An Investigation of Statistical Learning Curves: Do We Always Need Big Data ?

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Masters in Statistics
by
Yang Li

University of Canterbury
2017

Abstract

The rapid revolutionary rapid Big Data technology has attracted increasing attention and widely been used in many industries. It is not only benefiting our life dramatically, but also posing new challenges to us at the same time. In many situations, dealing with these big and complex data can extremely difficult. However, do we really always need big data?

This thesis attempted to investigate whether do we need a large dataset to build a model with acceptable accuracy, how the number of observations affect the performance of statistical predictive methods and use learning curves to describe this relationship. Some popular statistical learning methods were considered and applied on 3 large datasets. An efficient parallel coding strategy in R was also provided.

Table of Contents

Acknowledgments

I would like to my supervisors Dr. Marco Reale and Dr. Blair for their endless support, guidance and encouragement during my study. I really appreciate all their help to make me become better not only statistically but also non-statistically.

In the past two year, I had a very good time studying in the School of Mathematics and Statistics at the University of Canterbury. I would like to thank all the members in our department, especially my amazing lecturers Professor Jennifer Brown, Dr Elena Moltchanova, Dr Daniel Gerhard, Dr Carl Scarrott, Richard Penny and Patrick Graham, from who I learn a lot of knowledge, and my lovely classmates. I also appreciate Paul Brouwers and Steve Gourdie for all IT support.

I would like to thank my parents and my girl friend. I appreciate their understanding and support.

Chapter I

Introduction

1.1 Big Data

This is an era of data, where many human activities and natural phenomena can be described and analysed digitally [5]. Modern technology and digital devices are being widely used to increase the collection, storing and exchanging of data. This gives us a great opportunity to understand ourselves and the world better than ever before. Organizations are realizing the value of data in marketing analysis, strategy design and decision-making. For instance, by tracking customers' purchase and website visiting records, Amazon recommends goods to targeted users, which makes their advertisements more effective [7]. Rob (2014) outlined some former and currently being built "smart cities", equipped with digital devices and instruments, where data from human activities, the environment, and other sources are collected and analysed to improve everyday life, such as traffic guidance, monitoring the source of air pollution, etc [8].

The revolutionary collection, storing, transferring and analysis methods for large amounts of data is firstly called "big data" by John Mashey in the 1990s. Since then, "big data" is a term used for datasets which are so large or complex that traditional data management and analysis tools or methods are inefficient and insufficient to process them. Doug Laney (2001) summarized big data in three dimensions: volume, velocity and variety.

Volume refers to the size of data. A survey conducted by IBM in 2012 showed that datasets over one terabyte were considered to be big data by the majority of respondents [16]. For instance, Walmart collects nearly more than 2.5 petabytes (10^{15} bytes) of data per hour from customer transactions [11]. Beaver (2010) pointed out that Facebook processed over 1 mil-

lion photographs per second [15], and over 260 billion photos have been stored using over 20 petabytes of storage space [16]. However, the definitions of big data volume are relative and change overtime. The world's capability of storing data has nearly doubled every 40 months from the 1980s [49], and IBM estimated 2.5 quintillion (10^{18} bytes) of data is created everyday in various forms [48]. What are considered big data today may not be in the future, because both the volume of data and the processing tools are growing fast.

Velocity refers to the speed at which data are generated and required to be analysed. For some organizations, the velocity of data collection can be more important than the volum [11]. The real-time data generated from widely used digital devices such as smartphones and sensors can be used to generate real-time personalized customer offers. For instance, Alibaba use real time analysis tools to evaluate the up-to-date transaction streams of enterprises in order to decide whether to approve the loans to them [17]. In the smart cities described by Rob (2014), real time city data analysis is used to monitor and adjust the traffic flow [8].

Variety refers to the various sources and structures of data. A study conducted by Cukier in 2012 showed that the proportion of tabular data in spreadsheets and database is only about 5%. The advanced digital tool make many companies and organizations are able to generate various forms of non-tabular data, such as text, images, videos, etc. In these situation, specific machines or tools are required to organize these data to make them recognizable by data analysis tools [16].

Although some of these massive and complex data can be processed by super computers, some of them can no longer be processed or stored on a single computer. For instance, Memorial Sloan-Kettering Cancer Center used IBM's Watson supercomputer to construct a real-time cancer diagnose and treatments suggestion system [18]. However, supercomputers are not available to most organizations and industries because of their expensive purchase and maintenance costs. Therefore, big data poses real demands and challenges.

1.2 Why Statistical Learning?

In data analysis, the two main goals are to provide insight into the relationships between the features in the data, and to construct accurate models for predictive purposes [12]. However, large datasets can have many observations, many features and, hence, potentially more complex patterns. In these situations, classical statistical methods may require unrealistic properties and are often inefficient [14]. For instance, some classical statistical techniques concentrate more on testing hypotheses, where some subjective assumptions might be needed [18].

To extract the valuable information from the big datasets, it is necessary to develop more effective and powerful ways to let the data speak. Statistical learning is a set of applied statistics methods that help with learning the pattern and relation of data. These methods, which are computationally effective and well performed in classification, combine standard statistical thinking with machine learning idea [18]. Statistical learning techniques can be divided into two categories: supervised learning and unsupervised learning.

In unsupervised learning, the task is to explore unknown patterns or structures in the data, such as clustering, anomaly detection, etc. For instance, a mobile phone company might try to identify groups of users with similar usage in data, phone call time and text. Then they can design "combos" for each group to make deals targeted more precisely for their customers.

In supervised learning, the goal is to train a predictive model and evaluate its predicting performance using training data. For instance, suppose we are interested in how much new Amazon users are likely to spend on-line and we have information related it, such as the number of hits by the user, age, gender, etc. Using the purchase history and relevant information of former known customers, we can construct a model to describe the relationship between the customers' expenditure and other relevant features. Then, the new Amazon users' expenditure can be estimated using the other relevant features. In this example, the user's expenditure is also often called the *response* or *dependent variable*, and is usually denoted as Y. The known features such as the number of hits by the user, age and gender are also called *independent variables* or *predictors*, and are normally denoted using X, with a subscript to distinguish them. Then, the number of hits by the users can be denoted as X_1, the customers' age can be denoted as X_2 and the customers' gender can be denoted as X_3. The set of former customers, whose predictors and responses are known, is normally called *training data*. The set of new

customers, whose responses are unknown, is usually called *test data*. The goal is to construct a good model that can be used to predict unknown responses in test data, in other words, to find a function $\hat{f}$ such that $Y \approx \hat{f}(X)$. The question is how do we construct such models? When the data is huge and complex, the relationships between the response and predictors are more likely to be complex too. As a result, statistical learning methods, such as artificial neural networks, decision trees, random forest, supporting vector machine, etc. become popular because they can construct flexible models to capture the relationships between the features [13] [2].

1.3 About This Thesis

This thesis focuses on supervised learning. It can be time consuming and computationally expensive (or even impossible) to apply some supervised learning methods to large datasets. The main purpose of this thesis is to investigate how the number of observations affect the performances of different supervised learning algorithms. Furthermore, we investigate whether all the observations in large datasets are necessary to construct models with acceptable accuracy. The study consistsl of:

- testing and comparing the performances of different supervised learning algorithms on large datasets,
- using parallel computing techniques to implement algorithms effectively and
- comparing the computation time and memory usage of different supervised learning algorithms on large datasets.

In Chapter 2, a literature review of some popular supervised learning algorithms is given. We consider model selection methods and the learning curves in Chapter 3. A simulation study comparing different algorithms on three datasets is given in Chapter 4. Concluding remarks and future research is given in Chapter 5.

Chapter II

Algorithms

There is no one statistical approach will outperform all others in all situations [2]. It is difficult to select the most efficient method without a basic understanding of their key features [2]. In this chapter, we consider the basic features of some popular statistical learning techniques, which are K-Nearest Neighbours, Discriminant Analysis, Decision Trees, Bagging Trees, Random Forests, Boosting Trees and Support Vector Machine [1].

2.1 K*–Nearest Neighbours*

K-Nearest Neighbours (KNN) is one the oldest and simplest statistical learning methods [22], introduce by Cover and Hart (1967). In many areas, such as hand-writing and face recognition, KNN is a effective method, especially when combined with domains knowledge [23][24]. KNN makes prediction using a neighbourhood of known observations. The tuning parameter K is the number of neighbours considered, which is used to determine how many observations are used to make the prediction. The neighbourhood can be defined in several ways using the Manhattan distance, Euclidean distance and supreme distance, for example, and is sensitive to the way of distance measured [23]. Cross-validation can be applied to estimate the optimal K, which is the value with the least cross-validation error.

Denote a training dataset with p predictors, $(\mathbf{x}_i,\, y_i)$, $i = 1, 2, ..., n$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip})$ are the predictors of the i^{th} observation, and y_i is the class label. The prediction of an observation $\mathbf{x}_j$ in the test data is made by the training observations that are "close" to it. For classification problems, it is

$$\hat{f}\left(\mathbf{x}_j\right) = \text{Majority Voting}\left(\{y_i | \mathbf{x}_i \in \mathcal{N}_j\}\right),$$

where $\mathcal{N}_j$ is the subset of K observations in training data that are nearest to $\mathbf{x}_j$. For regression problems, it is

$$\hat{f}(\mathbf{x}_j) = \frac{1}{K} \sum_{i:\mathbf{x}_i \in \mathcal{N}_j} y_i,$$

where $\mathcal{N}_j$ is the subset of K observations in training data that are nearest to $\mathbf{x}_j$.

The distances can be measured by multiple ways. For example, the $\mathbb{L}^p$ distances between two points, $(x_{i1}, x_{i2}, ..., x_{ip})$ and $(x_{j1}, x_{j2}, ..., x_{jp})$ is,

$$L^d = \left(\sum_{k=1}^{p} |x_{ik} - x_{jk}|^d \right)^{\frac{1}{d}}.$$

When $d = 1$, we have Manhattan Distance, when $d = 2$, we have Euclidean Distance and when $d \to \infty$,

$$\lim_{d \to \infty} L^d = \max_k \left(|x_{ik} - x_{jk}| \right),$$

is called Chebyshev Distance. In different situations, we need to choose the best distance measuring methods. However, this approach can be computationally expensive without any previous knowledge about the potential range [23]. In this thesis, we considered the Euclidean Distance and each feature is standardized to mean 0 and variance 1, so that every predictor has the same effect.

The the number of neighbours K, controls the model's flexibility. When K is small, every estimate is made by a small number of training observations. So, the model is flexible because the decision boundary could change dramatically with small changes in training data. When K is big, the model is relatively stable, because all estimates are made using many observations. Flexible models have small bias and large variance [1]. More details about model bias and variance will be discussed in Chapter 3.

2.2 Discriminant Analysis

Discriminant Analysis is a parametric statistical learning algorithms model each class using discriminant functions $\Phi_k(\mathbf{x}), k = 1, 2, 3, ..., K$, and classify the observation $\mathbf{x}$ to the class with the biggest discriminant functions. The first discriminant analysis approach, Linear Discriminant Analysis (LDA), is a generalization of Fisher Linear Discriminant Analysis, proposed by R. A. Fisher (1936). It is widely used in pattern recognition, feature extraction and classification [3]. Comparing to logistic regression, the discriminant analysis is more stable when the response variable is well separated [1]. Secondly, in logistic regression, the response variable is the odds. Then, pivots are required if the response has multiple levels, which makes the model to be complex and less effective. However, discriminant analysis are more straightforward and easy to apply in this case.

Suppose a K levels classification problem with p predictors, where $2 \leq K < \infty$. Denote the predictors are $\mathbf{x} \in \mathbb{R}^p$, and the response variable is Y, where $Y = 1, 2, ..., K$. We set the discriminant functions to the posterior probabilities of Y given X, then

$$\Phi_k(\mathbf{x}) = Pr(Y = k|X = \mathbf{x}),$$

where $k = 1, 2, ..., K$. Denote the predictors of the k^{th} level follows the distribution

$$f_k(\mathbf{x}) = Pr(X = \mathbf{x}|Y = k),$$

and $\pi_i = Pr(Y = i)$ to be the prior probability of k^{th} level, then from the Bayesian theorem, we have,

$$Pr(Y = k|X = \mathbf{x}) = \frac{f_k(\mathbf{x})\pi_k}{\sum_{i=1}^{K} f_i(\mathbf{x})\pi_i}.$$

So, the posterior probability can be figured out, given π_i and $Pr(X = \mathbf{x}|Y = i)$, $i = 1, 2, ..., K$. The unlabeled observation, whose values of predictors are $\mathbf{x}$, can be classified to the i^{th} level, if $Pr(Y = i|X = \mathbf{x}) > Pr(Y = j|X = \mathbf{x})$, $j = 1, 2, ..., i-1, i+1, ..., k$. In applications, π_i can be estimated by $\hat{\pi}_i$, the proportion of the observations labelled the i^{th} level in the training data. The posterior probability, $Pr(Y = i|X = \mathbf{x})$, mostly depends on the class density, $f_i(\mathbf{x})$. There are multiple ways to estimate of the class density, such as Linear Discriminant Analysis(LDA)

and Quadratic Discriminant Analysis(QDA), Kernel Density Classification, Naive Bayesian Classifier, etc. In this thesis, we considered LDA and QDA, where the class density, $f_i(\mathbf{x})$, is assumed to be normal distributed.

2.2.1 LDA

In both LDA and QDA, the class density are supposed to be Gaussian, then

$$f_i(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma_i}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right),$$

where $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the mean vector and covariance matrix of the multivariate normal density of the i^{th} class. Therefore, we can figure out all probabilistic features given the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ for all class. In LDA, we assume the covariance matrix of each class are same, so $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$ for all $i = 1, 2, ..., K$. For two class problems, comparing the posterior probability of each class is equivalent to considering the log of ratio between them. So, we find that

$$\begin{aligned}\log\frac{Pr(Y=i|X=\mathbf{x})}{Pr(Y=j|X=\mathbf{x})} &= \log\frac{f_i(\mathbf{x})\cdot\pi_i}{f_j(\mathbf{x})\cdot\pi_j}\\ &= \log\frac{\pi_i}{\pi_j} - \frac{1}{2}(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)+\\ &\quad \mathbf{x}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)\end{aligned}$$

is a linear function of $\mathbf{x}$. It suggests that the decision boundary between these two classes is a straight line if the number of predictors $p = 2$; or a hyperplane if the number of predictors $p \geq 3$, whose norm is determined by $\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$. The prior probability, π_i and π_j, are estimated using the proportion of each class in the training data. The mean of each class, $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$, are estimated by the mean of each class in the training data $\hat{\boldsymbol{\mu}}_i$ and $\hat{\boldsymbol{\mu}}_j$. The covariance matrix $\boldsymbol{\Sigma}$ is estimated by the pooled variance of both classes $\hat{\boldsymbol{\Sigma}}$.

2.2.2 QDA

In QDA, the class densities are also assumed to be Gaussian distributed, but with different covariance matrix for each class, so they are estimated separately. So the mean of each class, $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$, are estimated by the mean of each class in the training data $\hat{\boldsymbol{\mu}}_i$ and $\hat{\boldsymbol{\mu}}_j$. The prior probability for each class, π_i and π_j, are estimated using the proportion of each class in the training data. The variances of each class $\boldsymbol{\Sigma}_i$ and $\boldsymbol{\Sigma}_k$ are estimated separately. Similarly to LDA, we can show the boundary of QDA classifier is determined by a quadratic of $\mathbf{x}$. That means the decision boundary made by QDA will be a curve or a surface for two classes problems.

Comparing with LDA, QDA is more flexible, but more parameters required to be estimated. In classification problem with K levels and p predictors, there are $p(p+1)/2$ parameters in each covariance matrix in total. Then, there are $Kp(p+1)/2$ parameters in QDA, comparing to $p(p+1)/2$ in LDA. As a result, QDA is potentially more flexible, and has a higher variance and lower bias, comparing to LDA. So, LDA is recommended in situations where the amount of observations is small and the difference in variance of each class are not significantly big [2].

2.3 Decision Tree

Decision trees is a set of powerful and efficient statistical learning techniques widely used both in regression and classification problems [26]. As the algorithm can be described as tree-like graph, it got its name. The basic idea of a decision tree is to split the predictor space into simple regions using the training data, and the prediction is made by the mean (for regression problems) or majority voting (for classification problems) of the observations that are in the same regions with the testing observation. One main advantage of decision trees is they are straightforward and visible, so it is easy to understand and interpret. Secondly, decision tree is a non-parameter method which can create classifiers with flexible boundaries to capture the patterns of data automatically. Thirdly, the splitting rules of decision tree computationally simple.

2.3.1 Classification and Regression Trees

In the following part, we introduce the procedure of classification and regression trees (CART) [27]. In CART, there are two major elements in the tree, one is node, which is the condition of the split, and the other one is terminal node (or leaf), which is the predicting regions. The node

is the condition applied in each splitting step. We split the predictor space until some stopping rule satisfied where a terminal node reach. Then, the predictions are made by the response of the observations in each terminal node. theoretically, there are various of ways to construct a split condition. However, it is impossible to consider every situation in practice. So, for simplicity and easy interpretation purpose, in CART, just the vertical conditions are applied the vertical conditions at each node, and complex classifiers are created by the combination of further partitions.

2.3.2 Construction Process

Suppose the variable of interest is Y and there are p predictors, $X_1, X_2, ..., X_p$, in the model. Given a training data, we consider a vertical binary split condition, 'Whether X_{n_1} is smaller than a specific value, c_1', where $n_1 \in \{1, 2, ..., p\}, c_1 \in \mathbb{R}$. Then, the training data is split into two subsets, $A_1^1 = \{\mathbf{x}|X_{n_1} > c_1\}$ and $A_2^1 = \{\mathbf{x}|X_{n_1} \leq c_1\}$. If any of the subset is a terminal node, we make the prediction by the majority voting of the training data in the subset for classification problems, or the mean of the training data in the subset for regression problems. The splitting is decided by choosing n_i, the predictor to use, and c_1, the cut-off point, which minimize the loss function. For example, for a regression problem, and the sum of squared error was used as the loss function. Then, at each node, the splitting was determined by

$$\{\hat{n_1}, \hat{c_1}\} = \min_{n_1, c_1} \left\{ \sum_{i:\mathbf{x}_i \in A_1^1} (y_i - \hat{y}_{A_1^1})^2 + \sum_{i:\mathbf{x}_i \in A_2^1} (y_i - \hat{y}_{A_2^1})^2 \right\},$$

where $\hat{y}_{A_1^1}$ and $\hat{y}_{A_2^1}$ are the means of the training observations in region A_1^1 and A_2^1 separately. For classification problems, the loss function is normally supposed to be mis-classification rate, but in tree algorithm, it is not sensitive enough, so Gini index and cross-entropy are used to measure the error rate for each region [1]. For K classes problems, denote $\hat{p}_k$ represent the rate of k^{th} class of the training observations in the specific region, then Gini index is

$$Gini = \sum_{k=1}^{K} \hat{p}_k(1 - \hat{p}_k).$$

From the equation above, we can see that Gini index is small when $\hat{p_k}$ is close to 0 or 1. Gini index is also called Gini impurity, used to measure the node purity, and regarded as a measurement which in favour of pure class splitting. An alternative way to measure the mis-classified rate is cross-entropy, given by

$$CE = -\sum_{k=1}^{K} \hat{p}_k \log \hat{p}_k.$$

It is showed both Gini index and cross-entropy are more sensitive to changes of class proportion in the node than mis-classification rate [1].

The tree is built by repeatedly further splitting the predictor regions until some stopping rules satisfied. There are multiple criterion are used as the stopping rule. For example, the minimum number of a node, which means splitting only can be made when the number of observation in that node is bigger than a pre-specified number. An alternative way is to setting a minimum purity of the terminal nodes, so the node will be split further if it is not pure enough. We continue split the node until all sub-nodes satisfy the stopping rules. Denote the tree classifier as

$$T(X, \Theta),$$

where Θ is a set of parameters, representing the splitting variable and the cut-off points used at each node. This tree construct process is called top-down, greedy approach which is also known as recursive binary splitting [2]. The 'top-down' means the tree splitting process start at the full predictor space which is a single node (or the root of the tree). The algorithm is greedy because we just consider the best split at specific node in the building process without caring about the previous splitting steps. As a result, the solution given by the algorithm is a series of local optimization at each node, rather than a global optimization.

To improve the predicting ability, we also consider the CART algorithm from a view of model complexity. Generally, we can get a much flexible model if more splitting are allowed, so a common way to measure the complexity of a tree is its depth, which is the maximum number of steps taken to split from the root to a terminal node. In practice, the widely used building tree strategy is to grow a big tree, and then using the cross-validation method to choose the best tree depth by pruning the tree. For classification problems, suppose we trained a big tree with M leafs and use Gini index to measure the purity, we introduce a tuning parameter α into the loss function

$$\sum_{m=1}^{M}\sum_{k=1}^{K}\hat{p}_k(1-\hat{p}_k)+\alpha|M|,$$

to control model's complexity. When α is small, the impurity of each leaf will dominate the loss function, the model will tend to be more complex to smaller the impurity for each leaf. If α is big, the model will suffer much cost of its complexity, which leads to model will tend to be less flexible. In practice, we use cross-validation to choose best α which is give best predicting testing error, thereby we can find the best depth of tree according to the tuning parameter, α.

2.3.3 Bagging Trees

In this section, we discuss the Bagging Trees, short for Bootstrap Aggregating Trees, introduced by Leo Breiman (1994). It is an ensemble statistical learning techniques used to improve a model's stability. Denote the train dataset is $\mathrm{Z} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$, and a classifier constructed using Z is $\hat{F}_{\mathrm{Z}}(\mathbf{x})$. To improve the performance of $\hat{F}_{\mathrm{Z}}(\mathbf{x})$, a set of bootstrapped samples, $\mathrm{Z}_1, \mathrm{Z}_2, ..., \mathrm{Z}_B$, are generate by repeatedly sampling $\mathbf{Z}$ of size n with replacement for B times. Next, these bootstrapped samples are separately used to build models, $\hat{F}_{\mathrm{Z}_1}(\mathrm{x}), \hat{F}_{\mathrm{Z}_2}(\mathbf{x})$,...., $\hat{F}_{\mathrm{Z}_B}(\mathbf{x})$. The averaging models of these bootstrapped models is called the Bagging model. The variance of the original model is reduced by applying the Bagging process. So, the Bagging models has better performance. For regression problems, the outcome of Bagging model is

$$\hat{F}_{bagging}(\mathbf{x}) = \frac{1}{B}\sum_{i=1}^{B}\hat{F}_{\mathrm{Z}_i}(\mathbf{x}).$$

For classification problems, the prediction outcome by Bagging model is given by majority voting,

$$\hat{F}_{bagging}(\mathbf{x}) = \text{Majority Voting}\left(\{\hat{F}_{\mathrm{Z}_i}(\mathbf{x})|\mathrm{Z}_i = 1, 2, ..., B\}\right).$$

The Bagging Trees is an approach applying the Bagging method on trees. We first generate B bootstrapped samples and use them to fit B Classification or Regression Trees, the Bagging Trees is the aggregating of these bootstrapped trees. In the construction process, each tree is fully grown to capture the structure of data as much as possible. That makes each tree has a small bias and large variance. However, the variance of final model can be controlled by taking the average of the classifiers in the later step. So the original model has been improved. Suppose the variance of each of the constructed trees is σ^2, and they are independent, it is easy to show that the variance of Bagging trees, based on B bootstrapped samples, is $\frac{1}{B}\sigma^2$. However, all tree are built using bootstrapped samples, which are sampled with replacement, some of the observations have been repeatedly used in each tree. As a result, they are correlated. Suppose the correlation between the trees is ρ, then the variance of the bagging model is

$$\begin{aligned}
\mathrm{V}\left(\hat{F}_{bagging}(\mathbf{x})\right) =& \mathrm{V}\left(\frac{1}{B}\sum_{i=1}^{B}\hat{F}_{\mathrm{Z}_i}(\mathbf{x})\right) \\
=& \frac{1}{B^2}\left[B \times \mathrm{V}\left(\hat{F}_{\mathrm{Z}_i}(\mathbf{x})\right) + (B^2 - B) \times \mathrm{Cov}\left(\hat{F}_{\mathrm{Z}_i}(\mathbf{x}), \hat{F}_{\mathrm{Z}_j}(\mathbf{x})\right)\right] \\
=& \frac{1}{B^2}\left[B \times \mathrm{V}\left(\hat{F}_{\mathrm{Z}_i}(\mathbf{x})\right) + (B^2 - B) \times \rho \times \mathrm{V}\left(\hat{F}_{\mathrm{Z}_i}(\mathbf{x})\right)\right] \\
=& \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.
\end{aligned}$$

We can see that when the number of bootstrapped samples B increases, the second term of the equation will be smaller, so the overall variance of the bagging model will be smaller. The idea in Random Forest is to improve the model by reducing, ρ, the correlation between the trees, therefore the variance of the ensemble model will be smaller.

Not all observations of the original dataset are not used in every model construction. In original dataset, the probability that the ith observation was sampled in the bth bootstrap sample is

$$P(\text{observation } i \in \text{bootstrap sample } b) = 1 - (1 - \frac{1}{n})^n$$

Then,

$$\begin{aligned}
\lim_{n\to\infty} 1 - (1 - \frac{1}{n})^n &= 1 - \lim_{n\to\infty} (1 - \frac{1}{n})^n \\
&= 1 - \lim_{n\to\infty} (1 + \frac{1}{n})^{-n} \\
&= 1 - e^{-1} \\
&\approx 0.632
\end{aligned}$$

It suggests that each tree make use of about $\frac{2}{3}$ of the total observations in construction process when the number of observations in the training set is sufficient big. That brings additional benefit, because the rest $\frac{1}{3}$ observations can be used to evaluate the ensemble model and estimate its prediction error, this error is called out-of-bag error [34]. It is showed that the out-of-bag error is almost identical to the estimated error given by leave-one-out cross validation, which is too optimistic comparing with k-fold cross-validation [35]. One advantage of out-of-bag error is that they can be measured at any stage of the model construction, rather than when the whole bagging process finished. So, they provide a very good version of model performance during the building process. Because the the Bagging Tree has been improved by the Random Forest approaches. So, more details of both algorithms will be discussed in the next section.

2.3.4 Random Forest

The original Random Forest algorithm was firstly proposed by Ho [36] and extended by combining with the idea of Bagging by Leo Breiman in 2001 [37]. Though decorrelation the bagged trees, Random Forest is one of the few robust algorithms that could handle dataset involving large number of predictors. Breiman(2001) pointed out that Random Forest is relative robust to the tuning parameters setting, so it is also called an " off-the-shelf " algorithm which is easy to be applied even for a new learner. Thirdly, they are easy to be applied in a parallel computing environment. In this section, we will outline the difference between the Bagging and Random Forest, and the issues involved.

Random Forest is generalization of Bagging trees. The only difference between Random Forest and Bagging trees is that only a random sample of m predictors are considered as the candidates at each split in trees in Random Forest. In other word, the best split variable at every node is chosen from m candidate predictors, which is a random sample of all predictors. Therefore, the parameter, m, is the main source (the other brought from bootstrap process) of the randomness in the algorithm. If m is big, the randomness of each tree is less, and the trees are much correlated. When m is equal to the number of predictors in the model, the Random Forest degrade to Bagging Trees. On the other hand, when m is small, the number of candidates predictors at each node is small, then, the splits will more likely to be determined by the random samples of candidate predictors. As a result, the correlation between the trees are small. Therefore, the main tuning parameter in Random Forest is m, the number of predictor considered at each split.

In Random Forest, each predictor's importances are measured in the ensemble model, which shows their overall contribution to the improvement of specific split-criterion. For example, if we use Gini index as the measure of impurity of the trees. Then, at each split, the variable used and the reduction of Gini index by the split will be recorded. The variable importance of the predictor is their accumulated reduction of Gini index in all trees in the forest.

In application of Random Forest, there are 3 main parameters involved, which are the number of candidates at each split ($mtry$), the number of trees ($ntree$) and the sizes of terminal nodes ($nodesize$). It is efficient to tuning the parameters in Random Forest to get the best predict model by minimizing the out-of-bag error. If there are p predictors in the training dataset, the default recommended value for $mtry$ is $\sqrt{p}$ and the minimum node size in 1 for classification

problems, and the recommended value for $mtry$ is $p/3$ and the minimum node size is 5 for regression problems [1]. Díaz-Uriarte and Alvarez de Andrés (2006) investigated the a range of $mtry$ from $\sqrt{p}$ to $13\sqrt{p}$ and found the larger $mtry$ make the variance importance more reliable. Goldstein et al.(2010) suggested that bigger $mtry$ might be needed when the number of predictor is large. For the number of trees in the model, $ntree$, larger value is always better [37]. Because the number of tree determines the number of bagging, it has been shown bigger number of bagging lower the variance of the ensemble model, without worrying about overfitting the data. When choosing the number of trees needed, out-of-bag error can be used as the only measure [29]. Normally, we can stop building more new trees when the out-of-bag error is flat. For example,we can stop building additional trees if the difference between maximum and minimum of out-of-bag error made by the last 100 trees is less than a positive number C.

2.4 Boosting Trees

Boosting algorithm is a set of statistical learning techniques generalized from an approach called "AdaBoost.M1", initially developed by Freund and Schapire (1996) [43]. The main idea of boosting is to build a "powerful " ensemble model by combining a series of "weak" classifier. Because every new classifier is constructed according to the information of existing ensemble model, those sequence of weak classifiers are highly related. A weak classifier is one which is just slightly better perform than a random classifier. (The random classifiers are constructed by using no information related to the variable of interest. So the performance of random classifiers is same as the random guessing.) Then, every newly built classifier just slightly improve the existing model, in other word, the model 'learn' the data slowly. However, combining enough number of weak classifiers, the ensemble model will be very powerful.

2.4.1 AdaBoost.M1.

AdaBoost, short for Adaptive Boosting, is the one of the most popular boosting algorithm. In this section, we present the construction process of AdaBoost. Suppose a two-class problem, the training dataset is

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\},$$

where $y_i \in \{-1, 1\}$ for all $i = 1, 2, ..., n$. We train a sequence of K weak classifiers using different weighted training data in each step. Suppose the weight of the i^{th} observation in k^{th} step

is $w_i^{(k)}$. To start with, we fit the first classifier, $G_1(X)$, where each observations in the training dataset are treated equally. So, the original weights of all observations are set to same, says $w_i^{(1)} = \frac{1}{n}$ for all $i = 1, ..., n$. Then, $G_1(X)$ is built in a way equals to one no weights applied, and we measure its performance using the error rate,

$$error_1 = \frac{1}{n} \sum_{i=1}^{n} I\left(y_i \neq G_1\left(\mathbf{x}_i\right)\right),$$

Where I is an indicator function, such that

$$I\left(y_i \neq G_1\left(\mathbf{x}_i\right)\right) = \begin{cases} 1, \text{if } y_i \neq G_1\left(\mathbf{x}_i\right) \\ \\ 0, \text{otherwise.} \end{cases}$$

Next, we build the second classifier $G_2(X)$, where the observations misclassified by $G_1(X)$ will be more concentrated by adding more weights on them. In AdaBoost Algorithm, when building the second classifier $G_2(X)$, all observations misclassified observations by $G_1(X)$ will be given new weights

$$w_i^{(2)} = w_i^{(1)} \times \frac{1 - error_1}{error_1},$$

for all i such as $G_1(\mathbf{x}_i) \neq y_i$; the weights of observations correctly classified by $G_1(X)$ will maintain the same($w_i^{(2)} = w_i^{(1)}$, for all i such as $G_1(\mathbf{x}_i) = y_i$). As long as every base classifier is better than random classifier, whose classification rate is 0.5, then $\frac{1-error_1}{error_1} > 1$, so it is ensured that the ones misclassified by $G_1(X)$ will obtain bigger weights in the next classifier. In the first step, the error rate for the first classifier $G_1(X)$ is just $error_1$, because the weights of every observations are set to same at the beginning, whereas their weights changed after the first step, so the error rate for second classifier, $G_2(X)$, will be a weighted error rate,

$$error_2 = \frac{\sum_{i=1}^{n} w_i^{(2)} I(y_i \neq G_2(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i^2}.$$

Algorithm 1 Adaboost.M1.

1: Initialize each observations' weights

$$w_i^{(1)} = \frac{1}{n}, i = 1, 2, ..., n$$

2: **for** k=1 to K **do**

3: Use the current weights, $w_i^{(k)}$, of each observation to build a classifier

$G_k(X)$

4: Calculate the weighted classification error rate of $G_k(X)$

$$error_k = \frac{\sum_{i=1}^{n} w_i^{(k)} I(y_i \neq G_k(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i^{(k)}},$$

5: Calculate the weight for $G_k(X)$

$$\alpha_k = \frac{1}{2} \log(\frac{1 - error_k}{error_k})$$

6: Update the weights of every observations for the next iteration

$$w_i^{(k+1)} = w_i^{(k)} \times \frac{1 - error_k}{error_k}$$

7: **end for**

8: Output ensemble classifier, $G(x) = \text{sign}\left\{\sum_{k=1}^{K} \alpha_k G_k(X)\right\}$

Then, we get a updated ensemble model by combining the weighted classifiers $G_1(X)$ and $G_2(X)$, denote

$$G_{(K)}(X) = \sum_{k=1}^{K} \alpha_k G_k(X).$$

Then, the ensemble classifier can be written as,

$$G(X) = \text{sign}\left\{G_{(2)}(X)\right\} = \text{sign}\left\{\sum_{k=1}^{2} \alpha_k G_k(X)\right\},$$

where $\alpha_k = \frac{1}{2}\log(\frac{1-error_k}{error_k})$, which are the weights for each base classifier. Through repeating this building process for K times, we get a sequence of 'weak' classifiers $G_1(X), ..., G_K(X)$, and their weights $\alpha_1, ..., \alpha_K$. At last, we combine these weighted base classifiers to get the final Adaboost ensemble classifier

$$G(X) = \text{sign}\left\{G_{(K)}(X)\right\} = \text{sign}\left\{\sum_{k=1}^{K} \alpha_k G_k(X)\right\}.$$

It is easy to find that the weights of each base classifier, α_k, is determined by classification performance of the k^{th} classifier $G_k(X)$. The base classifiers whose correct classification rate is higher will have a bigger influence on the final ensemble classifier. The setting of weight in AdaBoost algorithm make it equal to fitting a forward stepwise additive model using the exponential loss function

$$L(y, G_k(\mathbf{x})) = \exp\{-yG_k(\mathbf{x})\},$$

where the response $y \in \{-1, 1\}$. See the prove in appendix.

2.4.2 Boosting Trees

In this section, we present the Boosting Trees, where decision trees are used as those 'weak' base classifier, and the ensemble model is a combination of weighted decision tree built by using boosting algorithm. In CART, we discussed the constructing process of tree using the Gini index as the criterion, but in Boosting Trees, it will be a little different. For a two-class and exponential loss problem, suppose we wish to fit a classification tree, $T_k(X, \Theta_k)$, then, each tree minimize the sum of exponential loss,

$$\hat{\Theta}_k = arg \min_{\Theta} \sum_{i=1}^{n} w_i^{(k)} \exp\{-y_i T_k(\mathbf{x}_i, \Theta_k)\}.$$

However, same as the normal decision tree, global optimizing is computational expensive, so we also use the greedy approach, which is using the exponential loss as the splitting criterion to build each tree from top to down.

To improve the performance of trees, we build a large tree first and use the bottom-up procedure to prune the tree in CART. However, it is not necessary in Boosting Trees. Because each classifier is 'weak', so for each of them, just a slightly better than a random classifier, which means a small tree is good enough. Secondly, the trees are build sequentially in boosting, the performance of the ensemble model will be improved by further iterations. If we build a strong classifier, especially at the beginning, the performance of the ensemble model will potentially be worse [1]. Thirdly, the building big trees is computational expensive. So, in Boosting Trees, the depths of each tree are small. A good strategy is to specific a same depth value for every tree in boosting. It has been shown that the depth of tree is also limit the level of interaction of trees, the model works well using the depth between 4 and 8 normally, and using depth bigger than 6 will unlikely bring too much improvement [1].

To improve the performance of Boosting Tree, shrinkage technique is also applied. One of the most straightforward way is to scale down the contribution of each 'weak' classifier. For example, let's introduce a shrinkage parameter, γ, to the Boosting Tree algorithm, which makes the ensemble model

$$G(X) = G_{(k)}(X) = \gamma G_{(k-1)}(X) + G_k(X).$$

We can see the influence of each classifier, $G_k(X)$, is scaled down by γ. In practice, the shrinkage parameter, $\gamma \in (0, 1)$, is also known as the learning rate, which control the 'learning speed' of the boosting ensemble model. When the shrinkage parameter is small, each classifier's contribution is small and more classifiers are needed to capture the structure of data, so the 'learning speed' of the ensemble model is slow. On the contrary, if the shrinkage parameter is big, the 'learning speed' is fast and less iterations are needed.

In applications, slower learner, which is a boosting algorithm with lower learning rate and big number of iterations, gives better performance compared to faster learner, but the cost is computational time and resource. At the same time, with a lower learning rate, the ensemble model will overfit the data slower. For big datasets, or complex structured datasets, slow learning rates are recommended [33]. Normally, we specified a learning rate first, and then using model selection techniques to choose the number of iterations needed.

2.5 Support Vector Machine

In this section, we will discuss the support vector machine algorithm, which is a supervised learning technique firstly proposed by Corinna Cortes and Vapnik in 1990s [44][45][46], and it became popular since then [1].

2.5.1 Maximal Margin Classifier

In this section, we discuss the Maximal Margin Classifier which is a separating hyper-plane that farthest from the training observations.

Suppose a two classes problem, we have a training data set with p predictors, ($\mathbf{x}_i$, y_i), i=1,2,...,n, where $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip})$ are the predictors of the i^{th} observation, and $y_i \in \{1, -1\}$ is its the response. Then, the predictors span a p-dimensional spcae, $\mathbb{R}^p$. This space can be divided into two subspaces by a hyper-plane which is defined as

$$\{(x_1, x_2, ..., x_p) \in \mathbb{R}^p : x_1\beta_1 + x_2\beta_2 + ... + x_p\beta_p + \beta_0 = 0\},$$

where $\beta_0, \beta_1, \beta_2, ..., \beta_p$ are a sequence of real numbers. It is easy to show that $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_p)$ is the norm of the hyper-plane. When $\boldsymbol{\beta}$ is unit vector, the hyper-plane can be written as

$$\left\{\mathbf{x} \in \mathbb{R}^p : \mathbf{x}^T\boldsymbol{\beta} + \beta_0 = 0\right\},$$

where $\boldsymbol{\beta}$ represent the norm of the hyper-plane, and β_0 represent the location of the hyper-plane. In the following part, we denote $\boldsymbol{\beta}$ is a unit vector in $\mathbb{R}^p$, and β_0 is a real number, then, any hyper-plane in $\mathbb{R}^p$ can be described by $\{\boldsymbol{\beta}, \beta_0\}$. For any point $\mathbf{x} = (x_1, x_2, ..., x_p) \in \mathbb{R}^p$, its signed distance to hyper-plane $\{\boldsymbol{\beta}, \beta_0\}$ is

$$\mathbf{x}^T\boldsymbol{\beta} + \beta_0.$$

The points on one side of the hyper-plane $\{\boldsymbol{\beta}, \beta_0\}$ can be written as

$$\left\{\mathbf{x} \in \mathbb{R}^p : \mathbf{x}^T\boldsymbol{\beta} + \beta_0 > 0\right\}, \text{or}$$
$$\left\{\mathbf{x} \in \mathbb{R}^p : \mathbf{x}^T\boldsymbol{\beta} + \beta_0 < 0\right\}.$$

If $p = 2$, then $\mathbb{R}^p$ is normal 2-dimensional plane, and the hyper-plane is straight line. If $p = 3$, then $\mathbb{R}^p$ is normal 3-dimensional space, and the hyper-plane defined is normal plane. If $p > 3$, then this space is not visible.

In Maximal Margin Classifier, we suppose the train data is separable which means we can construct a separating hyper-plane, such that all the observations belong to different class are located on different side of the hyper-plane. Suppose $(\mathbf{x}_i, y_i)$ is the i^{th} the observation in the training data, where $y_i = 1$ or -1, representing the two response classes. The data is separable means existing separating hyper-planes $\{\boldsymbol{\beta}, \beta_0\}$ exists, such that

$$M_i = y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) > 0, \text{ for all } i = 1, 2, ..., n,$$

where M_i is a signed distance between the i^{th} observation and the separating hyper-plane

$\{\boldsymbol{\beta}, \beta_0\}$. It also suggest that every observation is on the correct side of the hyper-plane $\{\boldsymbol{\beta}, \beta_0\}$ in this case. Denote the width of margin of the separating hyper-plane $\{\boldsymbol{\beta}, \beta_0\}$ is

$$M = M(\boldsymbol{\beta}, \beta_0) = \min\{M_i\} > 0,$$

which is the minimum distance between observations and the hyper-plane. Then, every training observation is on the correct side, and at least M units away from the hyper-plane. The width of margin of a hyper-plane also describe how far the observations are separated by it.

When the training data is separable, there are always exist many hyper-planes separating the data perfectly. Maximal Margin Classifier is the one which divide the training data as much as possible. It can be written as

$$G(\mathbf{x}; \tilde{\boldsymbol{\beta}}, \tilde{\beta}_0) = \text{sign}(\mathbf{x}^T \tilde{\boldsymbol{\beta}} + \tilde{\beta}_0)$$
$$M(\tilde{\boldsymbol{\beta}}, \tilde{\beta}_0) = \max_{\boldsymbol{\beta}, \beta_0} \{M(\boldsymbol{\beta}, \beta_0)\},$$

which is the hyper-plane maximal its width of the margin with norm $\boldsymbol{\beta}$ and location β_0.

2.5.2 Support Vector Classifier

In many cases, the training data can not be separated by any hyper-plane perfectly, so the observations are overlap, which leads to there is no hyper-plane, such that all training observations belong to different class are located on different side. Then, Maximal Margin Classifier not exists in these situations. Another problem is the Maximal Margin Classifier is determined by the training observations on the margin, so the classifier will change dramatically if we just add a new observation in the margin or change the observations on the margin. It suggests that the Maximal Margin Classifier is very sensitive to some individuals in the training data.

To solve this problem, Support Vector Classifier is proposed, where some training observations on the wrong side of the margin of a hyper-plane is tolerated, and a measure of the confidence of every training observation being classified correctly is introduced. Denotes slack variables, $\epsilon_i \geq 0$, $i = 1, 2, ..., n$, representing the relative distance between the margin and the i^{th} observation, misclassified by the margin of the hyper-plane $\{\boldsymbol{\beta}, \beta_0\}$. Then, the signed

distance between the i^{th} training observation and the margin can be written as

$$y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) = M(1 - \epsilon_i),$$

where the hyper-plane is determined by the norm $\boldsymbol{\beta}$ and the location β_0, M is the width of its margin, and ϵ_i describes the relative distance. If $\epsilon_i = 0$, then the i^{th} training observation is on the correct side of the margin. If $\epsilon_i > 0$ then the i^{th} training observation violated the margin. If $\epsilon_i > 1$, then the i^{th} observation is on the wrong side of the hyper-plane, thereby it is misclassified by the classifier. In application, we set a positive number C as the tolerance limit, such that

$$C \geq \sum_{i=1}^{n} \epsilon_i$$

to control the relative number of observations accepted on the wrong side of margin. The Support Vector Classifier is the hyper-plane, which maximal its margin with a total tolerance of C, and it can be written as

$$C(\mathbf{x}; \hat{\boldsymbol{\beta}}, \hat{\beta}_0) = \text{sign}(\mathbf{x}^T\hat{\boldsymbol{\beta}} + \hat{\beta}_0), \text{such that}$$

$$(\hat{\boldsymbol{\beta}}, \hat{\beta}_0) = \max_{\boldsymbol{\beta}, \beta_0} \{M\}, \text{subject to}$$

$$\begin{cases} y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M(1 - \epsilon_i), \text{ for all } i = 1, 2, ..., n, \\ \\ \sum_{i=1}^{n} \epsilon_i \leq C, \ \epsilon_i \geq 0, \text{ for all } i = 1, 2, ..., n. \end{cases}$$

It can be find that the observations on the margin or on the wrong side of margin affect the classifier, these observations are called Support Vectors. Only the change of the Support Vectors will affect the Support Vector Classifier, whereas the classifier will not change if we just change the observations on the correct side of margin . When the total tolerance $C = 0$, no tolerance allowed, then the Support Vector Classifier degrade to Maximal Margin Classifier. When C is larger, the margin is wider because the restrictive condition is weaker, and the model will allow cumulate C observations on the wrong side of the margin. At the same time, the amount of Support Vectors, affecting the classifier, is big, which make the model more stable and have a potentially lower variance and higher bias. In contrary, if C is small, then the margin is narrow

and there are less amount of Support Vectors in the model. In this case, the classifier is more flexible, and have a relative high variance and low bias.

2.5.3 Support Vector Machine

In Support Vector Classifier, the boundaries between the two classes is hyper-plane determining by $\{\boldsymbol{\beta}, \beta_0\}$, and the classifier is

$$C(\mathbf{x}; \boldsymbol{\beta}, \beta_0) = \text{sign}(\mathbf{x}^T\boldsymbol{\beta} + \beta_0),$$

which is linear. However, it might be a poor setting when the boundary between the class are more likely to be non-linear sometimes. To improve the Support Vector Classifier, Support Vector Machine is proposed.

In Support Vector Machine, the boundary space is enlarged using high degree polynomial or some other kernels. For example, let's introduce the quadratic polynomial to the boundary space. Then, rather than fitting a classifier in p predictors space

$$X_1, X_2, ..., X_p,$$

we can fit a classifier in $2p$ predictors space

$$X_1, X_1^2, X_2, X_2^2, ..., X_p, X_p^2.$$

In this case, the original i^{th} observation in the training data, $(\mathbf{x}_i, y_i) = (x_{i1}, x_{i2}, ..., x_{ip}, y_i)$, will be modified to $(\mathbf{x}_i', y_i) = (x_{i1}, x_{i1}^2, x_{i2}, x_{i2}^2, ..., x_{ip}, x_{ip}^2, y_i)$, for all $i = 1, 2, ..., n$. The original parameters of the hyper-plane, $\{\boldsymbol{\beta}, \beta_0\} = \{\beta_1, \beta_2, ..., \beta_p, \beta_0\}$, will be modified to the parameters of p-dimensional quadratic surface

$$\{\boldsymbol{\beta}', \beta_0\} = \{\beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, ..., \beta_{p1}, \beta_{p2}, \beta_0\}.$$

Then, we can fit a Support Vector Classifier using the modified training data and parameter space $\{\boldsymbol{\beta}', \beta_0\}$ in a exactly same way with a normal Support Vector Classifier. The Support Vector Machine classifier is

$$C(\mathbf{x}'; \hat{\boldsymbol{\beta}}', \hat{\beta}_0) = \text{sign}(\mathbf{x}'^T \hat{\boldsymbol{\beta}}' + \hat{\beta}_0), \text{such that}$$

$$(\hat{\boldsymbol{\beta}}', \hat{\beta}_0) = \max_{\boldsymbol{\beta}', \beta_0} \{M\}, \text{subject to}$$

$$\begin{cases} y_i(\mathbf{x}_i'^T \boldsymbol{\beta}' + \beta_0) \geq M(1 - \epsilon_i), \text{ for all } i = 1, 2, ..., n, \\ \\ \sum_{i=1}^n \epsilon_i \leq C, \ \epsilon_i \geq 0, \text{ for all } i = 1, 2, ..., n. \end{cases}$$

We can see that the the boundary of the Support Vector Machine classifier is a quadratic surface determined by $\left\{\hat{\boldsymbol{\beta}}', \hat{\beta}_0\right\}$ in this case. Similar, we can extend the Support Vector Machine classifier by various way.

From the former part, we can find that the boundary of the Support Vector Machine classifier is determined by some kind of kernel function

$$K(\mathbf{x}, \boldsymbol{\beta}) = 0.$$

In Support Vector Classifier, the space of the classifier is given kernel function

$$K(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta}$$

Denote the inner product of two vector, $\mathbf{z}_i = (z_{i1}, z_{i2}, ..., z_{ip})$, $\mathbf{z}_j = (z_{j1}, z_{j2}, ..., z_{jp}) \in \mathbb{R}^p$, is

$$\langle \mathbf{z}_i, \mathbf{z}_j \rangle = \sum_{k=1}^{p} z_{ik} \times z_{jk}$$

Then, the kernel of Support Vector Classifier is $\langle \mathbf{x}, \boldsymbol{\beta} \rangle$. There are 3 popular kernel in Support Vector Machine [1], which are:

$$\text{d - Degree polynomial} : K(\mathbf{x}, \boldsymbol{\beta}) = (1 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)^d,$$

$$\text{Radial base} : K(\mathbf{x}, \boldsymbol{\beta}) = \exp\left(-\gamma |\mathbf{x} - \boldsymbol{\beta}|^2\right),$$

$$\text{Neural network} : K(\mathbf{x}, \boldsymbol{\beta}) = \tanh(\rho_1 \langle \mathbf{x}, \boldsymbol{\beta} \rangle + \rho_2).$$

In practice, we treat the tolerance C, the degree of kernels as the tuning parameters of the Support Vector Machine, and use cross-validation to choose best of them.

In this chapter, we discussed the basic ideas and features of some statistical learning algorithms. We also describe the conceptions of model complexity in some these algorithm, which will be further discussed in the next chapter. In the following part of this thesis, we will focus on the model selection methods and how to find out the one with potentially best prediction ability using observed data. At last, we will investigate how the data size affect the prediction performance of these algorithms and check whether do we always need a large data set to construct a good model.

Chapter III

Model Selection & Learning Curves

3.1 Model Selection

In previous chapter, we introduced some widely used statistical learning algorithms. Many of these algorithms have are involved tuning parameters to control the model complexity. More complex models can potentially fit the data better, but have a potential risk of over-fitting the data. If the models are too simple, they cannot fully capture patterns in data, called under-fitting the data. Validation approaches are commonly used to estimate predictive performance and to select the best tuning parameters. In this chapter, we will discuss the bias and variance trade-off, and validation approaches for model selection. In section 3.3, golden section search optimization is introduced. In the third part of this chapter, we conclude with learning curves, which are used to describe the relationship between the model performance as a function of training data size.

3.1.1 Bias, Variance and Model Complexity

In linear regression, the coefficients are estimated by minimizing the mean squared error

$$\text{MSE} = \sum_{i=1}^{n} \Big(y_i - \hat{f}(x_i) \Big)^2,$$

where y_i and $\hat{f}(x_i)$ are the observed and predicted response of i^{th} observation respectively. The MSE is called the training error, because it is computed using training data. However, we are not really interested in training error, because it is always possible to train a very complex model with very small training error. Hence, training error is not a testing measure of a model's predictive performance and an independent testing set of observations is required.

Unfortunately, an independent testing data set is not available in most cases.

Consider a statistical learning problem with response vector Y, predictor matrix X and estimated model $\hat{f}(X)$ using the training data τ. which is a sample from the population T. The test error is defined as

$$\mathrm{Err}_T = \mathrm{E}\left[L\left(Y, \hat{f}(X)\right)|\tau\right],$$

where $L(Y, \hat{f}(X)$ is a pre-defined loss function, which typically is mean squared error for regression problems or mis-classification rate for classification problems. The test error is expected average over the randomness brought by the prediction model $\hat{f}(X)$ and training data τ randomly sampled from the population.

In linear regression, we assume that the $Y = \hat{f}(X) + \epsilon$, where $\hat{f}(X)$ is linear function of predictors X, ϵ is an error term with mean 0 and variance σ_ϵ^2 and the error is measure by squared error. Then, the expected error of an unseen observation, $X = \mathbf{x}_0$, is

$$\begin{aligned}
\mathrm{Err}_T(\mathbf{x}_0) &= \mathrm{E}\left((Y - \hat{f}(\mathbf{x}_0))^2|X = \mathbf{x}_0\right) \\
&= [\mathrm{E}(\hat{f}(\mathbf{x}_0)) - f(\mathbf{x}_0)]^2 - \mathrm{E}[\hat{f}(\mathbf{x}_0) - \mathrm{E}(\hat{f}(\mathbf{x}_0))]^2 + \sigma_\epsilon^2 \\
&= \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}.
\end{aligned}$$

The first term is the squared bias of the model, which is the average difference between the prediction and the true mean. The second term is the variance of the model $\hat{f}(\mathbf{x}_0)$. The third term is the irreducible error, such as the measurement error, unless $\sigma_\epsilon^2 = 0$, which means every observation is exactly on a line.

If we fit a polynomial regression model, where no interaction terms considered, the sum of squared residual of polynomial model is smaller than it of linear model, which is shown in Appendix I. It suggests the more complex models fit the training data better than simple models, but more parameters or complexity introduced, so they will potentially has higher variance and low bias.

In many statistical learning algorithms, the model complexity is controlled by tuning parameters. Tuning the parameters of statistical method is a process of pursuing a trade-off between the model bias and variance.

3.2 Resampling Methods

The basic idea of resampling is to draw samples from the data repeatedly, and fit the model to each sample and evaluate the model fitting and obtain much stable models for some algorithms. This can be a computationally expensive process in the past, because the model need to be fit a model more times, but now, with the improvement of computation ability of computers, resampling methods are very common.

In application, we are always interested in assessing the goodness of a model fitting, which can be measure by error rate(for classification problems) or mean squared error(for regression problems). A common idea is to calculate the error of the model fitting by the data used to training the model. In statistical learning, this error is called training error. However, the training

3.2.1 Validation Approach

In validation approach, we randomly separate the original dataset into two parts, the training dataset and testing dataset. The training dataset is used to build the model, and the testing dataset or validation set is used to evaluate the model's predictive performance. There are two main drawbacks for validation approach. Firstly, it is likely to overestimate the prediction error of the model, because we just used a proportion of the whole available data to train the model, which is likely to be worse than the one using the whole data. The second, this approach is vulnerable to the subset chosen to train the model. Because the model might change dramatically if different observations are chosen to be training data.

3.2.2 Leave-One-Out and K-fold Cross-Validation

To overcome the problems of validation approach, leave-one-out cross-validation (LOOCV) uses a similar method. In LOOCV, one suggestion is used to test the model, and the others are used to construct the model. This is repeated for each observation and the average performance

is calculated. The advantage of LOOCV is that it has low bias, because the models are fit using $n-1$ observations every time, nearly the entire training data set. However, LOOCV is potentially expensive when n is large, because the model is fitted n times. In addition, the n training sets that are used are very similar. As a result, the models are highly correlated which leads to high variance.

In K-fold cross-validation, we randomly separate the training data into K groups, called folds, with approximately equal size. Then, keep one fold as the validation set and use the other folds training data to fit the model. Next, repeat this approach K times, using the fold once as a validation set. The K-fold CV error is average of them. K-fold CV is same with LOOCV when $K = n$, so LOOCV is a special case of K-fold CV.

The choice of K used in K-fold CV is also a trade-off between bias and variance. When K is big, the size of data used to fit model is close to the full training data size, and each training set is more similar. This means the models are similar, so the bias is small but the variance is quite big. When K is small, the bias is high but the variance is low. Typically, 10-fold cross validation is used for model selection application [1].

3.3 Golden Section Search

In last section, we discussed some resampling methods which can be applied in model selection and to find tuning parameters in algorithms. However, using resampling to evaluate every possible value of a tuning parameter can be computationally expensive. In this study, Golden Section Search is used when there is only one tuning parameter involved.

Golden Section Search is an optimization technique for finding an extreme point of a strictly unimodal function. It is a efficient way to find location of the extreme value through reducing the considered interval progressively. When there is only one tuning parameter in the statistical learning model, we can apply the Golden Section Search to find the best tuning parameter which gives the minimum K-fold cross validation error.

Suppose the cross validation error of tuning parameter α is CV(α), where $\alpha \in [a, b]$. We assume CV(α) is an unimodal function and apply Golden Section Search to find the best α which gives the smallest cross validation error. Denote $x_1, x_2 \in [a, b]$, such that $x_1 < x_2$ and

$$\frac{||b - x_1||}{||x_1 - a||} = \frac{||x_1 - a||}{||x_2 - x_1||}.$$

Then, we have

$$\mathrm{CV}(x_1), \mathrm{CV}(x_2) < \min\left(\mathrm{CV}(a), \mathrm{CV}(b)\right).$$

If $\mathrm{CV}(x_2) - \mathrm{CV}(x_1) > 0$, then the minimum can not locate in (x_2, b) because $\mathrm{CV}(\alpha)$ is a strict unimodal function. So the minimum must be located in interval (a, x_2). If $\mathrm{CV}(x_2) - \mathrm{CV}(x_1) < 0$, similarly, the minimum can not locate in (a, x_1), So the minimum must be located in interval (x_2, b). The Golden Section Search choose the interval in a way such that the proportional length divided by the points $\{a, x_1, x_2\}$ or $\{x_1, x_2, b\}$ are same. So we also need to let

$$\frac{||b - x_2||}{||x_2 - x_1||} = \frac{||b - x_1||}{||x_1 - a||}.$$

Then, it is easy to show

$$\theta = \frac{||b - x_1||}{||x_1 - a||} = \frac{1 + \sqrt{5}}{2}.$$

If $\mathrm{CV}(x_2) - \mathrm{CV}(x_1) > 0$, we consider $\mathrm{CV}(x_3)$, where $x_3 \in (a, x_1)$, such that

$$\frac{||x_2 - x_1||}{||x_1 - a||} = \frac{||x_1 - a||}{||x_3 - x_1||},$$

and compare $\mathrm{CV}(x_2)$ and $\mathrm{CV}(x_3)$ to determine the next interval. The reason of letting $x_3 \in (a, x_1)$ is the length of interval (a, x_1) is bigger than it of (x_1, x_2), therefore, it is more efficient to consider the value in the bigger interval. We continue this process till some stopping criteria applied. For example, the stopping criteria can be the length of interval to be considered. if its length is smaller than some threshold, we stop the algorithm.

Mathematically, the above setting ensure the interval length shrink by same proportion in each step, and the proportion of the location of point $\theta = \frac{1+\sqrt{5}}{2}$ is called the Golden Ratio. In this

thesis, we used golden section search to find the optimal number of neighbours in KNN. This give similar results comparing to the normal approach, but smaller the number of neighbours need to be considered, hence safe more computational resources.

3.4 Learning Curves

A learning curve is used to describe the relationship between the knowledge gained and the time cost. It was introduced by Hermann Ebbinghaus in 1885 in psychology. In statistical learning, learning curves are used to represent the relationship between the performance of statistical learning algorithms and the amount of the data used in training the model.

A learning curve satisfies three conditions. I) It is a positive and increasing function of the number of observation. II) It is a concave down function. III) The learning curve is bounded. For classification problems, it is straightforward to state that the accuracy can not beyond 100%.

3.4.1 Learning Curves Fitting

In this section, we consider fitting a learning curve. Denote the accuracy of a statistical learning model is *Acc* and the number of training observations n. Frey and Fisher (1999) use the power law function $\text{Acc} = a \times n^{-b}$ to fit the learning curve of C4.5 decision tree, where $a > 0$ and $b > 0$ are the parameters to estimate. Their work showed that power law is a potentially suitable family to fit learning curves. John and Langley used $\text{Acc} = a - b \times n^{-c}$ to fit learning curves for Naive Bayesian classifiers. Gu and Hu [38] compared the performance of several families, shown in Table 3.1, and found that the 3-parameter power law family, $\text{Acc} = a - b \times n^{c}$, performed better. Some studies showed the learning curves are not well behaved, because there might be a sudden increase in the learning curves, especially in the small samples part [39]. However, there are many studies showed the learning curves behaved well in large data sets [40]. The reason might be that the variation of the learning curves is relative large on small sample sizes. Because the variance of model constructed using small sample size can be big, which make the estimated learning curves are less reliable in former part.

To improve learning curves fitting methods, many studies dedicate to find good sample sizes schedules to fit learning curves more efficiently. For example, John and Langley [42] proposed a progressive sampling schedule $n_k = n_0 + k \times n_\alpha$, where the performances of the models us-

Model Name	Formula	Remarks
2-Parameters Power Law	$\text{Acc} = a \times n^b$	$a > 0, b > 0$
Logarithm	$\text{Acc} = a + b \times \log(n)$	$a > 0, b > 0, c > 0$
3-Parameters Power Law	$\text{Acc} = a - b \times n^{-c}$	$a > 0, b > 0, c > 0$
Vaper Pressure	$\text{Acc} = \exp\{(a + b/n + c \times \log n\}$	$a > 0, b > 0, c > 0$
MMF Model	$\text{Acc} = (a * b + c \times n^d)/(b + n^d)$	$a > 0, b > 0, c > 0$
Weibull	$\text{Acc} = a - b \times \exp\{-c \times n^d\}$	$a > 0, b > 0, c > 0$

Table 3.1: Candidate Families of the Learning Curves

ing a linear scheduled sample sizes were considered when fitting a learning curve, such as a sequence of sample sizes $\{100, 200, 300, 400, ..., N\}$. The drawback of this schedule is obvious. When the full training data size is large, a huge number of sizes need to be considered which is computational expensive. Alternatively, Foster and David showed geometric sampling schedules, $n_k = n_0 \times n_\alpha^k$, is more robust [40]. So, in this thesis, we considered the performance of algorithms in geometric sampling schedules, where the models used a geometric sequence of sample sizes, such as a sequence of sample size $\{100, 200, 400, 800, 1600, ..., N\}$.

Chapter IV

Simulation Study

4.1 Simulation Study

In this study, our goal was to investigate how the number of training observations affects performance of different statistical learning algorithms (discussed in Chapter 2), and whether this relationship can be described using a 3-parameter Power Law learning curve describe in section 3.4. This study is done by three shared computing environments. The first had 2 x Intel® Xeon® CPU E5-2667 @ 2.90 GHz (boost up to 3.50 GHz) – total of 12 cores, 24 threads of execution with 192GB of RAM. The second had 1 x Intel® Xeon® CPU X5670 @ 2.93GHz (boost up to 3.33 GHz) – total of 6 cores, 12 threads of execution with 48 GB of RAM. The third had 2 x Intel® Xeon® CPU E5-2690 v3 @ 2.60GHz (boost up to 3.50 GHz) – total of 24 cores, 48 threads of execution with 128GB of RAM. All the simulations and analysis were performed using **R** [52].

Parallel computing in this simulation was performed using the packages **doParallel** [53] and **foreach** [54]. At each run, a sample was selected from training dataset, a classifier built, accuracy measured and the processing time record. We call this a "job". The results of jobs done by one "worker" were recorded in a ".csv" file, and multiple "workers" output their results to several ".csv" files in parallel. A number of "jobs" were assigned to "worker" in advance, and the "worker" output the results when every new job finished during the running. There are a few advantages of this strategy. Firstly, all the results will be recorded automatically once they are finished, so an interrupt, for example a sudden power outage, will not destroy the results. This is important because the some of the "jobs" may takes days or weeks. Secondly, the progress of the simulation can be monitored by the output file. Because the running time of each job is recorded, the remaining running time can be roughly predicted. Thirdly, the maximum number of 'workers' on different sized jobs can be estimated within the limitation of memory. An example of **R** code to display parallel computing in this study can be found

in Appendix III. More details and applications of parallel computing packages **doParallel** and **foreach** can be found in [53] [54].

4.2 Simulation Setup

In this simulation study, we considered classification problems. The models' performances are measured using the classification rates on independent test data, which were not used in the training process. We will consider sample sizes that follow a geometric schedule number of observations, $0.01\%, 0.02\%, 0.04\%, 0.08\%, \ldots$ of the total, until to the full training data. For each sample size, simple random sample (without replacement) was used to choose the observations from the full training dataset. We selected 50 samples for each sample size and reported the averaged performances. We considerd 8 algorithms, which are K-Nearest Neighbours (KNN), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Support Vector Machine (SVM), Classification Tree (CT), Bagging, Random Forest (RF) and Boosting Trees (BTs).

For LDA and QDA, no tuning parameters needs to be considered. We used the functions **lda** and **qda** in the package **MASS** [55] to train and evaluate these models.

For KNN, we used the function **kknn** in the package **kknn** [56]. Euclidean distance was used in the model, and all the predictors were normalized to mean 0, deviance 1 to make them have the same influences. The tuning parameter, K, was chosen using golden section search approach described in section 3.3.

For CT, we used the package **rpart** [57]. The minimum number of observations in every terminal node is set to 50. The tuning parameter, tree's depth, was chosen by 10-fold cross validation. Details of it can be found in section 2.3.2.

In Bagging, we used the function **randomForest** in the package **randomForest** [58]. Each tree in Bagging should be fully built. However, in this package, the maximum depth of each tree is 32. So, the maximum depth of 32 was used, and we set the number of trees to 500. The model was evaluated using the out of bag error, which determined whether we needed to construct more trees. We stopped adding additional trees to the forest, if the variance of out-

bag-error over the past 50 trees was smaller than 0.0001. Because it suggests the performance the random forest tends to be stable, adding more tree can not improve the model further.

In RF, we used the function **randomForest** in the package **randomForest** [58]. The maximum depth of each tree was set to 32. The number of trees will be set to 500, and then the model was evaluated by the out of bag error. We stopped adding trees, if the variance of out-bag-error over the past 50 trees was smaller than 0.0001. Because the random forest is not very sensitive to the number of variables considered at each split ($Mtry$), it was set to $\sqrt{p}$, where p is the number of predictors. In random forest, more memory was required to store large numbers of trees. However, because these trees are built independently, so it is possible to break down the process. For example, in this simulation, we applied an updating strategy for large datasets. We built one tree and record its predictions for the testing data, and then deleted it to clear the memory. Repeating this process 500 times to get a prediction was more memory usage friendly than training all tree in once.

In BTs, we use the function **gbm** in the package **gbm** [59]. We considered depth 1, 2, 4 and 6. The shrinkage parameter were set to 0.001. The number of trees tried was set to 5000 at first and the optimal number of trees was chosen using 10-fold cross validation [33]. We stopped adding trees when the variance of the cross validation error made by last 50 trees was smaller than 0.0001.

The SVM approach used the function **tune** in the package **e1071** [60]. The kernel considered were linear, polynomial and radial basis. The degrees considered were 1, 2 and 3. All these tuning parameters were chosen using 10-fold cross validation.

The learning curves were fitted using function **nls** in the package **stats** [52]. In our study, we only used the 3-Parameter Power Law function, $\text{Acc} = a - b \times \text{n}^{-c}$, to fit the learning curves. The models were determined by non-linear least square estimates of the parameters a, b and c. In the fitting process, the function **SSasymp** was used to display a self-starting non-linear square asymptotic regression model, where the "start points" for each parameters and the maximum iterations were set manually to ensure converge. More details of this function can be found in [52].

4.3 Data Sets Details

In this study, we considered 3 datasets from the UCI repository [4]. The first dataset is from high-energy physics, **HepMass** [4]. The goal is to classify the signatures of exotic particles, where 1 is for "signal" and 0 for "background". The predictors are consist of low-level kinematic features, high-level kinematic features and particle mass. All the predictors are numerical. There are 7,000,000 observations in training set and another 3,500,000 observations in the testing set. In simulation experiment, a number of training observations were sampled from the training set to learn the classifier, and its performance was measured using the whole independent testing set.

The second dataset is called **YearPredictionMSD** [4]. The problem is to predict the release year of songs using audio features. The predictors are extracted from the timbre features from The Echo Nest API [41]. All the predictors are numerical. We divided the release year into two groups. The first group were the songs released after 2000, and the second group were the songs released before 2000. The first 463,715 examples in the data set were used for training model, and rest 51,630 examples for testing the model performance, which is recommended by the publisher.

The third dataset is called **PokerHand** [4]. The prediction variable is the score of a poker hand. We divided the scores into two levels, "Poker Hand" for the ones with scores bigger than 0, or "None" for scored equal to zero. Then, the problem becomes whether the statistical learning algorithm can identity a poker hand. The proportion of each class is approximately same. Because we wish to investigate how the number of training observation affect the model performance, we used the original testing data (with 1,000,000 observations) for training data, and the original training data (with 25010 observations) for testing data.

4.4 Simulation Results

4.4.1 Sampling Size vs. Model Accuracy

In the following graphs, the Y axis represents the classification rate and the X axis represents the number of training observations used in a log scale. The mean and variability of the estimated classification rate on each sample size were displayed using a sequence of box plots.

Figure 4.1 - Figure 4.4 show the simulation results of data set, **HepMass**. In Figure 4.1, we compared LDA and QDA. The average performance of LDA is better than QDA on every sample size. Hence, the linear classifier is better than the quadratic classifier in this case. In addition, the variance of LDA is also smaller than QDA. The reason might be that there are more parameters in QDA leads to the bigger variance of the QDA models (as illustrated in section 4.1). When the sample size increased, the average classification performance increased and the variance decreased for both LDA and QDA. However, the improvement of classifier's accuracy got smaller for both algorithms when more training observations were given. After about 15,000 observations given, the performance of both the algorithms tend to be flat, and no gain for more observations given. This is because the LDA and QDA classifiers are constructed using estimated means and variances (or covariances). As a result, in this case, full data might not be necessary to construct a model with acceptable accuracy using LDA or QDA approaches.

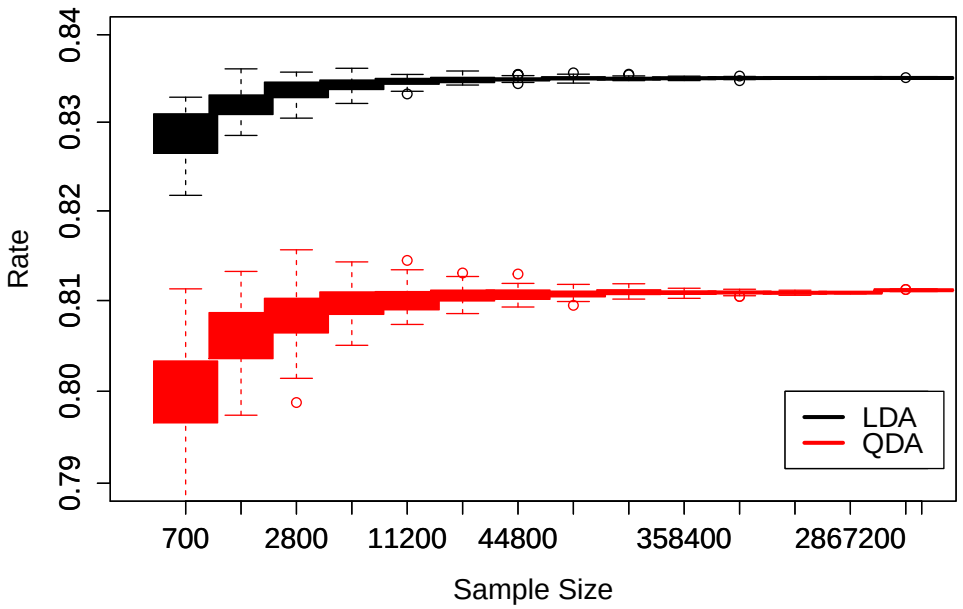

Figure 4.1: The performance of LDA and QDA in dataset **HepMass**

Figure 4.2 illustrates the performances of CT, Bagging and RF. The Random Forests had better predictions than a single classification tree or Bagging Trees, given same number of training

observations. Furthermore, the variances of prediction performance of Random Forests are also smaller than a single classification tree or Bagging on all sample sizes. As illustrated in section 2.4.2, Random Forests make improvements by bagging and decorrelating the trees, which make their performed better and variance lower. With the number of training observations grown, the averaged accuracy of all these 3 algorithm increased, but the gains get smaller. After given 11200 observations given, adding more observations only get slight gains. For example, the difference between the averaged accuracy of random forest using 20,000 and 2,000,000 is just around 1%, which is extremely small. It suggests that using full data may not make too much difference in this case.

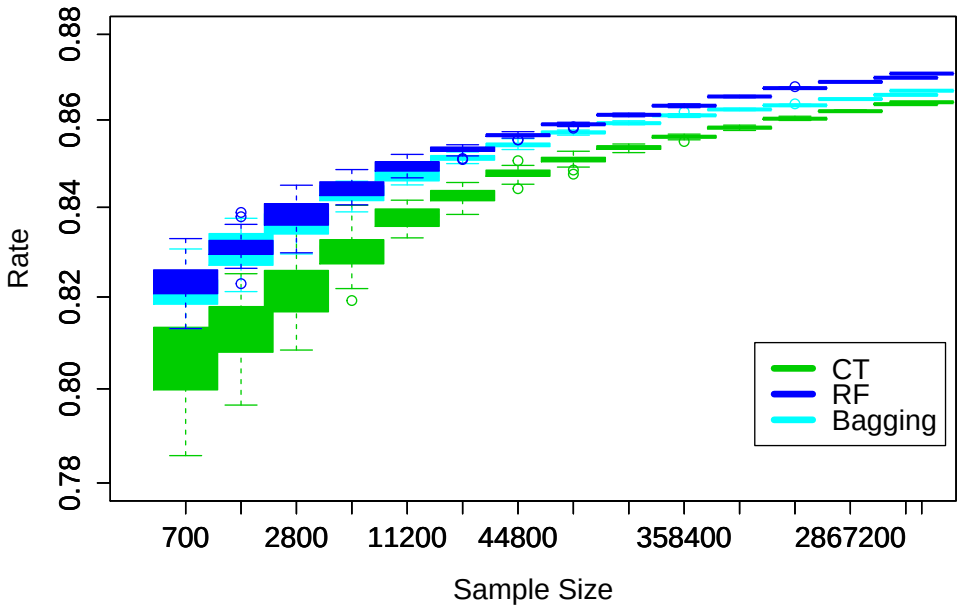

Figure 4.2: The performance of CT, Bagging and RF in dataset **HepMass**

Figure 4.3 shows the performance of Boosting Trees. When the number of training observations increased, the variance of model performance became smaller. We also can see the algorithm reached its limit given about 200,000 training observations. What is interesting is that the classification rate decreased when more than 358,400 training observations used. The reason may be that the parameters were not chosen correctly. In Boosting Trees, lower learning rate and more trees are necessary to capture the patterns of data when a large amount observations

are used in training [33]. When the number of observations increase, the pattern of data tends to be more complex. Because all trees in boosting are "slow learners"(only with depth of 1,2 or 4), far more trees are needed to capture these patterns even using the same learning rate. Otherwise, the performance of the algorithm might be even worse given more data. However, we did not change learning rate and number of trees for large sample size, because it is too computationally expensive.

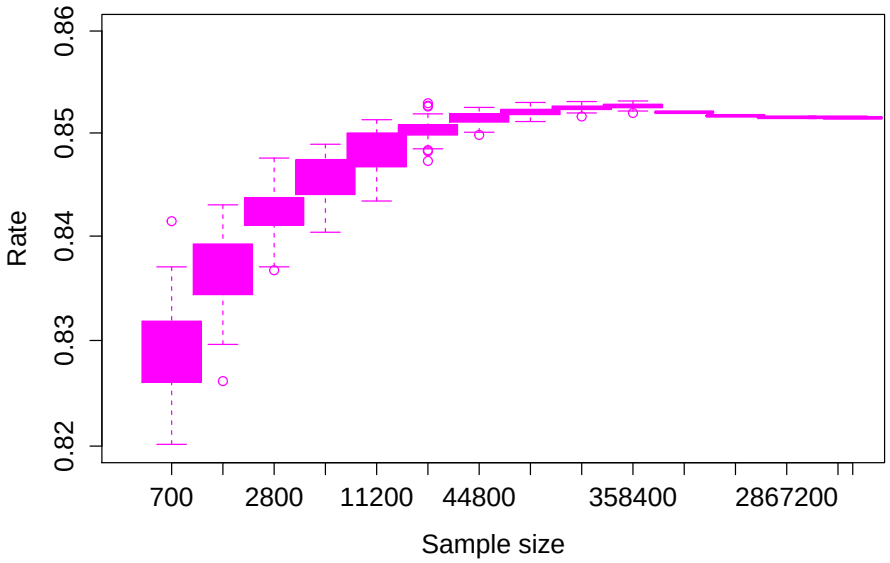

Figure 4.3: The performance of Boosting Trees in dataset **HepMass**

Figure 4.4 shows the estimated mean of classification rates of the algorithms on various sample size. The coloured lines are learning curves fitted by the 3-Parameter Power family functions. The learning curves fit the data well. The performance of CT, Bagging and RF increased faster than other algorithms, given more observations. It is because these tree based methods can make better use of the additional information to construct classifiers with more complex boundaries. These classifiers can capture the patterns of data better. The Support Vector Machine with 3 degrees radial kernel also performed well. Except Boosting trees, the prediction performance of all other algorithms increased with more training data given, but only slightly

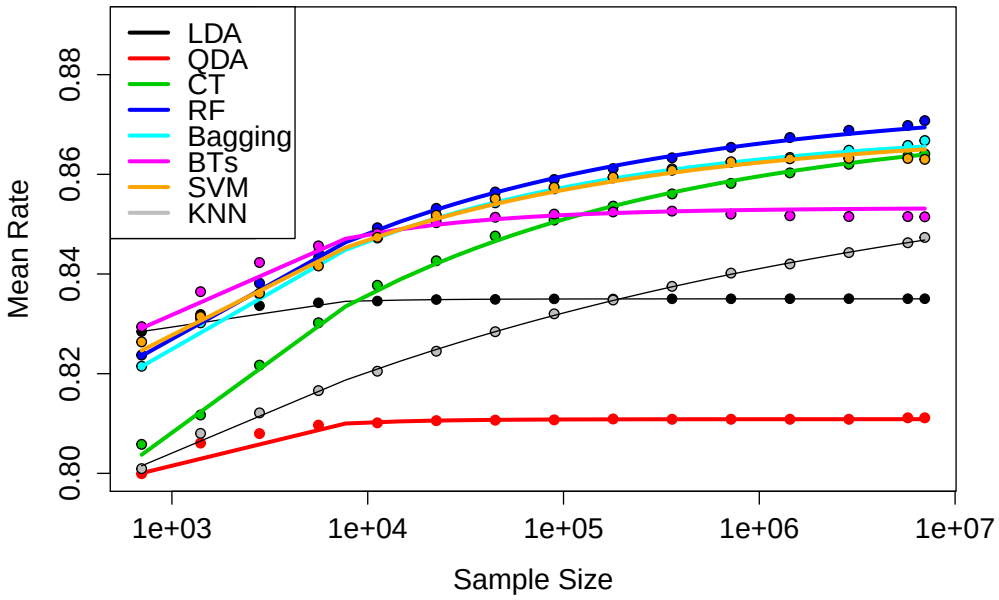

Figure 4.4: The averaged performances of the algorithms in dataset **HepMass**

when sample size is big. For instance, for SVM, the mean rate using 700 observations was about 82.7%, and it reached to nearly 84 % using 2800 observations, over 1 % gained. However, the difference of SVM using 10,000 and 1,000,000 observations is only about 1%.

Figure 4.5 - Figure 4.8 show the simulation results of data sets **YearPredictionMSD**, and we find what happened in this dataset is similar to dataset **HepMass**. In Figure 4.5, we can find that the classification rate increased for both algorithms when given more training observations, but reached their limit quickly. The variances of QDA classifiers are bigger than LDA, because there are fewer parameters to be estimated in LDA.

Figure 4.6 shows the performance of CT, Bagging and RF. The classification rate of all these 3 algorithms improved with given more training observations given, but the gains get smaller. For example, the difference between the RF average rate using 4,000 and 40,000 is just around 2%. As an improvement of Bagging Trees, RF is better than Bagging and CT in terms of rate and variance.

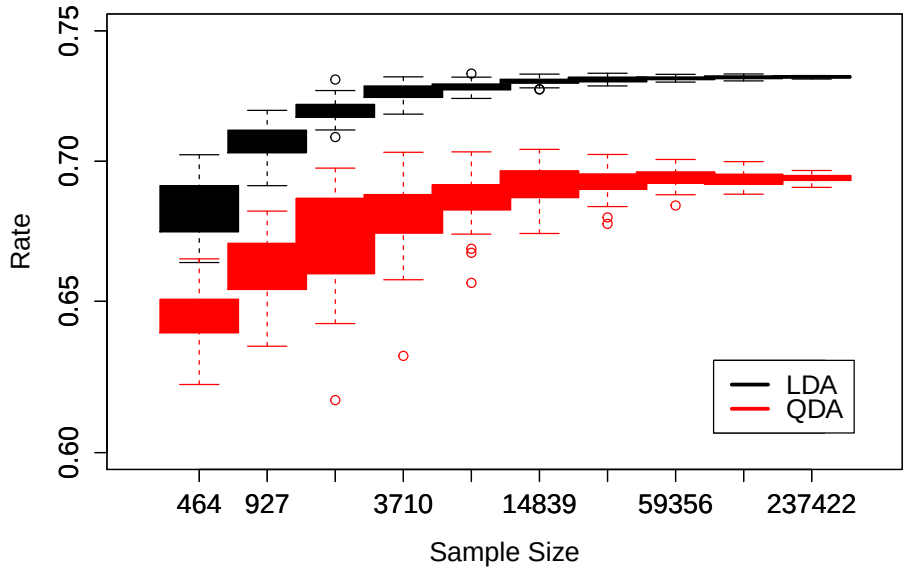

Figure 4.5: The performance of LDA and QDA in dataset **YearPredictionMSD**

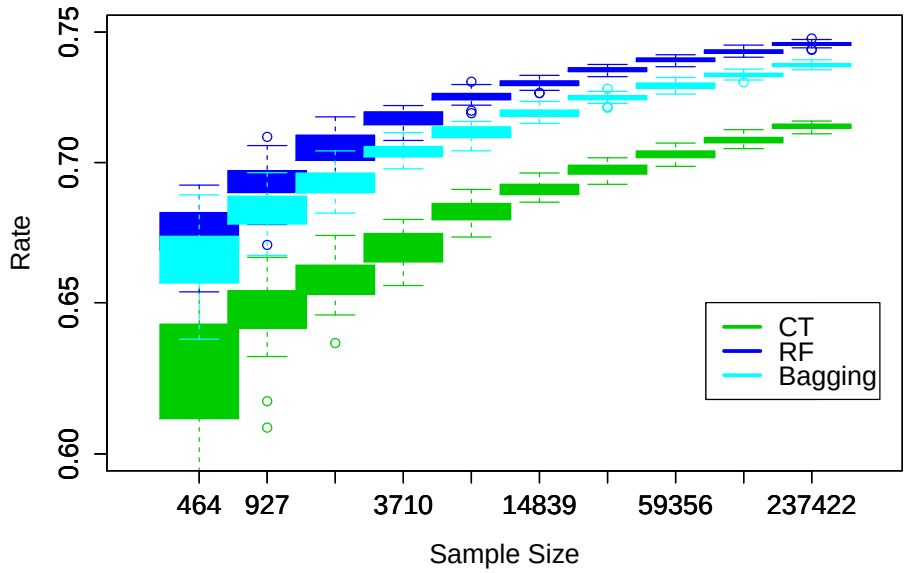

Figure 4.6: The performance of CT, Bagging and RF in dataset **YearPredictionMSD**

Figure 4.7 shows the performance of Boosting Tree, which it similar to what happened in dataset **HepMass**. We can see the algorithm reach its limit given about 200000 training observations. When more observations given, the rate of Boosting Tree decreased. As illustrated in dataset **HepMass**, the parameters were not chosen correctly for large size, but we did not adjusted it because of high computational cost.

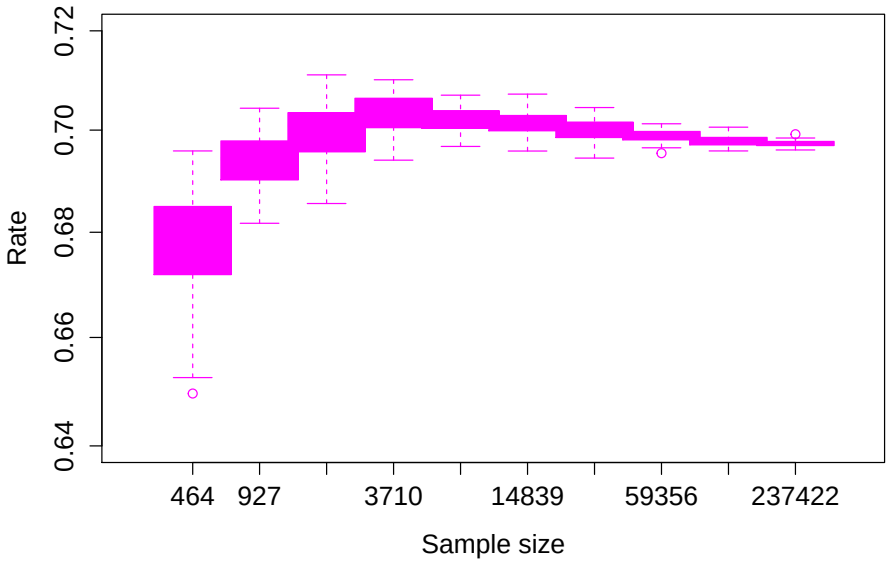

Figure 4.7: The performance of Boosting Trees in dataset **YearPredictionMSD**

Figure 4.8 shows the estimated mean of classification rates of the algorithms. The coloured lines are learning curves fitted by the 3-Parameter Power family functions. The estimated learning curve fit the data well, especially when the number of training observation is big. It suggests the learning curve might be more reliable with large amount sampling size. Except Boosting trees, the prediction performance of all other algorithms increased with more training data given, but only slightly, especially when the sample size was large. The mean rate of all algorithms were nearly flat over 5,000 training observations given. So, using the full data might not be necessary.

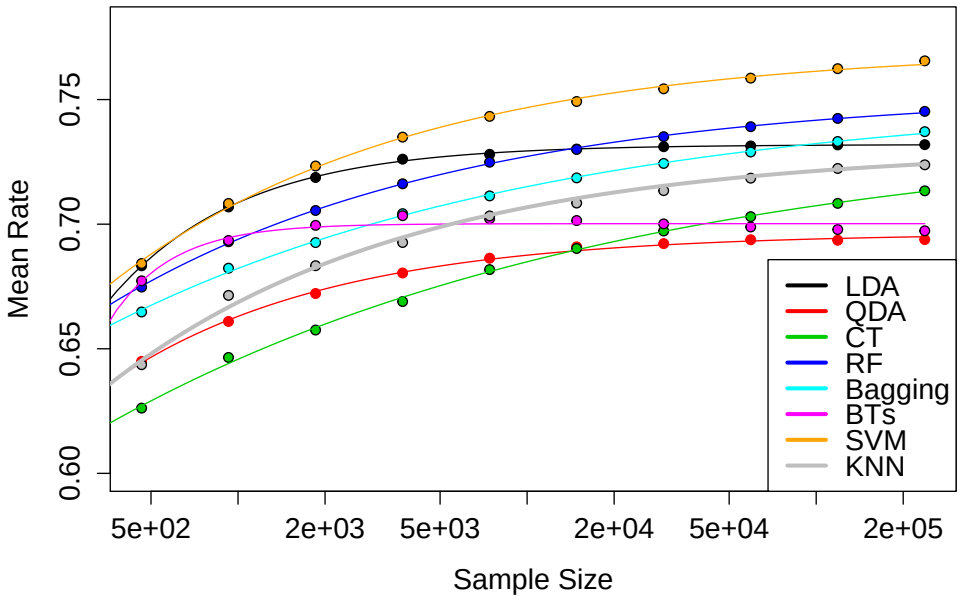

Figure 4.8: The averaged performances of the algorithms in dataset **YearPredictionMSD**

Figure 4.9 - Figure 4.12 show the simulation results of data sets, **PokerHand**. From Figure 4.9, we can the both LDA and QDA did not works well, because the goal in this dataset is to recognize the "hands" of the data, poker hands or non poker hands, but it is hard for LDA or QDA algorithms. For instance, in LDA, the classifiers were built using the estimated pooled variances and means of the classes, and they can not distinguish the observations which has a pair of same ranks, so the "single pair" hands will never be recognized by the algorithms, no matter how many observations given. As a result, we did not find any gains in LDA using more observations. The rates of QDA increased with more observations used, and reached its limit around 25,600 training observations given. It suggests QDA recognized some "hands". For example, the "single pair" and "two pair" might be recognized by QDA, because all these "hands" has same ranks, which means the correlations between some of their ranks are equal to "1" , then it can be recognised by QDA. As a result, If we do not know what are the possible form of these pattern, parametric methods, such as LDA and QDA, might be bad choice dealing with this sort of problems.

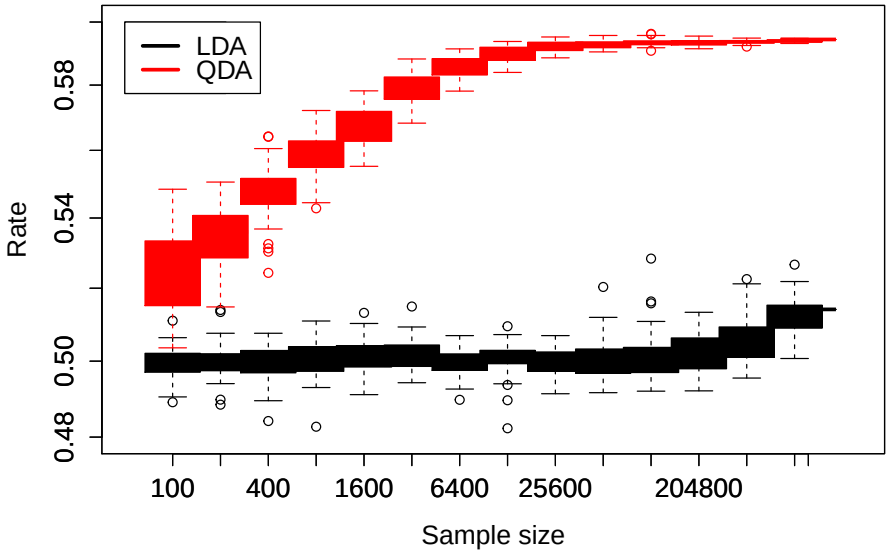

Figure 4.9: The performance of LDA and QDA in dataset **PokerHand**

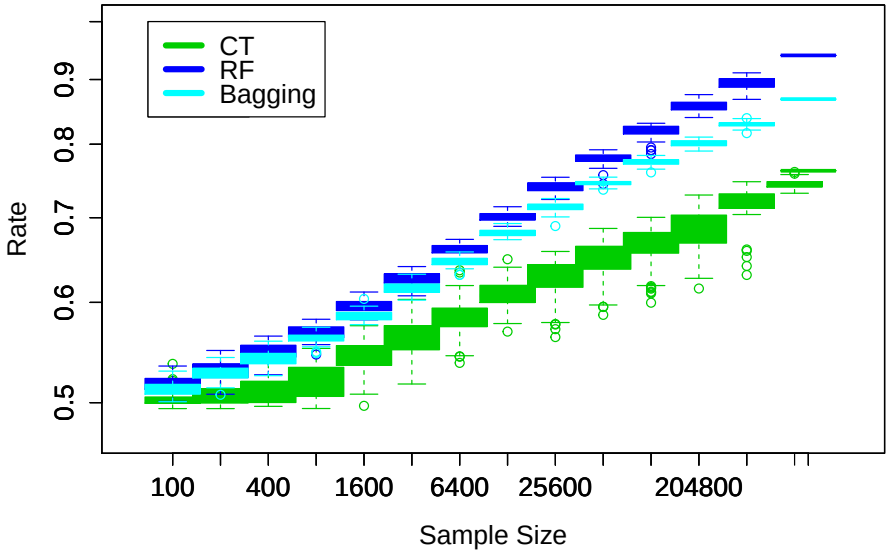

Figure 4.10: The performance of CT, Bagging and RF in dataset **PokerHand**

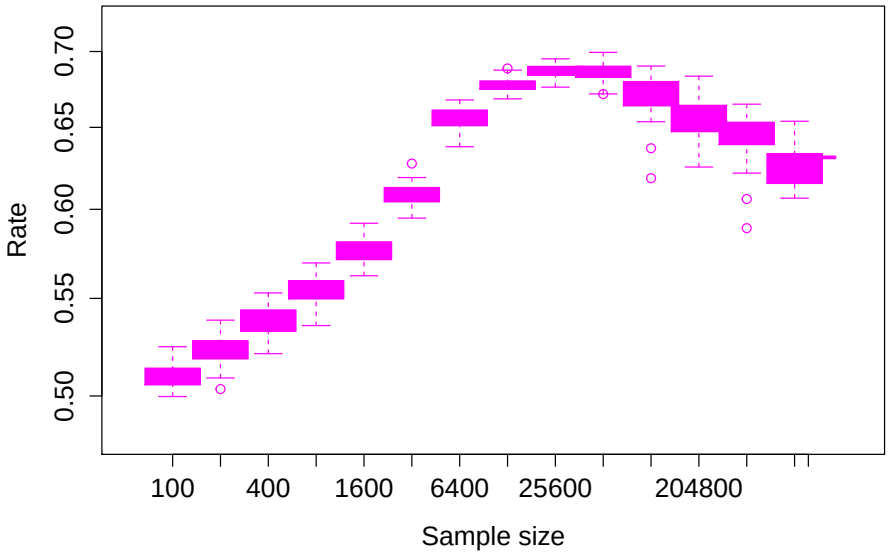

Figure 4.11: The performance of Boosting Trees in dataset **PokerHand**

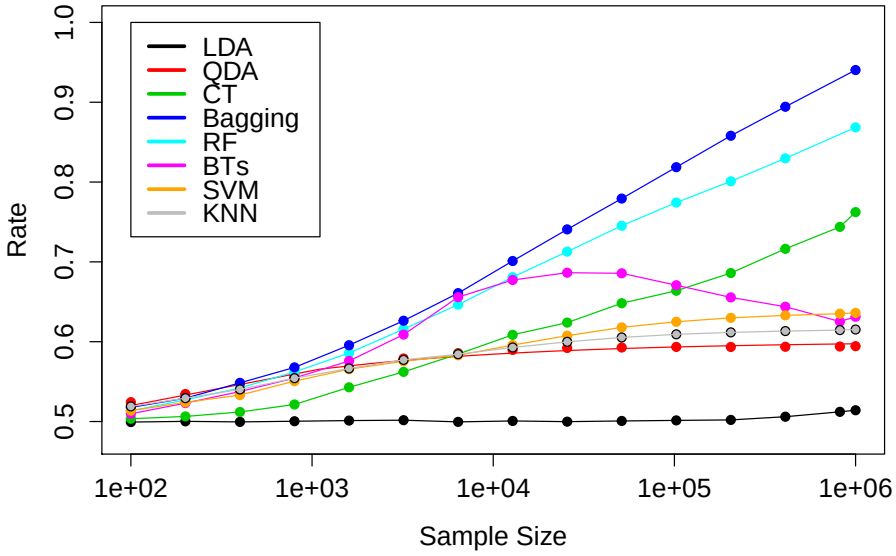

Figure 4.12: The averaged performances of the algorithms in dataset **PokerHand**

From Figure 4.10, RF, Bagging and CT works well, compared to LDA and QDA. The tree based methods can recognise the poker hands well. Increasing the number of training observations improved the performance of all these 3 algorithms. The classification rate of RF was better than Bagging and CT, and reached over 90% using the whole training set with smaller variance.

In Figure 4.11, the classifiers are the boosting trees. The performance increased at first, and decrease after given 25000 observations. The reason is same with previous results, the parameter setting need to take the data size into consideration, otherwise, using more observations could give even worse results. However, we did not adjusted because of the limited computational resources.

4.4.2 Sample Size vs. Computational Usage

In this simulation, we also consider the computational usage of different algorithms using different sample sizes. The results are shown in Table 4.1, Table 4.2 and Table 4.3. Please note that different machines have different properties, computational abilities and environment. In addition, the structures of different data sets will also make a difference in processing time and memory occupation. So, these results only can be used to measure the relative computational usage for different algorithms using different training data sizes.

We can find the LDA and QDA are extremely efficient to implement, because the classifiers are only determined by estimated means, variances (pooled variance for LDA) and proportion of each class. Single tree (CT) also can be built quickly, and the Bagging and RF took longer. KNN and SVM are time consuming, and both of the algorithms are very sensitive to the sample size, especially SVM. The reason is the tuning parameters need to be tuned in both approaches which is very time consuming. However, if we have bounds potential best ranges for the tuning parameters, the computational time can dramatically decrease. Table 4.1, Table 4.2 and Table 4.3 showed that Increasing number of training observations prolong the computational time dramatically for all algorithms (except LDA and QDA), but only small gains in model performances. For example, the Table 4.1 shows the computational time of different algorithms using different sample size in dataset **HepMass**, and we can see that the computational time for SVM using 0.1 % (or 7,000) observations is around 40 mins and over 2 weeks for 10 % (or 700,000) observations, but the prediction accuracy gain is only about 1%.

Another main barrier is the limited RAM of the computer(s). In the simulation study, we find the memory usage of some algorithms are very high, such as RF, Bagging and KNN, but only a small amount of memory required by CT, SVM and Boosting trees. This affect the number of cores we can use in parallel. For instance, for dataset **HepMass**, using about 70,000 observation to train a SVM classifier just used around 1GB of RAM, but over 10 GB of RAM for RF and 8 GB for KNN. In addition, the sensitivities of memory usage for some algorithms are various. For dataset **HepMass**, fit a SVM classifier with 700,000 training observations just used less than 5GB of RAM, but over 40GB for RF. However, because the trees are built independently using bootstrapped samples, we can break them down to save memory. For example, when the sample size was very large, we built one tree and record its predictions for the testing data, and then deleted it to clear the memory. Repeating this process 500 times to get the same results with building all 500 trees in once. But not all algorithms can be break easily, such as KNN.

Models	Sampling Fractions				
	0.01%	0.1%	1%	10%	100%
LDA	21 sec	22 sec	25 sec	54 sec	6 mins
QDA	31 sec	32 sec	36sec	49 sec	5 mins
KNN	8 mins	30 mins	4 hours	$>$ 2 days	$>$ 2 weeks
CT	8 sec	10 sec	40 sec	12 mins	3 hours
BTs	10 mins	20 mins	1 hour	15 hours	3 days
Bagging	2 mins	4 mins	18 mins	3 hours	$>$1 day
RF	2 mins	3 mins	10 mins	2hours	1 day
SVM	2mins	40 mins	$>$1 day	$>$ 2 weeks	$\gg$ 2 weeks

Table 4.1: The averaged computational time of every job (including model construction and testing using the specified size of data for once) for data set **HepMass**.

Models	Sampling Fractions			
	0.1%	1%	10%	100%
LDA	2 sec	3sec	10 sec	1mins
QDA	1 sec	3 sec	20 sec	2 mins
KNN	15 sec	21 mins	1 day	1 week
CT	1 sec	7 sec	2 mins	45 mins
BTs	18 sec	1 min	25 mins	20 hours
Bagging	2 sec	40 sec	20 mins	8 hours
RF	1 sec	20 sec	8 mins	3 hours
SVM	31 sec	2 hours	$>$ 1 day	$>$ 2 weeks

Table 4.2: The averaged computational time of every job (including model construction and testing using the specified size of data for once) for data set **YearPredictionMSD**.

Models	Sampling Fractions				
	0.01%	0.1%	1%	10%	100%
LDA	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec
QDA	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec
KNN	12 sec	1 min	15 mins	6 hours	$>$ 1 day
CT	$<$ 10 sec	$<$ 10 sec	$<$ 10 sec	12 sec	3 mins
BTs	$<$ 10 sec	12 sec	1 mins	10 mins	8 hours
Bagging	$<$ 10 sec	$<$ 10 sec	1 min	10 mins	3 hours
RF	$<$ 10 sec	$<$ 10 sec	1 min	10 mins	3 hours
SVM	$<$ 10 sec	20 sec	20 mins	10 hours	2 days

Table 4.3: The averaged computational time of every job (including model construction and testing using the specified size of data for once) for data set **PokerHand**.

Chapter V

Conclusion

In this thesis, we studied how the number of training observations affect the performance of some popular statistical learning algorithms. Three datasets were considered. In dataset **PokerHand**, the learning algorithms need to learn the rules of 'poker hands'. So, if the algorithms, such as the classification trees and random forests, has the potential to learn these rules, their classification rates increased significantly given more training observations, although the gains tends to be smaller. However, if the algorithms do not have the ability to learn the rules, such as LDA and SVM, giving more training observations will not bring too much benefits. So, in different applications, background knowledge can help us to choose more efficient algorithms, and just giving an inefficient algorithm more training observations might not bring too much benefit. In datasets **HepMass** and **YearPredictionMSD**, the simulation study showed that the gain of using large number of training observations is limited. All algorithms are tend to converge to their limit given enough training data, and more training observations can not bring too much gain.

Through comparing the performance of the algorithms, we find that the methods that can produce classifiers with complex boundary perform well when there are many training observations, such as classification trees and the random forests. This reveal the fact that the big datasets are more likely to have more complex feature and patterns, as a result, more powerful algorithms, which can construct classifiers with complex boundaries and learn the patterns efficiently and automatically, are preferable.

The main challenge of this thesis was the limited computing resources. Besides the number of cores, the amount of memory storage required for some algorithms increased exponentially with the number of observations. Displaying even a simple statistical learning method on larger dataset can be challenging. Through monitoring the computational usage of different

algorithms on various sizes of training data, we found LDA, QDA and the single classification tree approach (CT) were more computationally efficient in terms of computational time and memory usage. The memory usage of Boosting Trees and SVM were relative small, but far more computational time required. Random forest and Bagging Trees can be processed faster than SVM, but more memory required. However, the whole process of them can be easily break down because the trees were built independently using bootstrapped samples, such as the 'updating' approach used in this study.

Compared to the other algorithms considered in this study, we would like to recommend the Classification Tree and the Random Forest in big data situations, because both of them are able to construct classifier with complex boundary capture the complex patterns of big data efficiently. In addition, the computational time of them is relative short. For users with limited computational resources, classification trees (CT) are recommended. Otherwise, Random Forests are preferred, because they have a stronger 'learning' ability, compared to CT, but more powerful computers required.

In this thesis, we showed that increasing the number of observations improved the performance of the algorithms, but the gain tends to be very small, especially when the sample size is already very large. For example, in dataset **HepMass**, the accuracy improvement of the model using large data taking about 2 weeks is only about 1%, comparing to the one using smaller data which just took about half an hour. In addition, for some ensemble methods, such as Boosting Trees, more base classifiers and lower learning rate are required when the data is large. Without adjusting these parameters, boosting methods can even give a worse result given more observations. However, using lower learning rate can be extremely computational expensive. So, in applications, using large dataset might be not necessary to get a model with acceptable accuracy. Learning curves, which is used to describe the relationship between the sample size and model accuracy, are discussed. In our study, the 3-parameter Power Law functions fit the learning curves well, although more datasets and algorithms need to be further considered.

5.1 Future Research

In this thesis, we considered 3 classification datasets. It would be interesting to extend the simulation study to consider more datasets and more algorithms. In addition, future research can also consider the regression problems.

Learning curves provide way to describe the relationship between the sampling size and model accuracy. However, we also need to look at the variations of them. One potential direction for further research is whether can we mathematically prove this the relationship can be described as function families.

It would be interesting and valuable to investigate whether can we use the learning curves to predict the required number of observations to get a model with good prediction performance. That is, fitting a learning curve using small samples and determining the number of training observations required.

References

[1] Friedman, J., Hastie, T. and Tibshirani, R., 2001. The elements of statistical learning (Vol. 1). Springer, Berlin: Springer series in statistics.

[2] James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. An introduction to statistical learning (Vol. 6). New York: springer.

[3] Ghannadpour, S.S. and Hezarkhani, A., 2016. Exploration geochemistry data-application for anomaly separation based on discriminant function analysis in the Parkam porphyry system (Iran). Geosciences Journal, 20(6), pp.837-850.

[4] Bache, K. and Lichman, M. (2013). UCI machine learning repository.

[5] Slavakis, K., Giannakis, G.B. and Mateos, G., 2014. Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge. IEEE Signal Processing Magazine, 31(5), pp.18-31.

[6] Cook, S., Conrad, C., Fowlkes, A.L. and Mohebbi, M.H., 2011. Assessing Google flu trends performance in the United States during the 2009 influenza virus A (H1N1) pandemic. PloS one, 6(8), p.e23610.

[7] Bruell A. The next big ad platform: Retailer sites? ADS FOR BRANDS AND EXCHANGES FATTEN MARGINS FOR TARGET, AMAZON AND MORE. Advertising Age. 2015;86(21):16.

[8] Kitchin, R., 2014. The real-time city? Big data and smart urbanism. GeoJournal, 79(1), pp.1-14.

[9] Boyd, D. and Crawford, K., 2011, September. Six provocations for big data. In A decade in internet time: Symposium on the dynamics of the internet and society (Vol. 21). Oxford: Oxford Internet Institute.

[10] Kitchin, R., 2014. The data revolution: Big data, open data, data infrastructures and their consequences. Sage.

[11] Provost, F. and Fawcett, T., 2013. Data science and its relationship to big data and data-driven decision making. Big Data, 1(1), pp.51-59.

[12] Fan J, Han F, Liu H. Challenges of big data analysis. National science review. 2014 Jun 1;1(2):293-314.

[13] Bengtsson, T., Bickel, P. and Li, B., 2008. Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems. In Probability and statistics: Essays in honor of David A. Freedman (pp. 316-334). Institute of Mathematical Statistics.

[14] Muller, Klaus-Robert, Mika,S. . An introduction to kernel-based learning algorithms. In Handbook of Neural Network Signal Processing. CRC Press.

[15] Beaver, D., Kumar, S., Li, H.C., Sobel, J. and Vajgel, P., 2010, October. Finding a Needle in Haystack: Facebook's Photo Storage. In OSDI (Vol. 10, No. 2010, pp. 1-8).

[16] Gandomi, A. and Haider, M., 2015. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35(2), pp.137-144.

[17] Chen, M., Mao, S. and Liu, Y., 2014. Big data: A survey. Mobile Networks and Applications, 19(2), pp.171-209.

[18] Murdoch, T.B. and Detsky, A.S., 2013. The inevitable application of big data to health care. Jama, 309(13), pp.1351-1352.

[19] Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J., 2016. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.

[20] Hilbert, M. and López, P., 2011. The world's technological capacity to store, communicate, and compute information. science, 332(6025), pp.60-65.

[21] Data, I.B., 2015. IBM-Bringing Big Data to the Enterprise.

[22] Weinberger, K.Q. and Saul, L.K., 2009. Distance metric learning for large margin nearest neighbor classification. Journal of Machine Learning Research, 10(Feb), pp.207-244.

[23] Jain, A.K., Duin, R.P.W. and Mao, J., 2000. Statistical pattern recognition: A review. IEEE Transactions on pattern analysis and machine intelligence, 22(1), pp.4-37.

[24] Böhm, C. and Krebs, F., 2004. The k-nearest neighbour join: Turbo charging the KDD process. Knowledge and Information Systems, 6(6), pp.728-749.

[25] Ripley, B.D., 2007. Pattern recognition and neural networks. Cambridge university press.

[26] Rokach, L. and Maimon, O., 2014. Data mining with decision trees: theory and applications. World scientific.

[27] Breiman, L., Friedman, J.H., Stone, C.J. and Olshen, R.A., 1984. Classification and regression trees, new ed.

[28] Breiman, L., 1996. Bagging predictors. Machine learning, 24(2), pp.123-140.

[29] Goldstein, B.A., Polley, E.C. and Briggs, F., 2011. Random forests for genetic association studies. Statistical applications in genetics and molecular biology, 10(1).

[30] Díaz-Uriarte, R. and De Andres, S.A., 2006. Gene selection and classification of microarray data using random forest. BMC bioinformatics, 7(1), p.3.

[31] Goldstein, H., Kounali, D. and Robinson, A., 2008. Modelling measurement errors and category misclassifications in multilevel models. Statistical Modelling, 8(3), pp.243-261.

[32] Goldstein, B.A., Hubbard, A.E., Cutler, A. and Barcellos, L.F., 2010. An application of Random Forests to a genome-wide association dataset: methodological considerations and new findings. BMC genetics, 11(1), p.49.

[33] Elith, J., Leathwick, J.R. and Hastie, T., 2008. A working guide to boosted regression trees. Journal of Animal Ecology, 77(4), pp.802-813.

[34] Breiman, L., 2001. Using iterated bagging to debias regressions. Machine Learning, 45(3), pp.261-277.

[35] Wolpert, D.H. and Macready, W.G., 1999. An efficient method to estimate bagging's generalization error. Machine Learning, 35(1), pp.41-55.

[36] Ho, T.K., 1998. The random subspace method for constructing decision forests. IEEE transactions on pattern analysis and machine intelligence, 20(8), pp.832-844.

[37] Breiman, L., 2001. Random forests. Machine learning, 45(1), pp.5-32.

[38] Gu, B., Hu, F. and Liu, H., 2001, July. Modelling classification performance for large data sets. In International Conference on Web-Age Information Management (pp. 317-328). Springer Berlin Heidelberg.

[39] Haussler, D., Kearns, M., Seung, H.S. and Tishby, N., 1996. Rigorous learning curve bounds from statistical mechanics. Machine Learning, 25(2-3), pp.195-236.

[40] Provost, F., Jensen, D. and Oates, T., 1999, August. Efficient progressive sampling. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 23-32). ACM.

[41] Bertin-Mahieux, T., Ellis, D.P., Whitman, B. and Lamere, P., 2011, October. The Million Song Dataset. In ISMIR (Vol. 2, No. 9, p. 10).

[42] Zaki, M.J., Parthasarathy, S., Li, W. and Ogihara, M., 1997, April. Evaluation of sampling for data mining of association rules. In Research Issues in Data Engineering, 1997. Proceedings. Seventh International Workshop on (pp. 42-50). IEEE.

[43] Freund, Y. and Schapire, R.E., 1995, March. A desicion-theoretic generalization of on-line learning and an application to boosting. In European conference on computational learning theory (pp. 23-37). Springer Berlin Heidelberg.

[44] Cortes, C. and Vapnik, V., 1995. Support-vector networks. Machine learning, 20(3), pp.273-297.

[45] Vapnik, V., 2013. The nature of statistical learning theory. Springer science and business media.

[46] Vapnik, V.N., 1998. Statistical learning theory. Adaptive and learning systems for signal processing, communications, and control.

[47] Hilbert M, López P. The world's technological capacity to store, communicate, and compute information. Science. 2011;332(6025):60-5.

[48] "IBM What is big data? – Bringing big data to the enterprise". www.ibm.com. Retrieved 2013-08-26

[49] Hilbert M, López P. The world's technological capacity to store, communicate, and compute information. Science. 2011;332(6025):60-5.

[50] Herodotou H, Lim H, Luo G, Borisov N, Dong L, Cetin FB, et al. Starfish: A self-tuning system for big data analytics. ; 2011.

[51] Duan, Kaibo, S. Sathiya Keerthi, and Aun Neow Poo. "Evaluation of simple performance measures for tuning SVM hyperparameters." Neurocomputing 51 (2003): 41-59.

[52] R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

[53] Revolution Analytics and Steve Weston (2014). doParallel: Foreach parallel adaptor for the parallel package. R package version 1.0.8. http://CRAN.R-project.org/package=doParallel

[54] Revolution Analytics and Steve Weston (2014). foreach: Foreach looping construct for R. R package version 1.4.2. http://CRAN.R-project.org/package=foreach

[55] Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

[56] Klaus Schliep & Klaus Hechenbichler (2014). kknn: Weighted k-Nearest Neighbors. R package version 1.2-5. http://CRAN.R-project.org/package=kknn

[57] Terry Therneau, Beth Atkinson and Brian Ripley (2015). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-9. http://CRAN.R-project.org/package=rpart

[58] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

[59] Greg Ridgeway with contributions from others (2015). gbm: Generalized Boosted Regression Models. R package version 2.1.1. http://CRAN.R-project.org/package=gbm

[60] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2014). e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. R package version 1.6-4. http://CRAN.R-project.org/package=e1071

Appendices

Appendix I

In this section, we showed why the sum of squared residual of the polynomial regression is small than that of the linear regression. Consider a linear regression problems with n observations and p predictors. Denote the vector of response $\mathbf{Y}_{n\times 1}$, the design matrix $\mathbf{X}_{n\times p}$ and vector of residuals $\boldsymbol{\epsilon}_{n\times 1}$. Suppose $\mathbf{X}$ is a full rank matrix and $\text{rank}(\mathbf{X}) = p$ then

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

and Least Squared Estimate of coefficients are $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. So the sum of squared residual

$$\begin{aligned}
\boldsymbol{\epsilon}^T\boldsymbol{\epsilon} &= [\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}]^T[\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}] \\
&= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T\mathbf{X}\hat{\boldsymbol{\beta}} + \hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} \\
&= \mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{Y} \\
&= \mathbf{Y}^T[\mathbf{I}_n - \mathbf{X}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}]\mathbf{Y} \\
&= (n-p)\mathbf{Y}^T\mathbf{Y}
\end{aligned}$$

where $\mathbf{I}_n$ is the $n \times n$ identify matrix and $p = \text{rank}(\mathbf{X})$. If we fit a polynomial regression, denote new design matrix $\mathbf{A}$ and $\text{rank}(\mathbf{A}) = p'$, then because $\text{span}(\mathbf{X}) \in \text{span}(\mathbf{A})$, so $p' > p$. So the new sum of squared residual of it will be $(n-p')\mathbf{Y}^T\mathbf{Y}$, which is smaller than $(n-p)\mathbf{Y}^T\mathbf{Y}$.

Appendix II

The reason of setting in boosting

In this section, we will discuss the reason of the setting in AdaBoost algorithm. In general, the AdaBoost algorithm is equal to fitting a forward stepwise additive model using the exponential loss function

$$L(y, G_k(\mathbf{x})) = \exp\{-yG_k(\mathbf{x})\},$$

where the response $y \in \{-1, 1\}$.

Suppose after k^{th} iterations, the current weighs for each observations are $w_i^k, i = 1, 2, ..., n$, and we have a sequence of weak classifiers and their weights, $\{G_1(X), \alpha_1\}$, $\{G_2(X), \alpha_2\}$,...,$\{G_k(X), \alpha_k\}$. Then, the current ensemble classifier is

$$G_{(k)}(X) = \alpha_1 G_1(X) + \alpha_2 G_2(X) +, ..., +\alpha_k G_k(X).$$

the total error of $G_{(k)}(X)$ based on exponential loss function is

$$E_k = \sum_{i=1}^{n} \exp\{-y_i G_{(k)}(\mathbf{x}_i)\},$$

which is the weighted sum of loss for each observation. So, for observations correctly classified by $G_k(X)$, $y_i = G_k(\mathbf{x}_i) \Rightarrow y_i G_k(\mathbf{x}_i) = 1$, and for the misclassified observations by $G_k(X)$, $y_i \neq G_k(\mathbf{x}_i) \Rightarrow y_i G_k(\mathbf{x}_i) = -1$. To investigate the reason of settings for weights and the relationship between the iteration steps, let's consider the total loss of $G_{(k+1)}(X)$, given $G_{(k)}(X)$.

$$\begin{aligned} E_{k+1} &= \sum_{i=1}^{n} \exp\{-y_i G_{(k+1)}(\mathbf{x}_i)\} \\ &= \sum_{i=1}^{n} \exp\{-y_i [G_{(k)}(\mathbf{x}_i) + \alpha_{k+1} G_{k+1}(\mathbf{x}_i)]\} \end{aligned}$$

Let $w_i^{k+1} = \exp\{-y_i G_{(k)}(\mathbf{x}_i)\}$, then,

$$\begin{aligned} E_{k+1} &= \sum_{i=1}^{n} w_i^{k+1} \times \exp\{-\alpha_{k+1} y_i G_{k+1}(\mathbf{x}_i)\} \\ &= \sum_{y_i = G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{-\alpha_{k+1} y_i G_{k+1}(\mathbf{x}_i)\} + \\ &\quad \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{-\alpha_{k+1} y_i G_{k+1}(\mathbf{x}_i)\} \\ &= \sum_{y_i = G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{-\alpha_{k+1}\} + \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{\alpha_{k+1}\} \\ &= \sum_{i=1}^{n} w_i^{k+1} \times \exp\{-\alpha_{k+1}\} - \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{-\alpha_{k+1}\} + \\ &\quad \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times \exp\{\alpha_{k+1}\} \\ &= \sum_{i=1}^{n} w_i^{k+1} \times \exp\{-\alpha_{k+1}\} + \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times [\exp\{\alpha_{k+1}\} - \exp\{-\alpha_{k+1}\}] \end{aligned}$$

From the equation above, we can find that in terms of $G_{k+1}(X)$, minimizing E_{k+1} is equivalent to minimizing $\sum_{y_i \neq G_{k+1}(x_i)} w_i^{k+1}$, which is the sum of a fuction w_i^{k+1} (which will be shown as the i^{th} observation's weight at the $k+1^{\text{th}}$ step) of the observations misclassified by $G_{k+1}(X)$. Next, we consider α_{k+1}. Let

$$\frac{\partial E_{k+1}}{\partial \alpha_{k+1}} = 0,$$

We have

$$\sum_{i=1}^{n} w_i^{k+1} \times e^{-\alpha_{k+1}} = \sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1} \times [e^{\alpha_{k+1}} + e^{-\alpha_{k+1}}]$$

$$\alpha_{k+1} = \frac{1}{2} \log \frac{\sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1}}{\sum_{y_i = G_{k+1}(\mathbf{x}_i)} w_i^{k+1}}.$$

If we define the weighted error rate of $G_{k+1}(X)$,

$$\text{error}_{k+1} = \frac{\sum_{y_i \neq G_{k+1}(\mathbf{x}_i)} w_i^{k+1}}{\sum_{i=1}^{n} w_i^{k+1}},$$

then,

$$\alpha_{k+1} = \frac{1}{2} \log \frac{1 - \text{error}_{k+1}}{\text{error}_{k+1}}.$$

From the results above, we know how the weak classifiers, $G_k(X)$ and their weights, α_k are determined. Next, we will consider w_i^k, the weighs of each observation at each step in the iteration. In the former setting, the weights of each observations are set to

$$w_i^{k+1} = \exp\{-y_i G_{(k)}(x_i)\}.$$

So,

$$\frac{w_i^{k+1}}{w_i^k} = \frac{\exp\{-y_i G_{(k)}(\mathbf{x}_i)\}}{\exp\{-y_i G_{(k-1)}(\mathbf{x}_i)\}}$$

$$w_i^{k+1} = w_i^k \times \exp\{-\alpha_k y_i G_k(\mathbf{x}_i)\}.$$

For the observations that correctly classified by the last weak classifier, $G_k(X)$, satisfying $y_i G_k(\mathbf{x}_i) = 1$. Meanwhile,

$$\alpha_k = \frac{1}{2}\log\frac{\sum_{y_i \neq G_k(\mathbf{x}_i)} w_i^k}{\sum_{y_i = G_k(\mathbf{x}_i)} w_i^k} = 0,$$

Because $\sum_{y_i \neq G_k(\mathbf{x}_i)} w_i^k = 0$, when all of them are correctly classified by $G_k(X)$. So, the weights of the next iteration are

$$\begin{aligned} w_i^{k+1} =& w_i^k \times \exp\{-\alpha_k\} \\ =& w_i^k \end{aligned}$$

For the observations that correctly classified by the last weak classifier, $G_k(X)$, satisfying $y_i G_k(\mathbf{x}_i) = -1$. So, the weights of the next iteration are

$$w_i^{k+1} = w_i^k \times \exp\{\alpha_k\}.$$

Thus, we showed that AdaBoost algorithm is equivalent to fitting a forward stepwise additive model, minimizing the exponential loss. Comparing to minimizing the classification rate, the exponential loss is more sensitive estimated probabilities to each class. Friedman et al.(2000) showed that the AdaBoosting ensemble model, $G(x) = \text{sign}\{\sum_{k=1}^K \alpha_k G_k(X)\}$, is estimating

$$\frac{1}{2}\log\frac{P(Y=1|X=x)}{P(Y=-1|X=x)},$$

Which are the the half of the log of the odds. So the normal cut-point for the predicting classification is 0.

Appendix III

An Example of R code for Parallel Computing in SVM

```
library(foreach)
library(doParallel)
library(e1071)
library(readr)

YPdata <- read_csv("/scratch-network/yli226/Big_data_sets/YearPredictionMSD.txt")

YPdata$V1 <- as.numeric(YPdata$V1)

YPdata$V1 <-ifelse(YPdata$V1>=2000,1,0)
YPdata$V1 <- as.factor(YPdata$V1)


train <-YPdata[1:463715,]
test <-YPdata[463716:515345,]

################################################################################
#######  Specify the 'job', and what need to be recorded  (For SVM )######
################################################################################

svm0 <- function(sizes) {
  f <- matrix(ncol=7,nrow=1)
  timing <- proc.time()
  index <- sample(1:nrow(train),sizes,replace=F)
  samples <- train[index,]
  f[1,1] <- sizes
  tune.out1 <- tune (svm, V1~., data =samples,
                     ranges = list (degree=c(1,2,3),kernel =c('linear','polynomial','radial')
                     ))
  bestmod <- tune.out1$best.model
```

```
  degree1 <- as.numeric(tune.out1$best.parameters[1])
  kernel1 <- as.numeric(tune.out1$best.parameters[2])
  CV_Error <- 1-tune.out1$best.performance
  Test_Error <- mean(predict(bestmod,test) == test$V1)
  f[1,2] <- CV_Error
  f[1,3] <- Test_Error
  Time <- (proc.time() - timing)[3]
  f[1,4] <-Time
  f[1,5] <- date()
  f[1,6] <- degree1
  f[1,7] <- kernel1
  return(as.matrix(f))
}

#################################################################
#######  Genearating the geometric sequence sizes of data   ####
#################################################################

a <- c(0.001*2^(seq(0,log(1000)/log(2),by=1)),1)
b <- round(463715*a)
b

#################################################################################
#######  The sizes and repeats to be run for each 'worker' in this run  ######
#################################################################################

sizes <- b[1:5]
repeats <- 5

#########################################################
######  Specify the number of cores to be used  ######
#########################################################

cl <- 10
registerDoParallel(cl)

#########################################################################
######  Specify the direction of output files and their names   ######
#########################################################################

file0 <- '/scratch-network/yli226/Results_YearP/Parallel_results/YPdata_50_SVM'
```

```
################################################
###### The function to run in parallel     ######
################################################

m <-repeats
n <- length(sizes)

registerDoParallel(cl)


foreach(index = 1:cl,.combine=rbind)  %dopar%
{matrixAA <- matrix(ncol=7,nrow= m*n)
 colnames(matrixAA) <-c('Sampling.size','CV.Error','Rate','Time','Date','Degree','Kernel')
 for(i in 1: m){
   for(j in 1:n){
     file1 <- paste0(file0 ,min(sizes),'-',max(sizes), '_Core_',index, ".txt")
     write.table(matrixAA , file = file1 , row.names = F,col.names = T,sep=',')
     size <- sizes[j]
     matrixAA[m*(j-1)+i,] <- svm0(size)
     write.table(matrixAA , file = file1 , row.names = F,col.names = T,sep=',')
   }
 }
}

#####################################################################
###   Merging Results in to one file with order of sample sizes  ####
#####################################################################

library(data.table)

path0 <- '/scratch-network/yli226/Results_YearP/Parallel_results/'

files <- list.files(path = path0,
                    pattern = c('YPdata_50_SVM',".txt"))

temp <- lapply(paste0(path0,files), fread, sep=",")
data <- rbindlist( temp )

data$Sampling.size <- as.numeric(data$Sampling.size )
```

```
newdata <- data[order(Sampling.size),]


write.table(newdata ,   file = paste0(file0 ,'_',min(sizes),'-',max(sizes), ".txt"),
            row.names = F, col.names = T, sep=',')
```