination. In *Multiple Classifier Systems*, pages 30–44. Springer, 2000.

[85] Catherine A Shipp and Ludmila I Kuncheva. Relationships between combination methods and measures of diversity in combining classifiers. *Information fusion*, 3(2):135–148, 2002.

[86] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[87] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. Multi-label classification methods for multi-target regression. *arXiv preprint arXiv:1211.6581*, 2012.

[88] Aik Choon Tan, David Gilbert, and Yves Deville. Multi-class protein fold classification using a new ensemble machine learning approach. *Genome Informatics*, 14:206–217, 2003.

[89] Dacheng Tao, Xiaoou Tang, Xuelong Li, and Xindong Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1088–1099, 2006.

[90] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. In *Neural Nets*, pages 3–20. Springer, 2002.

[91] Vladimir Vapnik. *The nature of statistical learning theory.* Springer Science & Business Media, 2013.

[92] Terry Windeatt. Diversity measures for multiple classifier system analysis and design. *Information Fusion*, 6(1):21–36, 2005.

[93] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[94] Lei Xu, Adam Krzyżak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3):418–435, 1992.

[95] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010.