FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

# ENSEMBLE METHODS FOR CAPTURING DYNAMICS OF LIMIT ORDER BOOKS

By

JIAN WANG

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2017

Jian Wang defended this dissertation on July 18 , 2017.
The members of the supervisory committee were:

Jinfeng Zhang

Professor Co-Directing Dissertation

Giray Ökten

Professor Co-Directing Dissertation

Alec Kercheval

Committee Member

Washington Mio

Committee Member

Capstick Simon

University Representative

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

# ACKNOWLEDGMENTS

I am greatly grateful to many people during my life to pursue this degree.

First and foremost, I want to express my gratitude to Dr.Jinfeng Zhang, my thesis major professor. Without his patience and insight, it is impossible to complete this thesis. His profound knowledge in statistics and programming, smart vision in research direction and passion for both work and life has benefited me a lot. Dr. Zhang has provided every help I would imagine from a research advisor and it was an enjoyable experience of working with him.

I also deeply appreciate my co-advisor Dr. Giray Ökten, he put a lot of effort on this thesis and provided much insightful advice. Besides the solid knowledge of Monte Carlo methods he taught me, he also helped me so much from the very first day that I joined the department, including selecting courses, career preparation, and job search. I feel very fortunate to have Dr. Ökten as my co-advisor for accomplishing this thesis.

I want to extend my thanks to Dr.Alec N. Kercheval and Dr.Mio as my committee members and every course I had at our department in Florida state university is an excellent experience and I would like to thank all my instructors for their patience and effort: Dr.Kopriva, Dr. Nichols, Dr.Ewald, Dr.Aldrovandi , Dr.Fahim, Dr.Gallivan, Dr.Kim, Dr.Sussman and Dr.Zhu. Those courses help me to cultivate solid background of financial math, numerical analysis, and programming.

Furthermore, I owe my special thanks to Dr. Penelope Kirby, Dr. Penny LeNoir, Ms.Blackwelder, Mr. Dodaro, Mr. Grigorian and Mr. Wooland, who gave me a lot of instructions and suggestions when I worked as a teaching assistant in our department and mathematics lab. They are all very nice and point out my weak points during work, which gave me a good opportunity to enhance my presentation skills and teaching abilities.

I also want to thank my colleagues and friends at department: Yang Liu, He Huang, Jinghua Yan, Bo Zhao, Jian Geng, Ming Zhu, Wanwan Huang, Yuan Zhang, Wen Huang, Yaning Liu,

Qiuping Xu, Guifang Zhou, Pierre Garreau, Ahmed Islim, Dawna Jones, Linlin Xu, Yuying Tzeng, Szu-Yu Pai, Nguyet Nguyen, Dong Sun, Daozhi Han, Yuanda Chen, Chun-Yuan Chiu, Chenchen Zhou, Yao Dai, Chaoxu Pei, Fangxi Gu, Zailei Cheng, Jian Li, Xinru Yuan, Haixu Wang, Wei Meng and Qi Si for their help, cooperation and meaningful suggestions to my research.

Finally, I want to thank my father, Xinshan Wang, my mother, Hua Yang and my other family members for their supports during my Ph.D study period. They are the origin of all my motivation, confidence and strength.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ALGORITHMS

# ABSTRACT

Because of rapid developments in information technology, the limit order books (LOBs) mechanism has emerged to prevail in today's financial market. In this paper, we propose ensemble machine learning architectures for capturing the dynamics of high-frequency limit order books, such as predicting price- spread-crossing opportunities in a future time interval. The paper is more data-driven oriented, so experiments with five real-time stock data from NASDAQ, measured by nanoseconds, are established. The data samples are divided into training and validation parts, and we define the structure of our models by these two datasets. Compared with other models, such as logistic regression, support vector machine (SVM),etc., our out-of-sample testing results have shown that ensemble methods had better performance on both statistical measurements and computational efficiency. A simple trading strategy based on our models has shown excellent profit and loss (PnL) results. Although this paper focuses on LOBs, the similar frameworks and processes can be extended to other classification research areas.

**Keywords:** limit order books, high-frequency trading, data analysis, ensemble methods, F1 score.

# CHAPTER 1

# INTRODUCTION

## 1.1 High-frequency trading system

### 1.1.1 Evolution of high-frequency trading

Over the last few decades, information technology, including computing speed and memory volume, has developed a lot. According to this trend, a new class of trading system, which is called high-frequency trading (HFT) has appeared in today's financial markets. Generally speaking, HFT represents a program trading platform that uses powerful computers to transact a great number of orders at very fast speeds. It becomes more and more attractive because of some key factors:
1)High-frequency trading can help to narrow Spreads. In 2001, the unit of quoting prices in U.S. Stock exchanges changed from fractions to decimals. So the minimum spread between the bid and ask prices decreased from 1/6th of a dollar (6.25 cents) to one cent. The change of price unit provides traders better alternatives to seeking spread arbitrages, which results in a strong boost in an algorithmic trading system.
2) Regulations have changed to a favorable direction. In 2005, the Securities and Exchange Commission (SEC) passed the Regulation National Market System (Reg.NMS), which improved transparency and competition among different financial markets. In addition, this regulation also required trade orders to be posted nationally instead of at individual exchanges. So traders can benefit from profit os small price difference of one security among different exchanges.

High-frequency trading is an extension case of algorithmic trading, which turns over small positions of security very frequently. The U.S. Securities and Exchanges Commission conclude specific characteristics of HFT:

- Submit some orders and cancel them soon after the submission.
- Maintain very few or no overnight positions.

- Maintain a very high turnover rate of very small positions of a specific security or even a pool of securities. Meanwhile, the holding period for each position is also very short.
- Utilize complicated and high-performance computing program to generate, execute or cancel orders.
- Utilize individual data from exchanges and servers that belong to co-location providers that minimize network or other types of latencies.

According to a recent survey (Agarwal (2012)), high-frequency trading has taken a great number of shares in the U.S. and European equity trading volume. As shown in figure 1.1, in U.S., the percentage of HFT in equity turnover by volume maintained growth trend overall from the year 2005 to 2010. For example, in 2010, HFT occupied 56% by volume of the entire equity turnover, increased from 21 % in 2005.

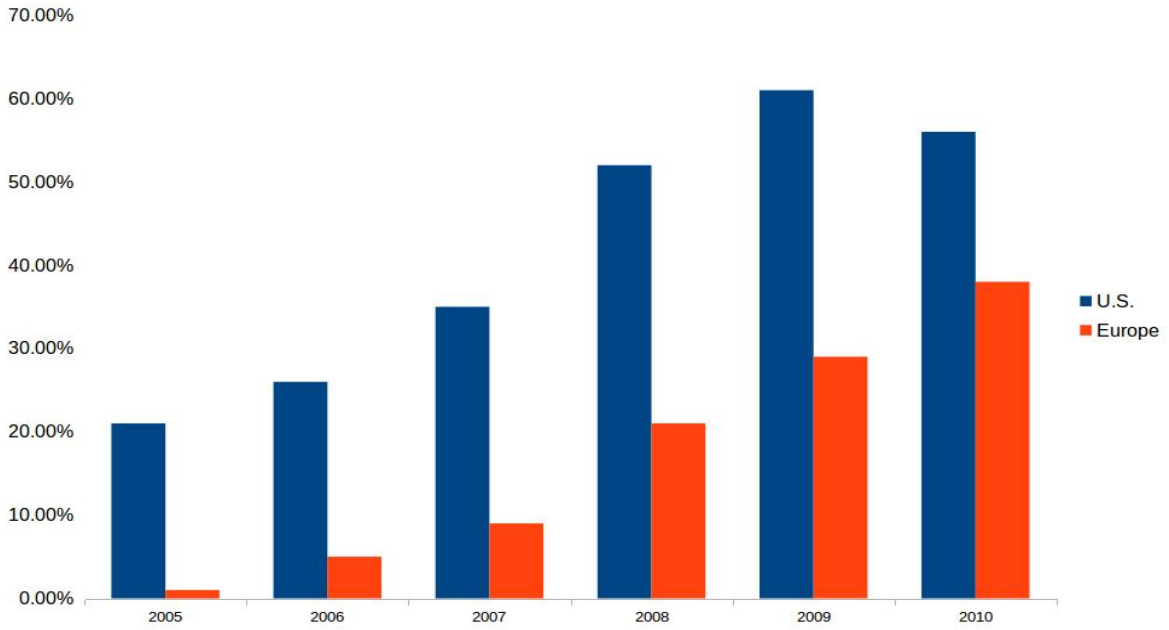


Figure 1.1: High-frequency trading as a % of equity turnover by volume, U.S. and by value, Europe 2005-2010

A similar situation happened in Europe. HFT only accounted for 1% of equity turnover by value in 2005. The percentage surged to 38 % in 2010, increased from 29 % in the previous year.

### 1.1.2 High-frequency trading strategies

Generally speaking, there are two main high-frequency trading strategies: passive and aggressive trading strategies. Passive strategies use limit orders which allow the brokerage to buy or sell the stocks at a specified level price, while aggressive strategies utilize market orders to buy or sell the stocks immediately. In the following, we give the description of main types of these two trading strategies.

For passive high-frequency trading strategies, the first widely used method is market making. This method allows the market maker to purchase a company's securities and, at the same time, the market maker also acts as an underwriter of the securities in a secondary public offering. Since the market maker can place bids on a security before its publication, the real buyers would have to place their buy orders higher than the market maker's bid price. Therefore the market maker can benefit from this higher opening. The second method for passive strategies is arbitrage trading. As described by its name, this method makes a profit by the price difference of the same or related securities. A simple example is as follows: the stock of one company that we called X is trading at \$10 on the New York Stock Exchange (NYSE), while at the same moment it is trading at the price of \$10.05 on the London Stock Exchange (LSE). Given that a trader can trade the stock on both markets with no time lag, he can buy one specific security on the NYSE and sell the same shares on the LSE just after buying. Obviously, a profit of 5 cents without risk can be earned by him. Those arbitrage opportunities will persist until the specialists on the NYSE or LSE adjust their price to eliminate the price difference.

The main type of aggressive high-frequency trading strategies consists of the following two types: Momentum ignition and order anticipate. The former one is a strategy that a proprietary trading firm buy or sell volumes of orders that will cause the price of the underlying security to significantly go up or down shortly. Such quick submission and cancellation of many orders of a stock will trigger other traders' algorithms to buy or sell the same stock more aggressively. After the trend of price movement is made in the market, the momentum maker will benefit from selling the stock at a higher price or buying the stock at a lower price. However, it is very difficult to distinguish between momentum ignition and *spoof*, which was defined as illegal according to the Dodd-Frank Act. For the order anticipate method, it can be described as a liquidity detection trading which confirms the existence of large institutional buyers or sellers in the marketplace and then trades ahead of these

buyers or sellers in anticipation that their large orders will move market prices (See Securities and Exchange Commission, 2014, p.8). The line between this method and another illegal action called *front-running* can be nuanced. *Front-running* is the unethical practice that a stockbroker trades securities in his private account based on the knowledge of advance knowledge of pending orders from its customers.

## 1.2 LOB dynamics

According to the above section, we know that under the aggressive strategies situation, it is very likely that the strategy will be deemed as illegal. Therefore choosing passive strategies in high-frequency trading is a good idea. Since most transactions in the passive trading strategies are buying and selling the securities at a particular price, LOBs play an indispensable role in this strategy. Actually, in today's financial market, more than half of stock exchanges now use a LOB mechanism to facilitate trade (Rosu (2009)). Some exchanges, such as Helsinki, Hong Kong, Shenzhen, Swiss, Tokyo, Toronto, and Vancouver Stock Exchanges, now use pure LOBs (Luckock (2001)). Some exchanges use hybrid of LOBs, which include the New York Stock Ex-change (NYSE), NASDAQ, and the London Stock Exchange (LSE) (Cont et al., 2010).

As described above, we can see that LOBs play a major role in financial trading architectures, it is beneficial for both scholars and practitioners to understand the dynamics of LOB. The advantages of capturing dynamics of LOBs include: finding an optimal opportunity to execute orders(Obizhaeva and Wang, 2013), improving the performance of electronic trading algorithms(Engle et al., 2006), Obtaining a better understanding of micro-market structure for Practitioners(Harris, 2003), and finally getting a clearer insight into market volatility(Kirilenko et al., 2015).

In this thesis, we first introduce basic definitions of LOBs, including math definition of the bid-ask spread, mid price, bid and ask side depth, spread crossing opportunities and so on. Then we use statistical and data driven methods, especially in the area of machine learning architectures, to model the future arbitrage opportunities of LOBs. We focus on ensemble machine learning algorithms to build our prediction models. As far as we know, there is no evidence shows that these methods were used to study LOBs' research. The introduction of ensemble machine learning algorithms will be given in the following section of this chapter and details can be found in chapter 6.

## 1.3 Ensemble method for classifiers

In spite of a great amount of research on LOBs, there is only a limited amount of literature which utilizes machine learning methods for capturing the LOBs. Furthermore, based on our knowledge, there is little evidence that ensemble methods were allied to this topic. In our research, we use AdaBoost (Adaptive Boosting) method, which was deemed as the best classifier off the shelf (Kégl, 2013), and random forest method to predict spread crossing over opportunities of LOBs.

Generally speaking, an ensemble classifier contains a set of individually trained classifiers (such as neural networks or decision trees) to get a better predictive performance by combining the predicting results of each classifier. Some past research shows that ensemble classifier is more accurate than any of the individual classifiers which constitute the ensemble. For example, Hansen and Salamon (1990) and Hashem (1997) have conducted both theoretical and empirical research which demonstrated that a good ensemble of neural networks is more accurate than its primary classifier.

Like other machine learning classifiers, ensemble methods have both advantages and disadvantages. The main advantage of ensembles is that there is little probability that all classifiers will make the same mistake. Actually, if each error is made by a minority of the classifiers, you can obtain an optimal classification. Additionally, ensembles are very likely to reduce the variance of classifiers. Therefore, if the classification algorithms are sensitive to small changes in the training data, ensemble methods tend to be very helpful. The disadvantages of ensemble methods are also notable, among them the biggest might be the lack of interpretation (Bühlmann, 2012). A linear combination of an individual classifier is much harder to interpret than a single classifier.

In our research, we utilize two typical ensemble algorithms, AdaBoost and random forest, to train and predict the labeled spread crossing over in a fixed time interval of LOBs. The price spread crossing opportunities can be labeled as ask price lower, bid spread higher and no crossing over in a fixed time horizon. Therefore our problem is a multi-class classification case, the one against one and one against all methods will be introduced to solve the multi-class classification problem. The real time data is divided into two parts with the percentage of 9:1, to make a training dataset and testing data set respectively. In addition, features with price, volume and order book arrival intensity of each price level are created, so every data sample in training and the testing dataset is represented as a vector of features. Moreover, Precision, recall and F1 score are used to measure

the performance of the models, since the existence of arbitrage in a relatively long time interval is rare and our dataset can be treated as an imbalanced data.

Experiments with real-time data from NASDAQ show that the ensemble models built in our paper can not only predict the arbitrage opportunities with high accuracy, but also can improve the prediction performance compared to basic classifiers, such as logistic regression, support vector machine and decision trees. We also design a naive trading strategy in the testing time interval and demonstrate the Profit and Loss (PnL) of our models. The cumulative Pnl curve shows that the traders can obtain positive return with zero investment, which indicates that the statistical arbitrages can be found in our models.

## 1.4 Purpose of the dissertation

The aim of this dissertation is to construct ensemble machine learning tools which predict the price changes of LOBs in a fixed time interval by extracting non-linear features from the real time data. Statistical properties of LOBs are also studied in our paper to find the hidden pattern behind data. We also try to develop a model framework that is operational and automated to reduce the possibility of human intervention. To deal with the unbalanced data problem, like the arbitrage opportunities in all the transactions in our paper, we propose new measures such as precision, recall and f1 score to test the performance of our models. Moreover, two multi-class schemes, one vs. one and one vs. rest, are compared to choose the better one for the practical problem. Finally, we design a naive trading system based on our ensemble models to show their practical applications. The results of Profit and Loss (PnL) are demonstrated to test the performance of the models. Our ultimate objective is to design a completed trading framework, including data extracting, data cleaning, features extracting, model constructing, model validation, and out-of-sample trading, that can be used to help the investors design trading strategies during the real-time transactions. Our research in this dissertation is a specific example that uses the machine learning tools to solve the real financial problems, the similar framework, process and idea in this paper can be extended to other classification and prediction research areas. The main contributions of our work are summarized in the following:

1) The model structure developed in this dissertation is highly reproducible and scalable which means other machine learning tools, such as neural networks, deep learning, and reinforcement

learning methods can be easily integrated with our framework. Besides, our models can be transplanted on different platforms without spending too much effort.

2) Instead of predicting price change in fixed future events, such as predicting the price of the next 5th coming order, we focus on predicting the price change in future fixed time interval. In practice, the coming time of the future event is hard to control, therefore it is very difficult to conduct a trading strategy based on future events. So forecasting the price change based on future time interval is of practical significance.

3) We use real-time stock data in a very low latency; the minimum time change can be a nanosecond. In addition, the data volume that we used is also very high; each stock will have millions number of data samples. Therefore, the results of our experiment are more credible and convincing.

4) AdaBoost and random forest technique are used to improve the prediction performance. The experimental results show that ensemble methods will improve the model efficiency in a significant way compared to the basic machine learning tools.

5) Feature importance based on random forest tools of different stocks are compared in this research. We also give our explanation of why those features are important.

6) A trading strategy with our model is designed based on the real-time dataset. A back-testing process shows a good profit gain and loss result based on the strategy which will be beneficial for traders to make decisions in daily work.

To better present the aforementioned contributions, the structure of this dissertation is summarized as follows.

Chapter 1: Introduction: reviews the background and history of high-frequency trading, trading strategies of high-frequency trading, and the evolution of LOBs.

Chapter 2: Literature Review: Briefly reviews the past research work including statistical properties of LOBs, mathematical definition of LOBs and machine learning application in LOBs, etc.

Chapter 3: Mathematical description and statistical properties of LOBs: List the main mathematics definition of LOBs, such as bid price, ask price, and bid-ask spread. In addition, some statistical properties are also mentioned in this chapter.

Chapter 4: Basic machine learning tools: Introducing some classic machine learning tools that we use as benchmarks to the ensemble methods. Those tools include logistic regression, lasso regression, ridge regression, support vector machine and decision tree.

Chapter 5: Ensemble methods for prediction: Introducing properties of ensemble methods, especially for AdaBoost and random forest.

Chapter 6: Model framework and results:In this chapter, we describe the data set that we use to train and test the model, the way to construct the features, and the framework of our models. Besides, the measurement for model performance such as precision, recall, and F1 score are also introduced here. Finally, we show the profit and loss of a simple trading strategy by using our model to predict the future arbitrage opportunities.

# CHAPTER 2

# LITERATURE REVIEW

In this chapter, we review the past research in finding the laws of limit order books and applying machine learning tools to financial markets. In the first part, we summarize the papers which describe the models and rules of limit order book. The research work of using machine learning methods to capture the dynamics of limit order book is generalized in the second part.

## 2.1 Limit order books dynamics and modeling

There are a lot of study on capturing limit order book dynamics, including theoretical and experimental results. Through simulating the effect of different order book depth,Bollerslev and Domowitz (1993) discover that when the order book depth increases, the autocorrelation of spreads will also rise.

Hamao and Hasbrouck (1995) studied the patterns of intra-day trades and quotes for part of stocks on the Tokyo Stock Exchange. They claim that an order that is waiting to be executed has a greater impact on the price movement than the one that is executed immediately. Therefore, after the market orders are executed, the market price is likely to move in the same direction. Additionally, they find that there is always some delay in execution when the market orders become the limit orders. After that, they also investigate the limit orders in the NYSESuperDOT market and find that the market orders often have worse performance than the limit orders that are placed at or above the best quote.

Maslov and Mills (2001) investigate the order book data from NASDAQ and discover that the number of limit orders placed on the bid sides and the number of limit orders placed on the ask sides are often largely unequal; the disequilibrium will cause the price change in the short term.

Bouchaud et al. (2002) studied three stocks in the Paris Bourse and find that the price of order books obeys a power law near the price at present time and the mean of the distribution is diverging.

Zovko et al. (2002) study around two million orders from the London Stock Exchange. They found that a power law distribution can be applied to the difference between the limit price and the best price. Besides, the volatility of the price is positively correlated with the relative limit price levels.

Potters and Bouchaud (2003) investigate the order book data in NASDAQ and discover that the tail of the price of the incoming order shows a very slow recession. Besides, they also find that rather than the power-law, the link between price response and volume follows a logarithmic distribution in the British and French financial market.

In Australian stock market, Hall et al. (2004) find that the order book depth and the market activity will significantly influence the buy-side pressure as well as the sell-side pressure. Therefore, traders tend to utilize the information inside the order book to infer other market participants' trading strategies.

Weber and Rosenow* (2005) studied the order books on the Island financial market. They find that the price impact function, which is used to describe the phenomenon that the stock trading will cause price changes, is convex. They discover that price changes and order flow are strongly anti-correlated, which will reduce the price impact of market orders.

Boehmer et al. (2007) use sample stock trading data in different markets to verify if past execution quality will affect order routing decisions. Their results show that routing decisions are related to the execution quality. The market will receive more orders if their execution costs are low and can fill orders immediately.

Ganchev et al. (2010) and Laruelle et al. (2011) introduce a numerical algorithm to optimally split orders across liquidity dark pools. They also give the experimental validation of the algorithm by using execution data in a dark pool from a brokerage.

A continuous time stochastic model was proposed by Cont et al. (2010) to capture the dynamics of a limit order book. Matrix computation and Laplace transform methods are used to calculate the probability of different events. Numerical results with real-time high-frequency data are also provided to show the efficiency of their model to capture the short-term dynamics of a limit order book.

Guo (2013) provides a solution to an optimal placement problem in a limit order book. Two models, with or without price impact, are proposed in their paper. Besides, they give the solutions of both single-period and multi-period situations.

Cont and Kukanov (2013) also study the optimal order placement problem; they transform this problem to a convex optimization problem. They consider the fee structure, the state of order books, the order flow properties and the preference of a trader. Besides, they provide an explicit solution in the single exchange situation and propose a stochastic method in the multiple exchange situation.

## 2.2 Machine learning methods for capturing limit order book dynamics

There is not much past research in applying the machine learning tools to limit order books or financial applications in general. To my best knowledge, the earliest research in this area can be traced back to Hutchinson et al. (1994). They propose a nonparametric method, learning network, to price and hedge options. Besides, real-time data sample(S&P 500 futures options from 1987 to 1991) was used to test the efficiency of their network pricing method.

Nevmyvaka et al. (2006) study the millisecond limit order data from NASDAQ and propose a reinforcement learning model to conduct an optimized trade execution. Their results show that reinforcement learning can significantly improve the performance compared with some simpler methods of optimization.

Kercheval and Zhang (2015) propose a support vector machine scheme to predict the movement of price for limit order books. Nanosecond time scale data from NASDAQ was used to build their model and feature selection process was conducted to show the importance of each feature. They also design a single buy-low and sell-high trading strategy to verify the practical value fo the model. The result of profit and loss(PnL) through this strategy is promising.

Park and Van Roy (2015) demonstrate a confidence triggered regularized adaptive certainty equivalent (CTRACE) policy for execution and learning on the same time and propose a reinforcement learning algorithm for improving the efficiency in linear-quadratic control problems. Monte Carlo simulation is utilized to show the CTRACE method is better than the certainty equivalent policy.

Sirignano (2016) builds a deep neural network algorithm in $\mathbb{R}^d$ space. Around 500 U.S. stocks are trained and tested and a cluster with 50 GPUs is used to accelerate the speed of computing. The experimental results show that their model outperforms the logistic regression model with non-linear features, empirical model, and a standard neural network model.

Some other related literature include: using hidden Markov chain models into financial market, for example,Idvall and Jonsson (2008), Hassan et al. (2007),Hassan and Nath (2005),Landen (2000),Mamon and Elliott (2007),Zhang (2004),Bulla (2006),Rossi and Gallo (2006),Rydén et al. (1998); using artificial neural networks to make prediction in financial markets,e.g.,Trippi and Turban (1992),Kuan and Liu (1995),Walczak (2001), Shadbolt (2002) ;using support vector machine combined with different kernels to predict the financial time series changes. Those works can be found in Cortes and Vapnik (1995),Tay and Cao (2001),Tay and Cao (2002),Cao (2003),Kim (2003), Pérez-Cruz et al. (2003),Huang et al. (2005),Van Gestel et al. (2001),Hazarika and Taylor (2002),Tino et al. (2005),Huang and Wu (2006),Huang and Wu (2008),Fletcher et al. (2009),Chalup and Mitschele (2008); predicting price change in financial market by use of Kalman Filtering. Related work can be found in Fletcher (2007), Julier and Uhlmann (1997),Evensen (2003),Bolland and Connor (1996),Bolland and Connor (1997)

# CHAPTER 3

# MATHEMATICAL DESCRIPTION AND STATISTICAL PROPERTIES OF LOBS

In this chapter, we give precise mathematical descriptions of LOBs' trading and exchanging principles. In addition, some basic statistical properties of LOBs, which have been proposed by past research, are also studied and described. Those properties include size of orders, shape of order books, time of arrival of orders, placement of orders and so on. Study of those properties will help us get a clearer insight of how to capture the dynamics of LOBs. For example, knowing the distribution of arrival of orders will help us build more meaningful features in our prediction models.

## 3.1 Mathematical descriptions of LOBs

In this section, we give a relatively formal definition of LOBs and other related aspects of LOBs, most of these definitions come from Gould et al. (2013). A LOB can be represented as a three-dimensional vector by the following definition:

**Definition 3.1.1** *Gould et al. (2013) An order* $x = (p_x, \omega_x, t - x)$ *submitted at time* $t_x$ *which price* $p_x$ *and size* $\omega_x > 0$*(respectively,* $\omega_x < 0$*) is a commitment to sell (respectively, buy) up to* $|\omega_x|$ *units of the traded asset at a price no less than (respectively, no greater than)* $P_x$*.*

For a given LOB, the units of order size and price are defined as follows:

**Definition 3.1.2** *Gould et al. (2013) The lot size of a LOB is the smallest amount of the asset that can be traded within it. All orders must arrive with a size* $\omega \in \{\pm k\sigma | k = 1, 2, ...\}$

**Definition 3.1.3** *Gould et al. (2013) The tick size* $\pi$ *of a LOB is the smallest permissible price interval between different orders within it. All orders must arrive with a price that is specified to the accuracy of* $\pi$

For example, if $\pi = 0.01$, then the largest permissible order price that is strictly less than \$1 is \$ 0.99, and all orders must be submitted at a price with exactly two decimal places.

**Definition 3.1.4** *Gould et al. (2013) The lot size $\sigma$ and tick size $\pi$ of a LOB are collectively called its resolution parameters.*

**Definition 3.1.5** *Gould et al. (2013) When a buy (respectively, sell) order $x$ is submitted, a LOB's trade-matching algorithm checks whether it is possible to match $x$ to some other previously submitted sell(respectively, buy) order. If so, the matching occurs immediately. If not, $x$ becomes active*

Active orders in a market make up a LOB:

**Definition 3.1.6** *Gould et al. (2013) a LOB $\mathcal{L}(t)$ is the set of all active orders in a market at time $t$.*

a LOB can be treated as a set of queues, each of which contains active bid or ask orders at a specified price.

**Definition 3.1.7** *Gould et al. (2013) The bid-side depth available at price $p$ and at time $t$ is:*

$$n^b(p,t) := \sum_{\{x \in \mathcal{B}(t) | p_x = p\}} \omega_x$$

*The ask-side depth available at price $p$ and at time $t$, denoted $n^a(p,t)$, is defined similarly using $\mathcal{A}(t)$*

The depth available is often demonstrated as multiples of the lot size. Since $\omega_x < 0$ for bid orders and $\omega_x > 0$ for ask orders, it implies that $n^b(p,t) \leq 0$ and $n^a(p,t) \geq 0$ for all prices $p$.

**Definition 3.1.8** *Gould et al. (2013) The bid-side depth profile at time $t$ is the set of all ordered pairs($p$, $n^b(p,t)$). The ask-side depth profile at time $t$ is the set of all ordered pairs($p$, $n^a(p,t)$).*

**Definition 3.1.9** *Gould et al. (2013) The mean bid-side depth available at price $p$ between times $t_1$ and $t_2$ is*

$$\bar{n}^b(p,t_1,t_2) := \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} n^b(p,t)dt$$

*The mean ask-side depth available at price $p$ between times $t_1$ and $t_2$, denoted as $\bar{n}^a(p,t_1,t_2)$, is defined similarly using the ask-side depth available.*

In the following, we define the terms of *bid price, ask price, mid price, and bid-ask spread*

**Definition 3.1.10** *Gould et al. (2013) The bid price at time t is the highest stated price among active buy orders at time t,*

$$b(t) := \max_{x \in \mathcal{B}(t)} p_x$$

*The ask price at time t is the lowest stated price among active sell orders at time t,*

$$a(t) := \min_{x \in \mathcal{A}(t)} p_x$$

**Definition 3.1.11** *Gould et al. (2013) The bid-ask spread at time t is* $s(t) := a(t) - b(t)$

**Definition 3.1.12** *Gould et al. (2013) The mid price at time t is* $m(t) := [a(t) + b(t)]/2$

More clearly, in a LOB, $b(t)$ is the highest price at which it is immediately possible to sell at least the lot size of the traded asset at time $t$, and $a(t)$ is the lowest price at which it is immediately possible to buy at least the lot size of the traded asset at time $t$. Considering prices relative to $b(t)$ and $a(t)$ is helpful in some cases.

**Definition 3.1.13** *Gould et al. (2013) For a given price p, the bid-relative price is* $\delta^b(p) := b(t) - p$ *and the ask-relative price is* $\delta^a(p) := p - a(t)$

Note that here is a difference in signs between the definition of $\delta$ for the two sides: $\delta^b(p)$ defines how much smaller that $p$ is than $b(t)$ and $\delta^a(p)$ illustrates how much larger that $p$ is than $a(t)$. It is useful that we compare the properties of orders on both the bid side and the ask side of a LOB.

**Definition 3.1.14** *Gould et al. (2013) For a given order* $x = (p_x, \omega_x, t_x)$*, the relative price of the order is :*

$$\delta^x := \begin{cases} \delta^b(p_x) \; if \; the \; order \; is \; a \; buy \; order, \\ \delta^a(p_x) \; if \; the \; order \; is \; a \; sell \; order, \end{cases}$$

**Definition 3.1.15** *Gould et al. (2013) The bid-side depth available at relative price p and at time t is:*

$$N^b(p,t) = \sum_{\{x \in \mathcal{B}(t) | \delta^x = p\}}$$

*The ask-side depth available at relative price p and at time t, denoted* $N^a(p,t)$*, is defined similarly using* $\mathcal{A}(t)$

**Definition 3.1.16** *Gould et al. (2013) The bid-side relative depth profile at time t is the set of all ordered pairs*$(p, N^b(p,t))$*. The ask-side relative depth profile at time t is the set of all ordered pairs*$(p, N^a(p,t))$*.*

**Definition 3.1.17** *Gould et al. (2013) The mean bid-side depth available at relative price p between times* $t_1$ *and* $t_2$ *is:*

$$\bar{N}^b(p, t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} N^b(p, t)$$

*The mean ask-side depth available at relative price p between times* $t_1$ *and* $t_2$ *denoted* $\bar{N}^a(p, t_1, t_2)$ *is defined similarly using the ask-side relative depth available.*

**Definition 3.1.18** *Gould et al. (2013) The mean bid-side relative depth profile between times* $t_1$ *and* $t_2$ *is the set of all ordered pairs*$(p, \bar{N}^b(p, t_1, t_2))$*. The mean ask-side relative depth profile between times* $t_1$ *and* $t_2$ *is the set of all ordered pairs* $(p, \bar{N}^a(p, t_1, t_2))$

while most traders use the relative depth profile to measure the phenomenon of LOBs, some research has claimed that the order arrival rates relies on relative prices rather than actual prices(Biais et al. (1995), Bouchaud et al. (2002), Potters and Bouchaud (2003), Zovko et al. (2002)), relative depth profiles do not contain information about the absolute prices at which trades occur. Moreover, relative depth profiles provide little information about the bid-ask spread and mid price. Therefore, it is better that we combine the relative depth profiles, bid price($b(t)$) and ask price($a(t)$) together when we consider the problem of LOBs. A completed view of the evolution of a LOB can be obtained if we consider all the information simultaneously.

## 3.2 Properties of LOBs

During the last two decades, the financial market has become more and more computerized. Therefore, it is easier for us to access extensive data on order books. To our knowledge, Biais et al. (1995) is the pioneer to study the computerized data flows of Paris Bourse. After that, a lot of related papers provide more empirical findings, statistical properties, and modeling views.(See, e.g.Gopikrishnan et al. (2000), Challet and Stinchcombe (2001), Maslov and Mills (2001), Bouchaud et al. (2002), Potters and Bouchaud (2003)). In this section, we give a brief introduction of some

basic empirical study results. Those fundamental statistical properties can be found through observing the real-time data. Some results of observations, such as time of arrival of orders, placement of orders, size of orders, shape of orders books and etc, are essential for calibrating the models of order flows and capturing the dynamics of order books.

### 3.2.1 Time of arrivals of orders

We compute the cumulative distribution for inter-arrival times of market orders for four stocks: Amazon, Google, Intel and Microsoft. The results are plotted in figure 3.1. From the figure, it is obvious that the log-normal and exponential distribution are not good fits. The weibull distribution was suggested in Ivanov et al. (2004).From our results, we can see that the weibull distribution(red line in our figure) is a relatively good fit to the original data. However, in our case, the Gamma distribution is the best fit for each stock.

In many past pieces of literature, the non-Poisson arrival times models have been introduced to deal with the "irregular" financial data. For example, Engle and Russell (1997) and Engle (2000) have proposed autoregressive condition intensity models, which are beneficial to capture the processes of orders' submission. Another research area that deals with the non-exponential arrival times relies on the branches of stochastic processes(see, e.g. Clark (1973);Silva and Yakovenko (2007), Huth and Abergel (2012)).

### 3.2.2 Volume of orders

Some past research works found that the distribution of order book sizes is difficult to measure. A widely accepted rule is power-law distribution.Gopikrishnan et al. (2000) and Maslov and Mills (2001) find that the market orders follow a power law which decays with an exponent $1+\mu \approx 2.3 \rightarrow 2.7$, and limit orders follow a power law which decays with an exponent $1 + \mu \approx 2.0$. Challet and Stinchcombe (2001) pay more attention to a clustering effect, that is, "round" size in numbers of shares of orders is often observed. Moreover, clusters often occur around 100's and 1000's. Until now, those research models have not reached a consensus. It makes sense that the distribution will vary greatly depending on product and market changes

In figure 3.2, the distributions of the number of orders in 5 minutes of four stocks are plotted. We use their mean value to normalize the data. The parameter that we used in power law is negative 2.1 and the parameter that we used in exponential distribution is $1 + \mu \approx 2.7$, which we

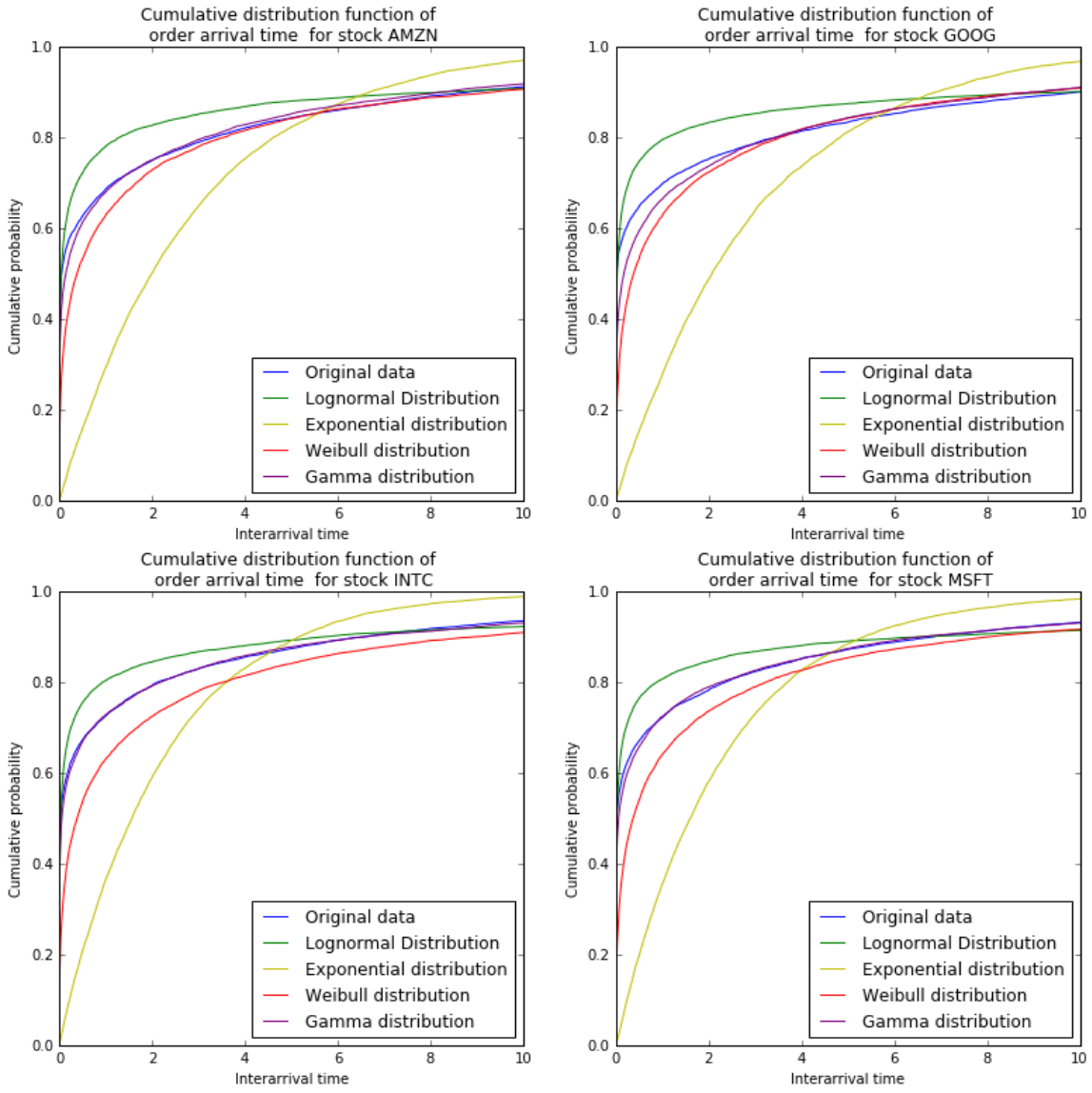


Figure 3.1: Cumulative distribution of inter-arrival time for stock: AMZN, GOOG, INTC and MSFT. In each panel, four distribution, Lognormal, Exponential Weibull and Gamma, are compared with the original dataset. X-axis represents the inter-arrival time of market orders and Y-axis shows the cumulative probability

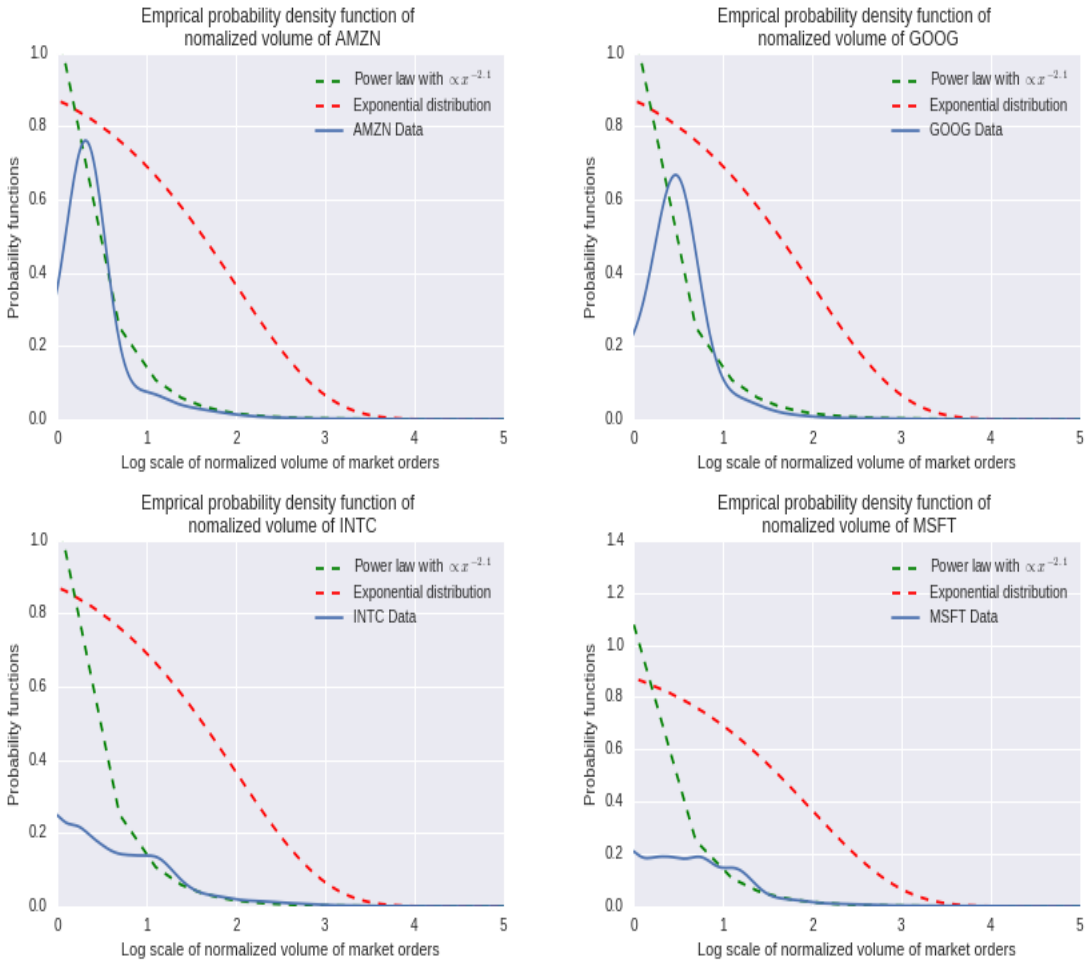


Figure 3.2: Distribution of the number of orders in 5 minutes of four stocks: AMZN, GOOG, INTC and MSFT. In each panel, power law with parameter negative 2.1 and exponential distribution are compared with the original dataset. X-axis represents the log scale of normalized volume of market orders and Y-axis shows the probability densities of different distributions

have mentioned above. We can see from the figure, in general, the exponential distribution is better than the power function for the fitting of empirical data. Meanwhile, for AMZN and Google, this effect is more significant. For INTC and MSFT, there is a large deviation in the initial part.

### 3.2.3 Placement of orders

On the French stocks market, Bouchaud et al. (2002) find that the placement of order books follows a broad power-law properties. After one year, Potters and Bouchaud (2003) claimed the same property on the US stocks market. The parameters of the power law are relatively stable: on the Paris Bourse, it is $1 + \mu \approx 1.6$, on the New York Stock Exchange, it is $1 + \mu \approx 2.0$ and $1 + \mu \approx 2.5$ on the London Stock Exchange. Mike and Farmer (2008) use a Student distribution with 1.3 degrees of freedom to fit the empirical distribution for LOBs.

In figure 3.3 we plot the probability distribution of placement of arriving limit orders. From this figure, we can find that the maximum probability of placement is around 0 price difference, which means the new order is more likely to place at the best price level. We also find heavy tails in the probability distribution, which means that there exists some significant price difference between the new order price and the best order price.

### 3.2.4 Intraday seasonality

The financial market always changes a lot during one trading day. Biais et al. (1995) study the order books in Paris Bourse and observe that the numbers of order book in a given period $\Delta T$ vary a lot during the day. Figure 3.4 and figure 3.5 describe the number of limit and market orders coming in a 5-minutes time intervals. From the chart, we observe a U shape of the numbers during the whole day. The dashed line is an ordinary quadratic least square fit of the data). The U shape tells us that the market is active at the beginning and the end of the day, and calm down around the midday. Potters and Bouchaud (2003) find that the average number of the market orders and limit orders in a $\Delta T$ period are highly correlated. In their papers, they use millisecond data sample and we use nano-second data, the result becomes similar, both market order and limit order figures demonstrate a U shape.

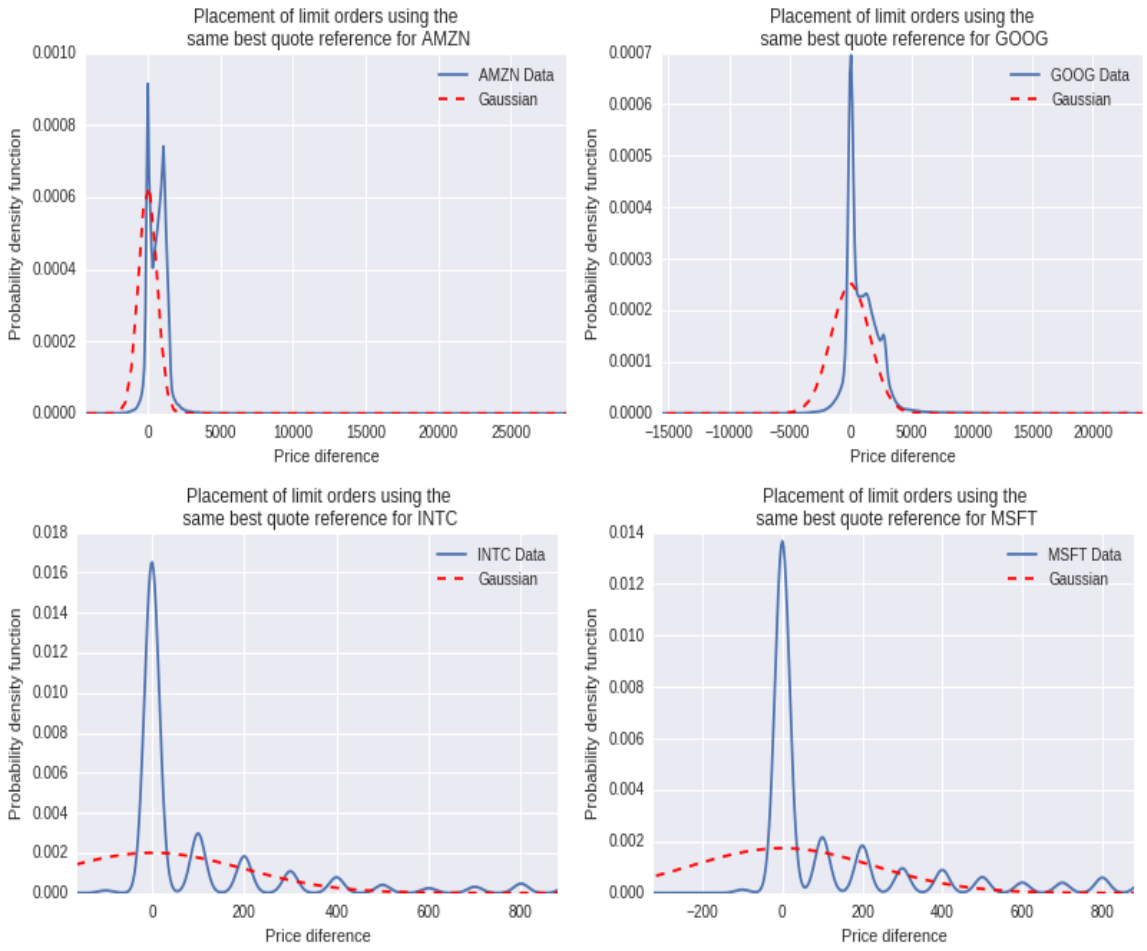


Figure 3.3: Density function of placement of limit orders using the same best quote. Four stocks that we use: AMZN, GOOG, INTC and MSFT. In each panel, Gaussian distribution is compared with the original dataset. X-axis represents the price difference between the price of arrival order and the best price before the arrival of this order. Y-axis shows the probability density function of the order book placement

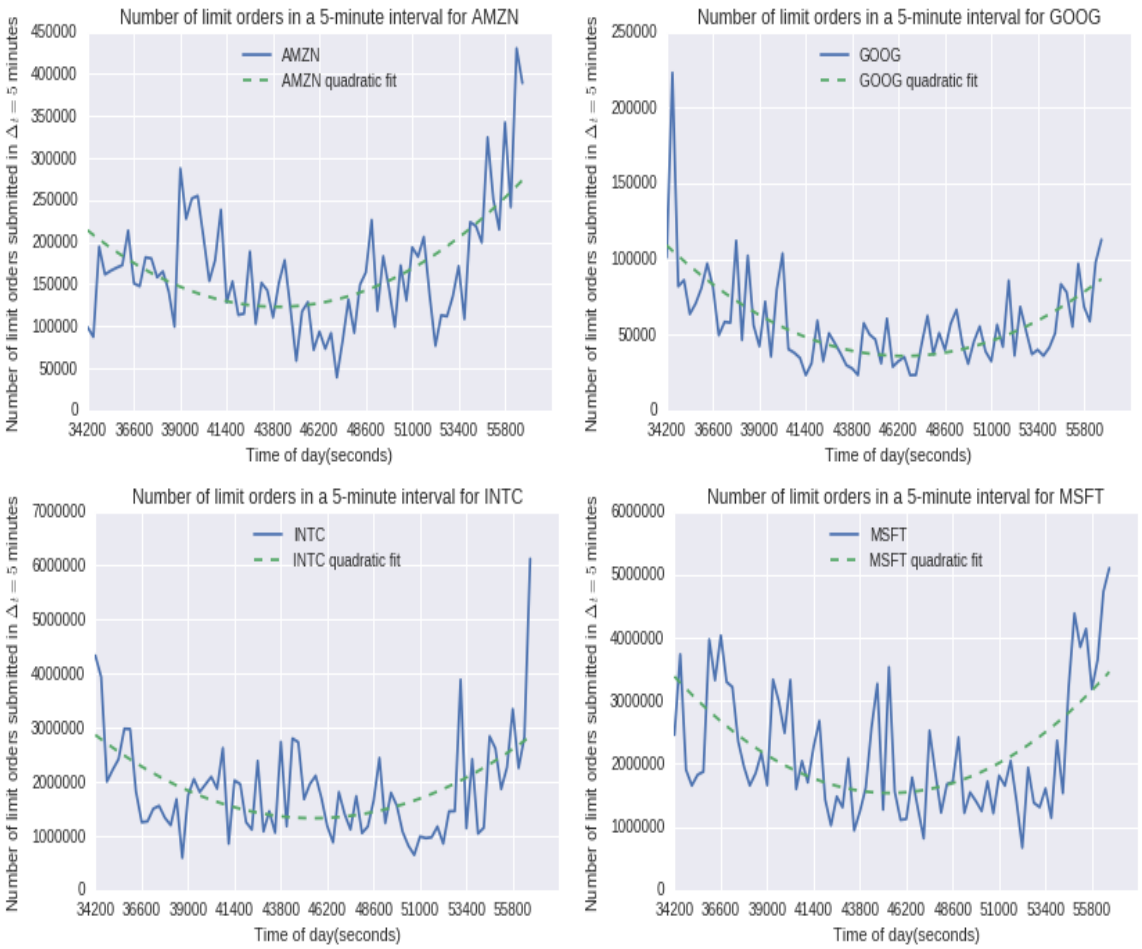


Figure 3.4: Normalized average number of limit orders in a 5-minute time interval. Four stocks that we use: AMZN, GOOG, INTC and MSFT. In each panel, least square quadratic function is used to fit the original dataset. X-axis represents the time of day in seconds. Y-axis shows the number of limit order submitted in a 5-minute time interval

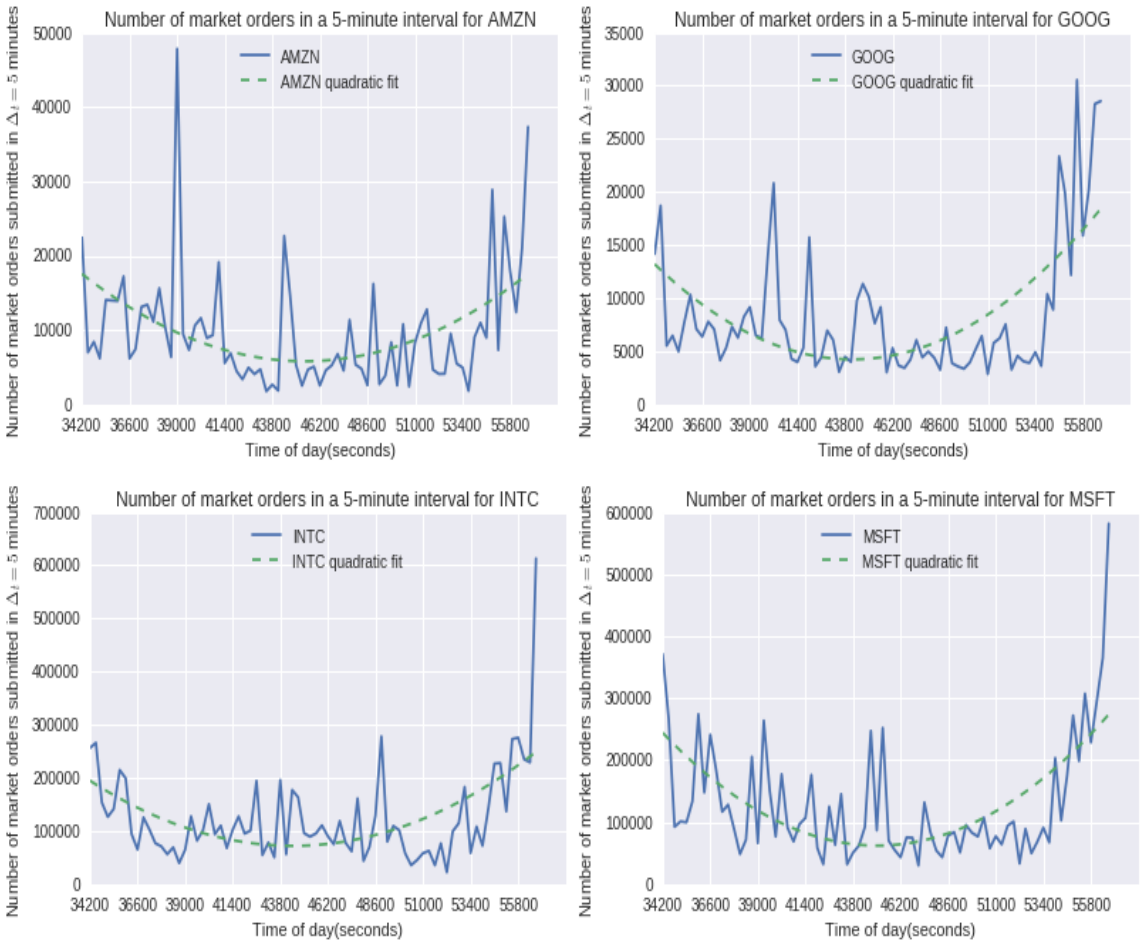


Figure 3.5: Normalized average number of market orders in a 5-minute time interval. Four stocks that we use: AMZN, GOOG, INTC and MSFT. In each panel, least square quadratic function is used to fit the original dataset. X-axis represents the time of day in seconds. Y-axis shows the number of market order submitted in a 5-minute time interval

### 3.2.5 Average shape of the order book

Bouchaud et al. (2002) study the stocks in French capital market and find an extraordinary phenomenon that the maximum of average volume in an order book often deviates from the best price. In figure 3.6, we plot the average number of five stocks based on a different price level. Our results show that when price level increases, the quantity of order books tend to be large, but not always. We can see that on the ask side of INTC, the minimum quantity occurs at the maximum price level. This might be caused by different trading strategies and preference between the US and French financial market.


Figure 3.6: Average quantity of market order books. Five stocks that we use: AAPL, AMZN, GOOG, INTC and MSFT. X-axis represents the price level: positive axis is ask side and vice versa. Y-axis shows the average numbers of shares (normalized by mean)

# CHAPTER 4

# BASIC MACHINE LEARNING SCHEMES

In our research, we use ensemble methods to predict future price changes of limit order books. As comparisons, some basic machine learning tools, such as logistic regression, lasso , and ridge regression and support vector machine, are used. We call them "basic", not because they are simple, but because they have a relatively long history and have already been widely used before. In the following, we give a brief introduction to these models.

## 4.1 Logistic regression

Logistic regression is a statistical tool which is used to find the fitting model to characterize the relationship between the independent variables and the response variable. Unlike traditional linear regression, the response variable of logistic regression is logit transformation of the probability. More clearly, suppose that we have a binary dependent variable $Y$, and we try to define the condition probability $Pr(Y = 1|X = x)$, which is denoted as $p(x)$. An unbounded range transformation, which is called **logistic** (or **logit**) **transformation** $log\frac{p}{1-p}$, is introduced. The advantage of **logit transformation** is that it is unbounded in both positive and negative directions when $p(x)$ approaches 1 and 0 respectively. Formally, the formula of logistic regression model is as follows:

$$log\frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta \tag{4.1}$$

Solving the above formula for $p$ will give us:

$$p(x; b, \omega) = \frac{e^{\beta_0 + x\cdot\beta}}{1 + e^{\beta_0 + x\cdot\beta}} = \frac{1}{1 + e^{-(\beta_0 + x\cdot\beta)}} \tag{4.2}$$

To solve the binary classification problem, we can assume $Y = 1$ when $p >= 0.5$ and $Y = 0$ when $p < 0.5$. Logistic regression is a widely used tool in applied statistics and discrete data analysis. There are some reasons for this:

- It is a traditional and classical model with a relatively long history, which can be tracked back to Bliss (1934). It has a wide range of applications in a lot of areas, so there are many related materials and cases that one can refer to.

- It has "exponential family" distribution property, which states that the probability of a specific variable $v$ can be described linearly by $e^{\beta_0+\sum_{j=1}^{m} f_j(v)\beta_j}$. If we let the vector $v$ as binary, 0 or 1 and the functions $f_j$ all become identical, then we can obtain a logistic regression. Exponential families take a very important role in the statistical and physical research area, so it is beneficial if we turn a problem into logistic regression problem.

- Contingency table plays a critical role in classification problem since if we take a contingency table with two columns(independent variable $Y$) and many rows(dependent variables $x$), the contingency table becomes the classification problem to some extent. Logistic regression gives a linear interpolation for the log-odds, so it is very helpful in the classification problem.

Because the logistic regression deals with the probabilities, rather than just classes, we can use likelihood to fit the model.Assume that we have independent variable $Y$(denoted by $y_i$ for each data ) and dependent variables $x$(denoted by $x_i$ for each data), the probability is either $p$, if $y_i = 1$, or $1 - p$, if $y_i = 0$. The Likelihood is defined as follows:

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} \tag{4.3}$$

Take *log* transformation on both sides, the log-likelihood becomes:

$$\begin{aligned} l(\beta_0, \beta) &= \sum_{i=1}^{n} y_i log p(x_i) + (1 - y_i) log(1 - p(x_i)) \\ &= \sum_{i=1}^{n} log(1 - p(x)) + \sum_{i=1}^{n} y_i log \frac{p(x)}{(1 - p(x))} \\ &= \sum_{i=1}^{n} log(1 - p(x_i)) + \sum_{i=1}^{n} y_i(\beta_0 + x \cdot \beta) \\ &= \sum_{i=1}^{n} -log(1 + \exp^{\beta_0 + x_i \cdot \beta}) + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta) \end{aligned} \tag{4.4}$$

To find the maximum of the log-likelihood, we can take the first derivative of the log-likelihood function with respect to coefficient and set it equal to zero. The result is in the following:

$$\begin{aligned}\frac{\partial l}{\partial \beta_j} &= -\sum_{i=1}^{n} \frac{1}{1+\exp^{(\beta_0+x_i\beta)}} \exp^{(\beta_0+x_i\beta)} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=1}^{n} (y_i - p(x_i;\beta_0,\beta)) x_{ij}\end{aligned} \tag{4.5}$$

From en:likelihood$_d erivative, we can see that it is a transcendental equation which cannot transform to a polynomia$

## 4.2 Lasso regression and ridge regression

lasso and ridge regression add a different regulation parameter to the ordinary least square regression, so before we describe lasso and ridge regression, we first give a short introduction of ordinary least square regression(OLS). Assume the response variable is $Y$ and the dependent variable is $x$, the following model:

$$y_i = \beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + ... + x_{ik}\beta_k + e_i \tag{4.6}$$

where $\beta_i$ corresponds to the coefficient of $x_i$ and $e_i$ is the error term with E[$e_i$]=0
is called least square regression. To make the error term become zero, we take the expectation on both sides and let $E(y_i) = \hat{y}_i$, now equation 4.6 becomes:

$$\begin{aligned}E(y_i) = \hat{y}_i &= \beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + ... + x_{ik}\beta_k \\ \hat{y} &= \boldsymbol{X\beta}\end{aligned} \tag{4.7}$$

So the error term $\boldsymbol{e = Y - \hat{Y}}$ and we can define a loss function for the least square regression as $f = \boldsymbol{e'e}$. Our goal is to minimize this loss function. To realize this , we can first define the loss function according to $\beta, x$, and $y$.

$$\begin{aligned}f = \boldsymbol{e'e} &= \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= (y_i - \hat{y}_i)'(y_i - \hat{y}_i) \\ &= \boldsymbol{(y - x\beta)'(y - x\beta)} \\ &= \boldsymbol{y'y - \beta'X'y - y'X\beta + \beta'X'X\beta} \\ &= \boldsymbol{y'y - 2y'X\beta + \beta'X'X\beta}\end{aligned} \tag{4.8}$$

Now let the first derivative of loss function to the coefficient $\beta$ equal to zero, i.e. $\frac{\partial f}{\partial \beta} = 0$. Since $\boldsymbol{y'y}$ is constant with respect to $\beta$, $\frac{\boldsymbol{\partial y'y}}{\boldsymbol{\partial \beta}}$ is a null vector with the length equal to the number of data

samples. The derivative of the second term is:

$$\frac{\boldsymbol{\partial(-2y'X\beta)}}{\boldsymbol{\partial\beta}} = \boldsymbol{-2X'y} \tag{4.9}$$

For the third term:

$$\frac{\boldsymbol{\partial\beta'X'X\beta}}{\boldsymbol{\partial\beta}} = \boldsymbol{-2X'X\beta} \tag{4.10}$$

Now let the derivative of loss function equal to zero, we obtain that:

$$\frac{\boldsymbol{\partial f}}{\boldsymbol{\partial\beta}} = \boldsymbol{2X'X\beta - 2X'y} = 0 \tag{4.11}$$

So we have:

$$\begin{aligned} \frac{\boldsymbol{\partial f}}{\boldsymbol{\partial\beta}} = \boldsymbol{2X'X\beta - 2X'y} &= 0 \\ \boldsymbol{X'X\beta} &= \boldsymbol{X'y} \\ \boldsymbol{X'X\beta} &= \boldsymbol{X'y} \\ \boldsymbol{\hat{\beta}} &= \boldsymbol{(X'X)^{-1}X'y} \end{aligned} \tag{4.12}$$

The $\hat{\beta}$ is our fitted coefficients which make the loss function smallest. In fact, the expectation of $\hat{\beta}$ will equal to $\beta$, we give the proof as follows:

$$\begin{aligned} \boldsymbol{E(\beta)} &= \boldsymbol{E[(X'X)^{-1}X'y]} \\ &= \boldsymbol{(X'X)^{-1}X'E(y)} \\ &= \boldsymbol{(X'X)^{-1}X'[X\beta]} \\ &= \boldsymbol{\beta} \end{aligned} \tag{4.13}$$

For the least square regression, if the rank ($\boldsymbol{X}$) is equal to the number of independent variables, then the solution system only has one solution $\hat{\beta}$. But when the rank($\boldsymbol{X}$) is greater than the number of independent variables, the system will have infinitely many solutions, which make the solutions lack credibility. Furthermore, even rank($\boldsymbol{X}$) equals to the number of independent variables, when the sample size is close to the number of independent variables, least square regression can have very poor prediction results.

How can we overcome these problems? One way is to add some constraints to the least squares model, that is:

$$\begin{aligned} &\min_{\beta\in\mathbb{R}^p} ||y - X\beta||_2^2 \\ &\textit{subject to } \beta \in C \end{aligned} \tag{4.14}$$

for some $C \in \mathbb{R}^p$, where $p$ is the number of independent variables. Lasso regression and ridge regression are two famous examples that choose $C$ to $l_1$ norm region and $l_2$ norm region. More clearly:

Ridge regression:

$$\begin{aligned} &\min_{\beta \in \mathbb{R}^p} ||y - X\beta||_2^2 \\ &subject\ to\ ||\beta||_2 \leq t \end{aligned} \tag{4.15}$$

Lasso regression:

$$\begin{aligned} &\min_{\beta \in \mathbb{R}^p} ||y - X\beta||_2^2 \\ &subject\ to\ ||\beta||_1 \leq t \end{aligned} \tag{4.16}$$

We can rewrite equation 4.15 and 4.16 with the KKT conditions and duality as:

Ridge regression:

$$\min_{\beta \in \mathbb{R}^p} ||y - X\beta||_2^2 + \lambda||\beta||_2 \tag{4.17}$$

Lasso regression:

$$\min_{\beta \in \mathbb{R}^p} ||y - X\beta||_2^2 + \lambda||\beta||_1 \tag{4.18}$$

To our best knowledge, lasso regression was introduced by Tibshirani (1996) and Chen et al. (2001). Hoerl and Kennard (1970) were the first ones who propose ridge regression. From figure 4.1 and figure 4.2, we can see that the constraint sets for both ridge and lasso are convex. The $l_1$ norm causes sparsity of the estimated coefficients, that is, there are $\beta_j = 0$ for many j in the set $1, ..., p$, when the tuning parameter becomes large. However, this is not the case in ridge regression, which will always generate non-zero estimators, when the tuning parameter changes.

From the KKT conditions(equation 4.18 and equation 4.17), it is feasible to find the regression estimators as a function of $\lambda$, which is the tuning parameter. This is called *solution path or regularization path* of the dual problem. Figure 4.3 and figure 4.4 show the regulation path of an example, we can see that lasso estimators will become zero when the tuning parameter rises. However, the ridge estimators will only approach zero but never touch zero in most cases.

## 4.3 Support vector machine

As far as we know, Boser et al. (1992) first introduced the support vector machine(SVM) scheme, which is now widely used in bio-informatics, finance, and other areas. Generally speaking, SVM is

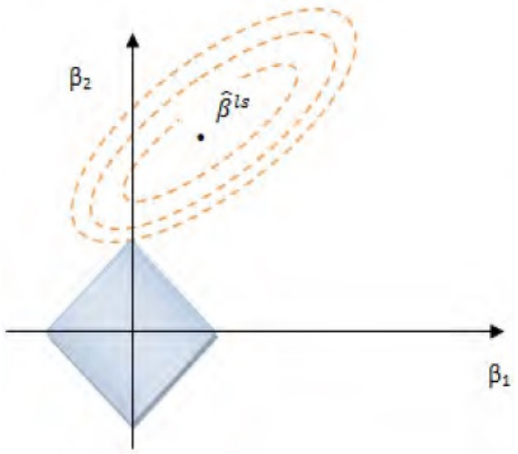


Figure 4.1: Illustration of Lasso constraints, redraw the chart in chapter 3 of Friedman et al. (2001)

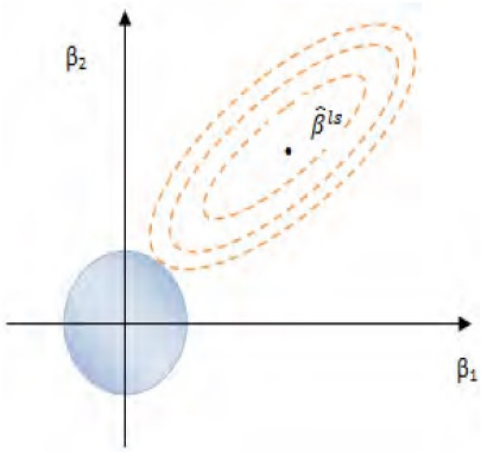


Figure 4.2: Illustration of Ridge constraints, redraw the chart in chapter 3 of Friedman et al. (2001)

used to solve a linear two classes classification problem. Suppose there are input variables $x$ and output $y$ and we assume that $y$ only have two values: 1 for positive examples and -1 for negative examples. The relationship between $x$ and $y$ is:

$$f(x) = \sum_{i=1}^{N} \omega^T x + b \tag{4.19}$$

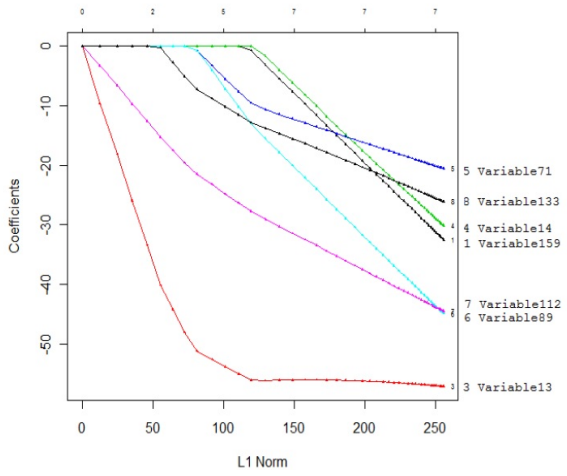


Figure 4.3: An example of the lasso regulation path with the increasing of the tuning parameter, each line represents the lasso solution , redraw the chart in chapter 3 of Friedman et al. (2001)

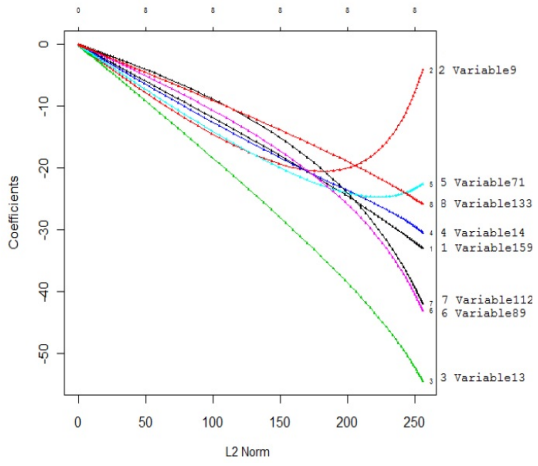


Figure 4.4: An example of the ridge regulation path with the increasing of the tuning parameter, each line represents the ridge solution , redraw the chart in chapter 3 of Friedman et al. (2001)

Where $\omega$ represents the *weight vector*, and $b$ is known as the bias. The *hyperplane* $\{x : f(x) = \omega^T x + b\}$ divide the space into two side: when $f(x) = \omega^T x + b > 0$, we treat the data in the positive side and vice versa. The boundary that split the space into positive and negative is called

the *decision boundary*. Figure 4.5 shows an example of decision boundary of SVM. The black line divide the data into two sets, the red data points are marked as positive and the blue ones are marked as negative. How to find the best decision boundary is the key problem in SVM learning process. To explain this, we need first introduce the definition of margin. When the decision boundary is defined, we denote the closet positive(negative) points to the hyperplane as $x_+(x_-)$. $\hat{\omega}$, the unit vector of $\omega$, is given by $\frac{\omega}{||\omega||}$ where $||\omega||$ is the norm of $\omega$. The margin of a decision boundary $f$ with respect to a dataset $\mathbb{D}$ can be defined as:

$$m_{\mathbb{D}}(f) = \frac{1}{2}\hat{\omega}^T(x_+ - x_-) \tag{4.20}$$

Also, we assume that the distance of $x_+$ and $x_-$ to the decision boundary $f$ is equal which means:

$$\begin{aligned} f(x_+) &= \omega^T x_+ + b = a \\ f(x_-) &= \omega^T x_- + b \end{aligned} \tag{4.21}$$

for some fixed number $a > 0$ and $x_+$, $x_-$ can also called support vectors. To solve the margin, we set $a = 1$ in equation 4.21, add the two formula together and divide by $||\omega||$, we can get:

$$m_{\mathbb{D}}(f) = \frac{1}{2}\hat{\omega}^T(x_+ - x_-) = \frac{1}{||\omega||} \tag{4.22}$$

The goal of support vector machine scheme is to maximize the margin $\frac{1}{||w||}$, which is same as minimizing $||w||^2$. So the support vector machine problem can be transformed to the following quadratic optimization problem:

$$\begin{aligned} \underset{\omega,b}{\text{minimize}} \quad & \frac{1}{2}||\omega||^2 \\ \text{subject to} \quad & y_i(\omega^T x_i + b) >= 1,\ i = 1, \ldots, n. \end{aligned} \tag{4.23}$$

In the ideal case, the above equation can perfectly separate the data if they are linearly separable. However, in practice, data is often not linearly separable, to allow errors we can replace the right side of equation 4.23 with the following:

$$y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, \ldots, n. \tag{4.24}$$

where $\xi_i \geq 0$ are called *slack variables*. Under this condition, some misclassifications are allowed if the example lies in the margin and we call this kind of error a margin error. To penalize the margin errors and misclassification, equation 4.23 can be transformed to:

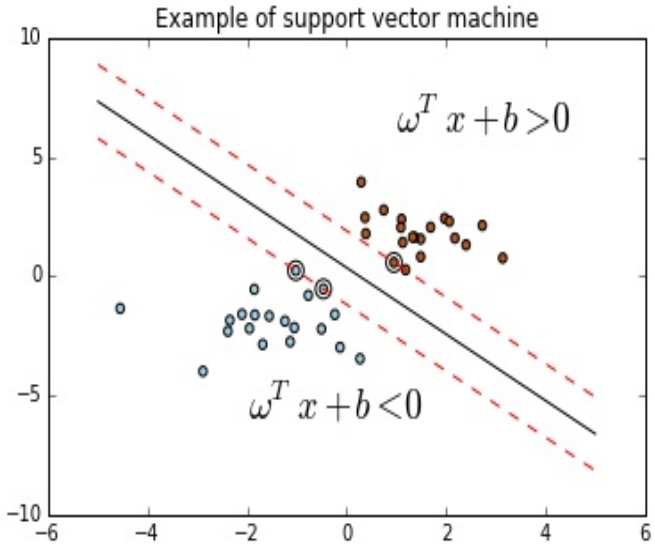


Figure 4.5: Example of a linear support vector machine classifier. The black line is the decision boundary and the circled data are the support vectors.

$$\begin{aligned} \underset{\omega,b}{\text{minimize}} \quad & \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\omega^T x_i + b) \geq 1 - \xi_i,\ i = 1, \ldots, n, \xi_i \geq 0 \end{aligned} \tag{4.25}$$

This formula is called *soft margin* SVM, which was proposed by Cortes and Vapnik (1995). For solving this formula, we can use a Lagrange multipliers method to obtain the dual form which utilize the inner product.

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^T x_j \\ \text{subject to} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0,\ 0 \leq \alpha_i \leq C \end{aligned} \tag{4.26}$$

Here we use the formula $\omega = \sum_{i=1}^{n} y_i \alpha_i x_i$.

As we mentioned before, in practice, the data is often non-linearly separable. To handle this problem, we often map our data from the input space $\mathcal{X}$ to a feature space $\mathcal{F}$. In the space $\mathcal{F}$, the data may be close to linear separable and the discriminant function in this space is: $f(x) =$

$\omega^T\Phi(x)+b$. Suppose the weight vector can be represented by a linear combination of the training samples, i.e. $\omega=\sum_{i=1}^{n}\alpha_i x_i$ then:

$$f(x)=\sum_{i=1}^{n}\alpha_i x_i^T x+b \tag{4.27}$$

The above formula becomes the following form in the space $\mathcal{F}$

$$f(x)=\sum_{i=1}^{n}\alpha_i\Phi(x_i)^T\Phi(x)+b \tag{4.28}$$

If we define a kernel function $k(x,x')$ as

$$k(x,x')=\Phi(x)^T\Phi(x') \tag{4.29}$$

then the equation 4.28 can be written in terms of the kernel function:

$$f(x)=\sum_{i=1}^{n}\alpha_i k(x,x_i)+b \tag{4.30}$$

The following is an example: let $\Phi(x)=(x_1^2,\sqrt{2}x_1x_2,x_2^2)^T$, if we use the kernel function: $k(x,x')=(x^Tx)^2$, then:

$$\begin{aligned}\Phi(x)^T\Phi(z)&=(x_1^2,\sqrt{2}x_1x_2,x_2^2)^T(z_1,\sqrt{2}z_1z_2,z_2^2)\\&=x_1^2z_1^2+2x_1x_2z_1z_2+x_2^2z_2^2\\&=(x^Tz)^2\end{aligned}$$

We can see that the mapping function $\Phi$ maps the data $x$ in the low dimensional space to the data $\Phi(x)$ in the high dimensional space. The inner product of the high dimensional data is equal to the mapping results of the inner product of the low dimensional data under the kernel function.The following are some widely used types of the kernel function.

Linear kernel: mapping function $\Phi$ is identity, i.e. no mapping

$$k(x,z)=x^Tz \tag{4.31}$$

Polynomial kernel(with degree d):

$$k(x,z)=(x^Tz)^d\ or\ (1+x^Tz)^d \tag{4.32}$$

Radial Basis function kernel: mapping the variable x to an infinite dimensional feature space.

$$k(x,z)=exp(-r||x-z||^2) \tag{4.33}$$

There has

## 4.4 Decision tree

Ensemble models combine some weaker classifiers to become a strong classifier. Decision tree is a commonly used weaker classifier. We also use decision tree result as the reference to our ensemble models. In the following, we give a brief introduction of decision tree learning scheme. Decision tree learning is a method to the approximate discrete target function. The learned function can be treated as decision processes which apply the if-then rules. Decision trees classify examples by sorting them from the top root node to the bottom leaf node. Each node in the tree represents a test of one characteristic of the example and each branch from that node specifies one of the possible results for this characteristic

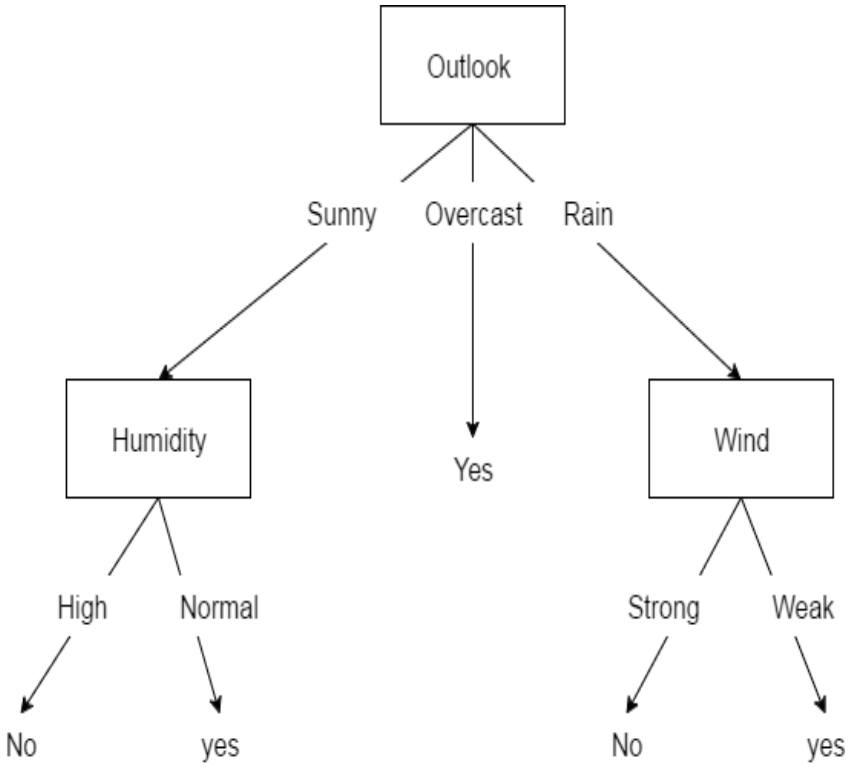


Figure 4.6: Example of decision tree, determine if a person will go out for playing table tennis or not.Nodes are attributes of the weather in a given day and the output in the last layer is yes or not.

Figure 4.6 gives us an illustration of a typical decision tree learning process. Given an instance (Outlook =Sunny, Temperature =Hot, Humidity=High and Wind =Weak), it will go down to he

leftmost branch of this decision tree and will be classified negative, which will not go to tennis on this day.

To improve the efficiency of the learning process, we would like to select the characteristic that is most helpful to classify the samples. We introduce a definition called "*entropy*" which measures the degree of impurity in a given sample set. Suppose there is a data sample $S$, including positive and negative instances, the entropy of $S$ correspond to this binomial classification is :

$$Entropy(S) = -p_+ log_2 p_+ - p_- log 2 p_- \tag{4.34}$$

where the $p_+$ is the ratio of positive instances in sample $S$ and $p_-$ is the ratio of negative instances in $S$.For example, is a data set $S$ has 14 instances, of which nine are positive and five are negative(we denote them as $[9+, 5-]$). Then the entropy of $S$ is:

$$\begin{aligned} Entropy([9+, 5-]) &= -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} \\ &= 0.940 \end{aligned} \tag{4.35}$$

More generally, if the target value takes more than two different classes, then the entropy of $S$ can be written as:

$$Entropy(S) = \sum_{i=1} C - p_i log_2 p_i \tag{4.36}$$

where $p_i$ represents the ratio of the number of samples with class i to the whole sample size and $C$ is the number of different classes. We now introduce another new definition that judge the effectiveness of a feature to classify the data sample and call it *information gain*. More formally, information gain regard to a sample data $S$ is:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)}^{\frac{|S_v|}{|S|}} Entropy(S_v) \tag{4.37}$$

where $Values(A)$ is the collection of all possible values for feature $A$ and $S_v$ is the subset of $S$ where feature A has value $v(S_v = s \in S | A(s) = v)$ Now let's see an example of information gain. We still assume a collection $S$ including 14 instances [9+, 5-] and the instances that have Wind=Weak contain 6 positive and 2 negative examples, the remainder has Wind = Strong. The information

gain of sorting these 14 samples by the feature Wind can be obtained as :

$$\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &= [6+, 2-] \\
S_{strong} &= [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in Values(A) \in \{Weak, Strong\}} \frac{S_v}{S} Entropy(S_v) \\
&= Entropy(S) - (\frac{8}{14}) Entropy(S_{Weak}) - (\frac{6}{14}) Entropy(S_{strong}) \\
&= 0.940 - (\frac{8}{14}) 0.811 - \frac{6}{14} 1.00 \\
&= 0.048
\end{aligned} \tag{4.38}$$

There ar several reasons that we choose the regression tree as our base classifier. First, it conducts the feature selection automatically. During each step, the regression tree selects one variable with the highest information gain. When the whole process finished, the model actually gives us a relatively sorted variable list based on the importance of the feature. Second, it can capture the non-linear properties of the features. Friedman et al. (2000) presents that the boosting model based on regression tree is an approximation to additive regression models which can deal with the non-linearities of predictors. Friedman et al. (2000) also conduct some experimental tests and show that boosting method with decision tree has a good approximation of some non-linear functions.

# CHAPTER 5

# ENSEMBLE METHODS FOR PREDICTION

AdaBoost and random forest learning algorithms are two machine learning models that we use to improve prediction performance of price change for limit order books. These models are two widely used ensemble methods. In this chapter, we give a brief description of ensemble methods, especially AdaBoost and random forest algorithms.

## 5.1 An introduction to ensemble methods

Ensemble methods are designed for improving the performance of a statistical fitting model. It combines the results of some fitting models, if those models are relatively independent, the combined result will decrease the predicting errors. To be formal, let's consider a real-valued function

$$g : \mathbb{R}^d \to \mathbb{R} \tag{5.1}$$

that is based on data sample $(X_1, Y_1), ..., (X_n, Y_n)$, where $X$ is in a $d$-dimensional real value space and $Y$ is the response variable based on $X$. We define a projecting process called "*base procedure*" which will generate a real-value function based on some $d$-dimensional real input data. To use ensemble methods, we usually do the base procedure several times by changing the input data sample. For example , we can give different weights to dependent variables in each procedure to get different predicting results, $g_1(\hat{\omega}_1^T X), g_2(\hat{\omega}_2^T X), g_3(\hat{\omega}_3^T X), ..., g_n(\hat{\omega}_n^T X)$, where $\hat{\omega}_i$ is the weights in the model in procedure $i$ and $n$ is the number of times that we run the model. Therefore, the results of the ensemble methods $\hat{g}_{ens}X$ is the linear combinations of the generated individual function $g_k(\hat{\omega}_k^T X)$

$$\hat{g}_{ens}(X) = \sum_{k=1}^{n} c_k \hat{g}_k(\omega_k^T X) \tag{5.2}$$

where $C_k$ is the coefficients of a linear combination of the individual function $\hat{g}_k$.The $c_k$ is usually identical, which equal to $\frac{1}{n}$ for the linear model. For other models, such as boosting, $\sum_{k-1}^{n} c_k$ will increase when n gets larger.

Why the ensemble models will improve the predictive performance of an individual model? The reasons can be concluded in the following: 1) Re-sampling the data, such as bagging procedure, can be treated as a variance reduction method.Bühlmann (2012) 2) Boosting methods can reduce both bias and variance. Since boosting is a weighted average of some weaker classifiers, so it usually has lower variance. Besides, boosting method pays more attention to the sample of poor predictions, so it often reduces the bias too.

## 5.2 Bagging

Bagging is a very important ensemble method. Since we will both use bagging method during the process of building AdaBoost and random forest scheme, so in this section, we give a short introduction of this technique. Suppose we have pairs $(X_i, Y_i), i = 1, ..., n$ , where $X_i$ is a d-dimensional real value vector and $Y_i$ is a real value defined as a response variable. If $Y_i$ only takes discrete integers, then it is a classification problem, otherwise, it is a regression problem. Now define a function estimator, which generates result via a given base process

$$\hat{g}(\cdot) = h_n((X_1, Y_1), ..., (X_n, Y_n))(\cdot) : \mathcal{R}^d \to \mathcal{R} \tag{5.3}$$

where $h_n$ is a function that estimates a function according to the given data. The algorithm of bagging is in the following:

**Algorithm 5.1:** Bagging algorithm, Breiman (1996)

1. Randomly draw n times with replacement from the data $(X_1, Y_1), ..., (X_n, Y_n)$ and define a bootstrap sample data as $(X_1^*, Y_1^*), ..., (X_n^*, Y_n^*)$;
2. Derive the bootstrapped estimator $\hat{g}^*(\cdot)$, $\hat{g}^*(\cdot) = h_n((X_1^*, Y_1^*), ..., (X_n^*, Y_n^*))(\cdot)$;
3. Repeat step 1 and step 2 $M$ times, where $M$ is often chosen as 50 or 100, yielding $\hat{g}^{*k}(\cdot)(k = 1, ..., M)$. The bagged estimator is $\hat{g}_{Bag}(\cdot) = M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}(\cdot)$

## 5.3 Bootstrap

Bootstrap is another important scheme that is used in random forest method. It was first introduced by Efron (1979). This algorithm draws plenty of independent bootstrap samples, assesses the corresponding bootstrap replications, and evaluates the standard error of parameter $\hat{\theta}$ by the empirical standard error, denoted by $\hat{se}_B$, where $B$ is the number of bootstrap samples that we

used. The pseudocode of algorithm is described in the following:

**Algorithm 5.2:** Bootstrap algorithm, Breiman (1996)

1 Choose $B$ independent bootstrap samples $x_1^*, ..., x_B^*$, every sample contains $n$ data values drawing with replacement from $x$.

2 Estimates the bootstrap replication corresponding to each bootstrap sample

$$\hat{\theta}^*(b) = s(x_b^*), b = 1, ..., B$$

3 Evaluate the standard error $se_F(\hat{\theta})$ by the sample standard error of the $B$ replicates

$$\hat{se}_B = [\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^*(b) - \hat{\theta}^*(.))^2]$$

with

$$\hat{\theta}^*(.) = B^{-1}\sum_{b=1}^{B}\hat{\theta}^*(b)$$

When the sample size $B$ goes to infinity, the limit of $\hat{se}_B$ becomes an ideal bootstrap estimate.

## 5.4 AdaBoost learning scheme

This section we study the procedure and properties of an important learning scheme, AdaBoost, which is deemed as the best off the shelf classifier in the word(Friedman et al. (2000)). The purpose of boosting method is to combine many weaker models together to generate a strong "committee". Freund and Schapire (1995) first introduced the AdaBoost scheme.Suppose we have a response variable $Y$, which can only take two values, 1 and -1. We also assume a vector of independent variables $X$ and the predicted result of a classifier $G(x)$ can also only take one of the two values $1, -1$. The error rate of the predict value $G(x)$ and the true value is:

$$\bar{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)) \tag{5.4}$$

Now we assume that $G_m(x)$ is a sequence of weak classifiers, $m = 1, 2, ..., M$, the strong classifier $G(x)$ is obtained through a weighted majority vote of the results generated by the weak classifiers $G_m(x)$:

$$G(x) = sign(\sum_{m=1}^{M}\alpha_m G_m(x)) \tag{5.5}$$

where $\alpha_1, \alpha_2, ..., \alpha_M$ are generated by the AdaBoost algorithm. In each process, weights will be updated and those data that are misclassified will get more attention. Algorithm 5.3 shows the main procedure of AdaBoost. At beginning, all the initial weights are set to be $\frac{1}{N}$, which means that they are all equal. For each successive step, the weights are individually changed based on the classification result on the last step. For example, at step k, those data that are misclassified by the classifier $G_{k-1}$ will increase their weights, meanwhile, those data that are correctly classified will decrease their weights.The factor that we update the weights is $exp(\alpha_m)$ in the algorithm, which is an exponential function of the error rate in the last step.

**Algorithm 5.3:** AdaBoost algorithm, Friedman et al. (2001)

1 Initialize the observation weights $\omega_i$=1/$N$, $i$=1, 2, ..., $N$;
2 **for** *m=1 to M* **do**
3 Fit a classifier $G_m(x)$ to the training data using weights $\omega_i$;
4 Compute ;

$$err_m = \frac{\sum_{i=1}^{N} \omega_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} \omega_i}$$

5 Compute $\alpha_m = log((1 - err_m)/err_m)$;
6 Set $\omega_i \leftarrow \omega_i \cdot exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, ..., N$ ;
7 Output $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$ ;

In general, AdaBoost is a kind of forward stag-wise additive model that uses the exponential loss function. Algorithm 5.4 shows the procedure of forward stag-wise additive model.

**Algorithm 5.4:** Forward Stag-wise Additive Modeling, Friedman et al. (2001)

1 Initialize $f_0(x) = 0$;
2 **for** *m=1 to M* **do**
a) Compute ;
$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$ ;
b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

The loss function of AdaBoost is :

$$L(y, f(x)) = exp(-yf(x)) \tag{5.6}$$

Substitute into algorithm 5.4, we can get

$$(\beta_m, \gamma_m) = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} exp(-y_i(f_{m-1}(x_i) + \beta G(x_i)) \tag{5.7}$$

This can be transformed to:

$$(\beta_m, \gamma_m) = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} \omega_i^{(m)} exp(-\beta y_i G(x_i)) \tag{5.8}$$

where $\omega_i^{(m)} = exp(-y_i f_{m-1}(x_i))$. To solve equation 5.8, first, assume $\beta > 0$, the solution of $G_m(x)$ in equation 5.8 is:

$$G_m = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} \omega_i^{(m)} I(y_i \neq G(x_i)) \tag{5.9}$$

To prove this, from equation 5.8, we can easily get:

$$\sum_{i=1}^{N} \omega_i^{(m)} exp(-\beta y_i G(x_i)) = e^{-\beta} \sum_{y_i = G(x_i)} \omega_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} \omega_i^{(m)} \tag{5.10}$$

which can be transformed into the following form if we add and minus term $e^{-\beta} \sum_{y_i \neq G(x_i)} \omega_i^{(m)}$ simultaneously.

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} \omega_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} \omega_i^{(m)} \tag{5.11}$$

Now put $G_m$ into equation 5.8 and solve parameter $\beta$ by taking first derivative, we can get:

$$\beta_m = \frac{1}{2} log \frac{1 - err_m}{err_m} \tag{5.12}$$

where $err_m$ is the weighted error rate that is minimized:

$$err_m = \frac{\sum_{i=1}^{N} \omega_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} \omega_i^{(m)}} \tag{5.13}$$

Now the function $f_m(x)$ is updated:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x) \tag{5.14}$$

and the weight in the next step is:

$$\omega^{(m+1)} = \omega_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)} \tag{5.15}$$

since $\omega_i^{(m)} = exp(-y_i f_{m-1}(x_i))$

We know that $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, so equation 5.15 can be written as :

$$\omega^{(m+1)} = \omega_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m} \tag{5.16}$$

where $\alpha_m = 2\beta_m$ and equation 5.16 is equivalent to algorithm 5.3 line 6, since the $e^{-\beta_m}$ do not change with i and multiply all the weights. So if we normalize the weights, this term will have no impact.

## 5.5 Random forest

Breiman (2001) presents random forest method which connects bagging algorithm with random variable selection method. In his paper, he shows that random forest model is competitive compared with arcing and boosting techniques. Besides, random forest focus on variance reduction as well as decreasing bias.

**Algorithm 5.5:** Random forest, Breiman (2001)

1 **for** *b=1 to B* **do**
  (a) Draw a bootstrap sample $Z^*$ of size N from the training data.
  (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by re- cursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
  i. Select m variables at random from the p variables.
  ii. Pick the best variable/split-point among the m.
  iii. Split the node into two daughter nodes.
2 Output the ensemble of trees $\{T_b\}_1^B$
  To make a prediction at a new point x:
3 Let $\hat{C}_b(x)$ be the class prediction of the both random forest tree. Then $\hat{C}_{rf}^B(x)$= majority vote $\{\hat{C}_b(x)\}_1^B$

Algorithm 5.5 shows the main procedure of random forest technique. In each step, it uses the bootstrap method to generate a sample data set, so the generated trees in each step are relatively de-correlated. Therefore the average of results in each step tends to reduce the bias of the model. Since an individual tree model is well known for its noisy property, it is beneficial if we average different tree models. Build different identical distributed(i.d.) tress with low correlation can also reduce variance. We can see that if we average $N$ i.i.d. random variables, each has a variance $\sigma^2$, then the variance of the average is $\frac{\sigma^2}{N}$, which will approach to zero when $N$ becomes infinity. In the case that the variables are not i.i.d but i.d, the correlation between each pair is $\rho$, then the variance of the average is:

$$\rho\sigma^2 + \frac{1-\rho}{N}\sigma^2 \tag{5.17}$$

When N increases, $\frac{1-\rho}{N}\sigma^2$ will decrease to zero. So average will also be beneficial to variance reduction in the i.d. case. Moreover, if through reducing the pairwise correlation $\rho$, the first term $\rho\sigma^2$ can also be reduced. This can be achieved by random selecting of the input data. Besides, the tree models are highly non-linear, so the bagging method during the process in the random forest can improve to reduce the correlation $\rho$.

When random forest method is used for solving classification problem, it will draw the final solution by a majority vote among the result from each tree model. In a regression problem, the final result of random forest regression model is simply the mean of results from each tree. In addition, Breiman also makes some recommendation about how to choose the size $M$ to select the subset variables. For classification problem, the minimum size of nodes is one and the default node size is $[\sqrt{p}]$. For regression problem, the minimum size of nodes is five and the default node size is $[p/3]$, where the notation "[ ]" represents the rounding function.

# CHAPTER 6

# MODEL FRAMEWORK AND RESULTS

In this chapter, we focus on describing the framework of our models that apply ensemble method. The main part of this chapter including reviewing some aspects of limit order books, describing the data, introduction of model framework, the measurement that we use to test the models, numerical results of binary classification problem, multi-category classification problem, and so on.

## 6.1 Review of LOBs

According to Roşu (2009), the electronic limit order books prevail in the world's financial market. A lot of company use electronic trading to promote transactions. An order $x$ with volume $\omega_x < 0 (\omega_x > 0$, on the other hand) and price $p_x$ is a duty by its owner to buy (sell, on the other hand) up to $|\omega_x|$ units of the security at a price no greater than (no less than, on the other hand) $p_x$. Orders can match the requirements of the other side when arrive are deemed as market orders. Orders can not match the requirements of the other side when arrive are called limit orders. The bid price $b_t$ is the leading price among buy orders at time $t$. The ask price $a_t$ is the lowest price among the prices that all the active sellers are willing to get. The difference between $a_t$ and $b_t$ is called bid-ask spread. The mean of $a_t$ and $b_t$ is named as mid price. For the limit order book, each time the ask price will be higher than the bid price, since they can not meet the execution requirements mutually. Figure 6.1 shows a snapshot of a 10-level limit order book. The green arrow represents the mid-price between the best bid and best ask price. Additionally, all the ask prices are higher than all the bid prices. If the price of a new coming sell order (respectively, buy order) is lower(respectively, higher) than the best ask(respectively, bid) price, then the order will be executed, otherwise, it will come into the team of limit order books.

## 6.2 Data description

In this section, we give a detailed description of our dataset which is used for training the models. Our empirical calculations are based on a data set that describes the LOB dynamics for

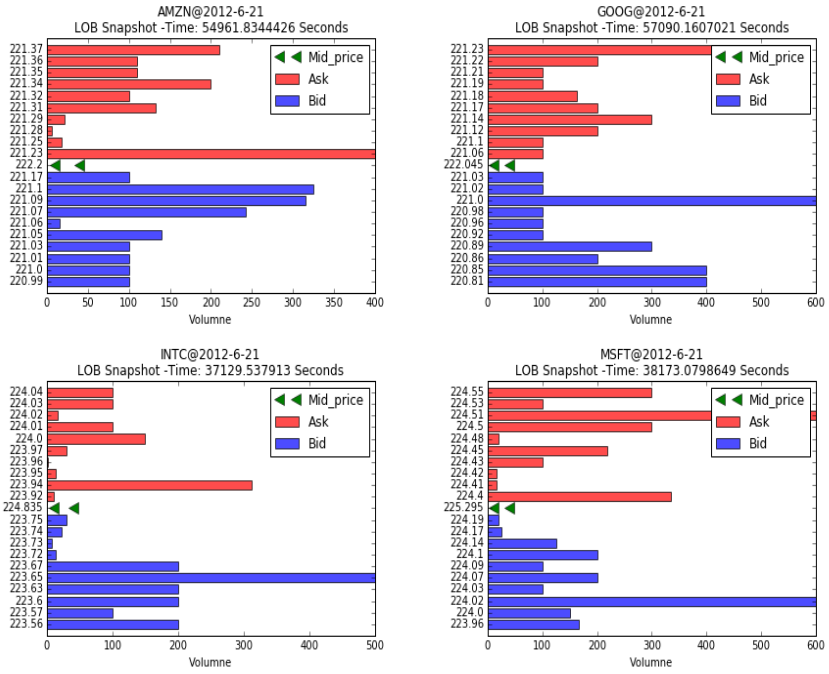


Figure 6.1: Limit order book examples, X-axis denotes the volume of each order book and the Y-axis is the price level of each order book, the left top is AMZN, the right top is GOOG, the left bottom is INTC, and the right bottom is MSFT.Those are snapshots from 2012-06-21

Table 6.1: Limit order book statistics

| | AAPL | AMZN | GOOG | INTC | MSFT |
|---|---|---|---|---|---|
| Total number of events at the best quotes | 400391 | 269748 | 147916 | 624040 | 668765 |
| Percentage of order arrivals | 1.7% | 2.2% | 2.4% | 3.3% | 1.9% |
| Percentage of limit order arrivals | 52.9% | 53.1% | 52.7% | 52.9% | 51.7% |
| Percentage of limit order cancellations | 45.4% | 44.7% | 44.9% | 43.8% | 46.4% |
| Mean bid-ask spread($) | 0.1117 | 0.0118 | 0.0122 | 0.0127 | 0.0115 |
| Mean trade price($) | 45.14 | 31.27 | 41.60 | 24.49 | 28.42 |
| Mean volume at the best quotes(shares) | 5131 | 6740 | 2092 | 3514 | 11423 |
| Mean size of orders(shares) | 617 | 742 | 361 | 569 | 857 |
| Mean size of price - maintaining orders | 455 | 520 | 269 | 383 | 625 |

5 highly liquid stocks traded on Nasdaq on one specific trading day 2012-06-21. The data that we study originates from the LOBSTER database Huang et al. (2015), which lists all order arrivals, limit order arrivals, and cancellations that occur on the Nasdaq platform during 09:30 to 16:00 on each trading day. There is no trading on weekends or public holidays, so those days are excluded in our study. According to past experience, there might some extraordinary trading activities shortly after the opening auction or just before the close auction, so we also exclude the data during the first 30 minutes and last 30 minutes. We use the trading platform of Nasdaq, the tick size for each stock is $\pi = \$0.01$, and the prices of different stocks vary a lot, from less than 300 dollar to above 500 dollars. We choose five stocks from information technology companies containing Apple (AAPL), Amazon (AMZN), Google (GOOG), Microsoft (MSFT), and Intel(INTC). Table 6.1 lists summary statistics describing trading activity for these 5 stocks during our sample period. In our data, the non-displayable orders, commonly called hidden orders are included. The type of the data includes:1) Submission of a new limit order data 2) Cancellation of the orders, which will partially delete the volume of order books 3) Deletion, which will delete all the volume of the order book 4)Execution of a visible limit order 5) Execution of a hidden limit order. For each order, there are 10 level of the bid and ask prices, the bid and ask volume. Time is on a nanosecond($10^{-9}$ S) basis.

Table 6.2: Message book example, a sample on 2012-06-21

| Time(sec) | Event Type | Quantity | Price | Side |
|---|---|---|---|---|
| 34200.017459617 | 5 | 1 | 2238200 | -1 |
| 34200.18960767 | 1 | 21 | 2238100 | 1 |
| 34200.18960767 | 1 | 20 | 2239600 | -1 |
| 34200.18960767 | 1 | 100 | 2237500 | 1 |
| 34200.18960767 | 1 | 13 | 2240000 | -1 |
| 34200.18960767 | 1 | 2 | 2236500 | 1 |

Table 6.3: Message book event type, a sample on 2012-06-21

| Type | Description |
|---|---|
| 1 | Submission of a new limit order |
| 2 | Cancellation(Partial deletion) |
| 3 | Deletion (Total deletion of a limit order) |
| 4 | Execution of a visible limit order |
| 5 | Execution of a hidden limit order |

### 6.2.1 Limit order and message order

Table 6.2, Table 6.3, Table 6.4 and Table 6.5, show examples of message and limit order books, containing a message book sample, message book event type, message book direction and a limit order book sample. Note that execution of a sell (buy) limit order corresponds to a buyer (seller) initial trade, i.e. Buy(sell) trade.

In our data, timestamps are expressed in nano-second and prices are multiplying by 10000 to remove the decimals. Currency is in US dollars and tick size, defined as the smallest possible gap between the ask and bid prices, is one cent. The order quantities are constrained to positive integers

Table 6.4: Message book direction, a sample on 2012-06-21

| Direction | Description |
|---|---|
| -1 | Sell limit order |
| 1 | Buy limit order |

Table 6.5: Limit book example of stock AMZN, a sample on 2012-06-21

| Level 1 | | | | Level 2 | | | | ... |
|---|---|---|---|---|---|---|---|---|
| Ask | | Bid | | Ask | | Bid | | ... |
| Price | Quantity | Price | Quantity | Price | Quantity | Price | Quantity | |
| 2239500 | 100 | 2231800 | 100 | 2239900 | 100 | 2230700 | 200 | ... |
| 2239500 | 100 | 2238100 | 21 | 2239900 | 100 | 2231800 | 100 | ... |
| 2239500 | 100 | 2238100 | 21 | 2239600 | 20 | 2231800 | 100 | ... |
| 2239500 | 100 | 2238100 | 21 | 2239600 | 20 | 2237500 | 100 | ... |
| 2239500 | 100 | 2238100 | 21 | 2239600 | 20 | 2237500 | 100 | ... |
| 2239500 | 100 | 2238100 | 21 | 2239600 | 20 | 2237500 | 100 | ... |

. An example of our limit order book can be found in table 6.5. We can see the order book will be updated by a new entry in the message book. For example, the second line in message book(table 6.2) shows a bid order of a price 2238100 with a volume 21, and the type of this order is submit a new order which changes the information of the order book. In the first line of table 6.5 the best bid price is 2231800, which is lower than the new coming bid order 2238100, so the best bid price will be updated. We can see that in the second line of table 6.5 the level 1 (best) bid price is changed to 2238100 with the volume 21 and the original best bid price 2231800 is transferred to the level 2 (second best) bid price.

### 6.2.2 Features

Time intervals between two consecutive events can vary from milliseconds to several minutes of difference. Event-based data representation avoids issues related to such big differences in data flow. As a result, each of our representations is a vector that contains information for 10 consecutive events. Event-based data description leads to a dataset of approximately half a million representations (i.e. 394337 representations). We represent these events using the 144-dimensional representation proposed recently by Kercheval and Zhang (2015) , formed by three types of features: a) the raw data of a ten-level limit order containing price and volume values for bid and ask orders, b) features describing the state of the LOB exploiting past information and c) features describing the information edge in the raw data by taking time into account. Derivation of time, stock price and volume are calculated for short and long term projection. Particularly, types in features $v_7$ , $v_8$ and $v_9$ are: trades, orders, cancellations, deletion, execution of a visible limit order

Table 6.6: Features summary, nine datasets including basic set, time insensitive set, and time sensitive set, totally 156 features

| Basic Set | Description(i=level index, n=10) |
|---|---|
| $v_1 = \{P_i^{ask}, V_i^{ask}, P_i^{bid}, V_i^{bid}\}_{i=1}^n$ | price and volume(n levels) |
| Time-insensitive Set | Description(i=level index) |
| $v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^n$ | bid ask spreads and mid prices(n levels) |
| $v_3 = \{\lvert P_{i+1}^{ask} - P_1^{ask}\rvert, (P_{i+1}^{bid} - P_1^{bid}\rvert, \lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^{n-1}$ | price difference |
| $v_4 = \{\frac{1}{n}\sum_{i=1}^n P_i^{ask}, \frac{1}{n}\sum_{i=1}^n P_i^{bid}, \frac{1}{n}\sum_{i=1}^n V_i^{ask}, \frac{1}{n}\sum_{i=1}^n V_i^{bid}\}$ | mean prices and volumes |
| $v_5 = \{\sum_{i=1}^n (P_i^{ask} - P_i^{bid}), \sum_{i=1}^n (V_i^{ask} - V_i^{bid})\}$ | accumulated difference |
| Time-sensitive Set | Description(i=level index) |
| $v_6 = \{\partial P_i^{ask}/\partial t, \partial P_i^{bid}/\partial t, \partial V_i^{ask}/\partial t, \partial V_i^{bid}/\partial t\}_{i=1}^n$ | price and volume derivatives |
| $v_7 = \{\lambda_{\Delta_t}^{la}, \lambda_{\Delta_t}^{lb}, \lambda_{\Delta_t}^{ma}, \lambda_{\Delta_t}^{mb}, \lambda_{\Delta_t}^{ca}, \lambda_{\Delta_t}^{cb}\}$ | average intensity of each type |
| $v_8 = \{1_{\lambda_{\Delta_t}^{la} > \lambda_{\Delta_T}^{la}}, 1_{\lambda_{\Delta_t}^{lb} > \lambda_{\Delta_T}^{lb}}, 1_{\lambda_{\Delta_t}^{ma} > \lambda_{\Delta_T}^{ma}}, 1_{\lambda_{\Delta_t}^{mb} > \lambda_{\Delta_T}^{mb}}\}$ | relative intensity indicators |
| $v_9 = \{\partial\lambda^{ma}/\partial t, \partial\lambda^{lb}/\partial t, \partial\lambda^{mb}/\partial t, \partial\lambda^{la}/\partial t\}$ | accelerations(/limit) |

and execution of a hidden limit order respectively. The features are listed in the following figure and can be referred in Kercheval and Zhang (2015), details can be found in table 6.6.

### 6.2.3 Order Price, order book volume, order book type, and relative depth

Here we study some properties of order price, order book volume, and order book type in a given day. We can see from figure 6.2 to figure 6.6, both best ask price and best bid price are decreasing on 2012-06-21.

From figure 6.7 we can see that for all the five stocks, among all the types, putting a new order book and canceling all the order book accounting for the vast majority of the proportion. In chapter 1, we have mentioned that a momentum strategy will put and cancel order book in a very short time to ignite the momentum from which the executor can beneficial from the spread by other investors.

Figure 6.8 shows the same trend with the situation of order book volume. It is another example to show the arbitrage strategy of high-frequency trading companies. Figure 6.9, figure 6.10, and figure 6.11 show the number of trading, the volume of trading and relative depth of trading on 2012-06-21.

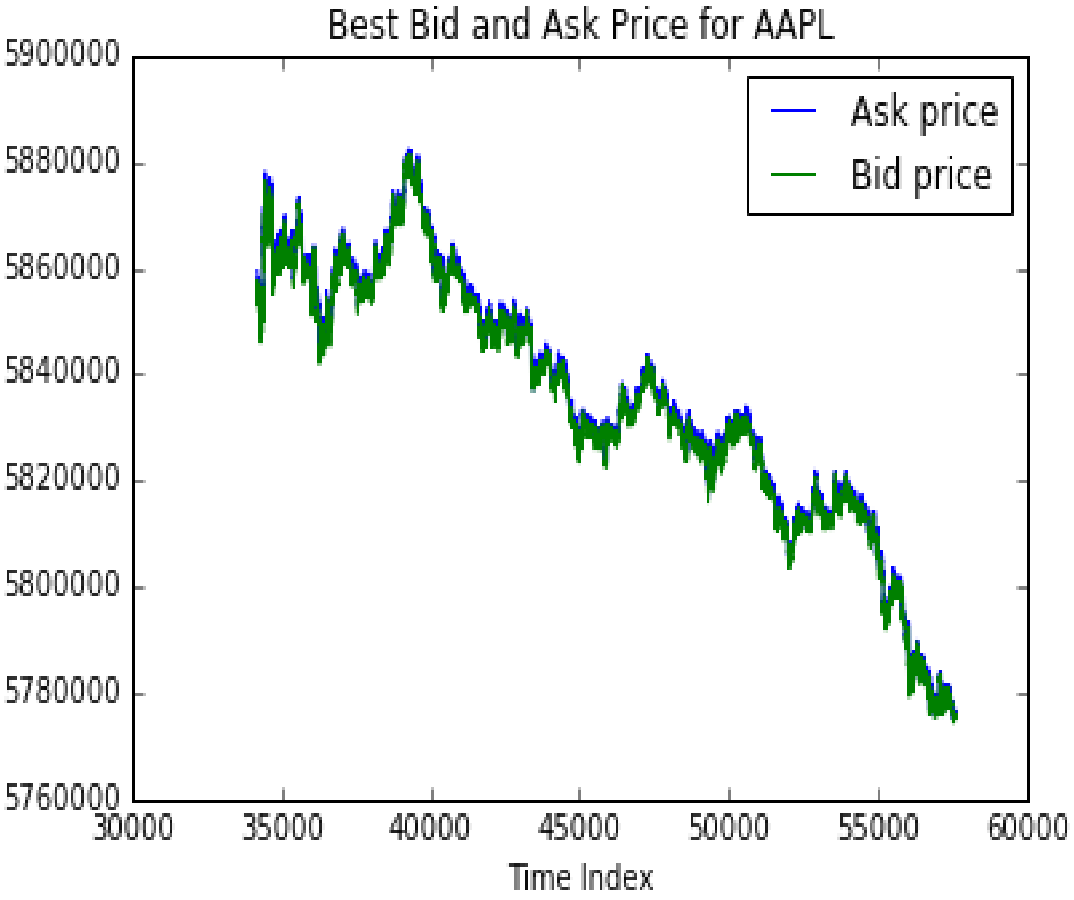


Figure 6.2: Best bid and ask price of AAPL in 2012-06-21, X-axis is the time index, Y-axis is the prices multiply by 100, the green line is the bid price and the blue line is the ask price


Figure 6.3: Best bid and ask price of AMZN in 2012-06-21, X-axis is the time index, Y-axis is the prices multiply by 100, the green line is the bid price and the blue line is the ask price


Figure 6.4: Best bid and ask price of GOOG in 2012-06-21, X-axis is the time index, Y-axis is the prices multiply by 100, the green line is the bid price and the blue line is the ask price

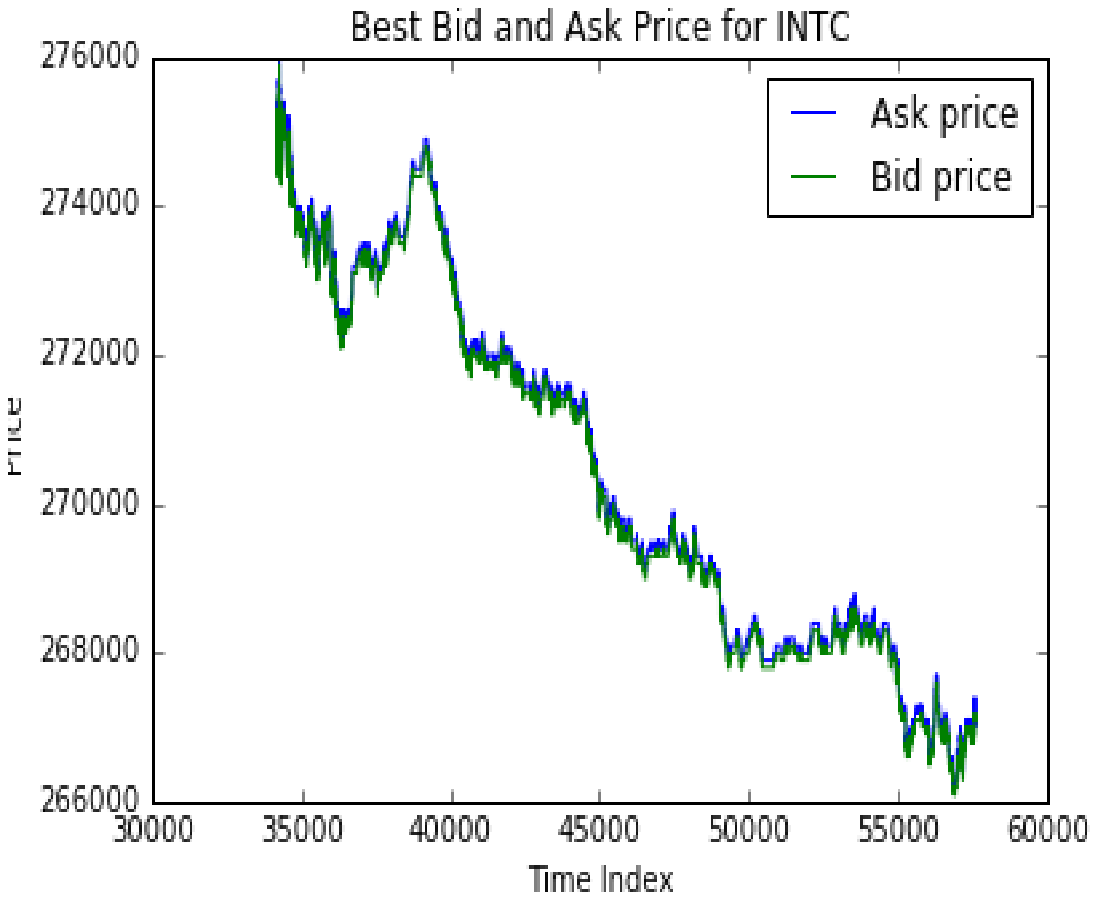


Figure 6.5: Best bid and ask price of INTC in 2012-06-21, X-axis is the time index, Y-axis is the prices multiply by 100, the green line is the bid price and the blue line is the ask price

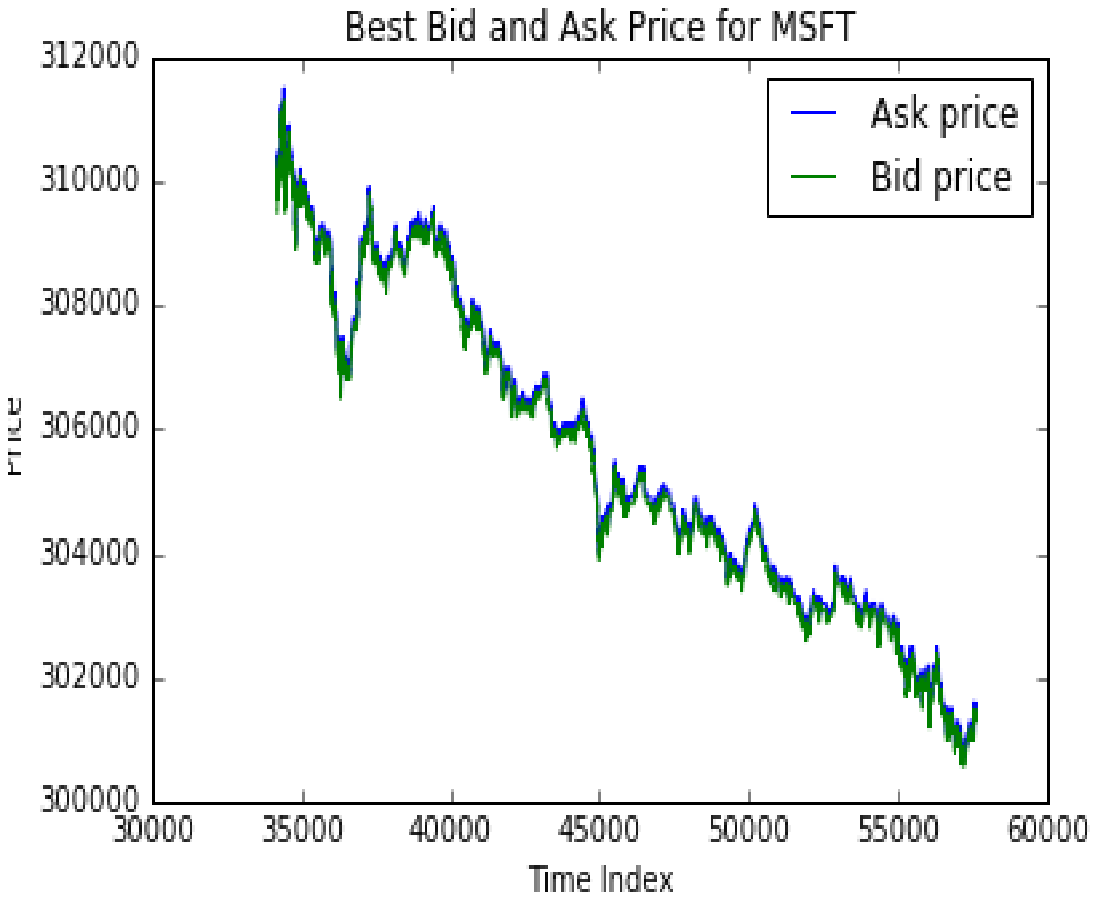


Figure 6.6: Best bid and ask price of MSFT in 2012-06-21, X-axis is the time index, Y-axis is the prices multiply by 100, the green line is the bid price and the blue line is the ask price

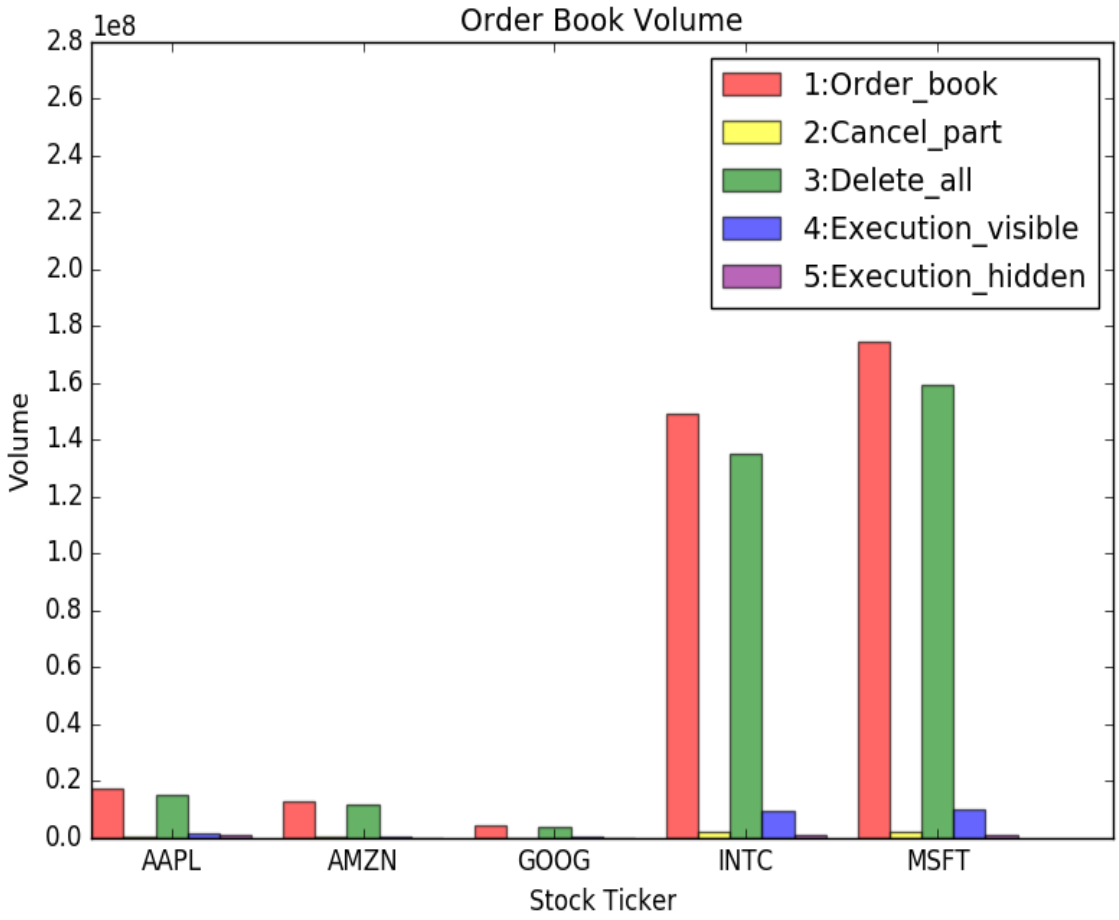


Figure 6.7: Order book volume comparison for five stocks in 2012-06-21, X-axis is the stock ticker and Y-axis is the volume of trade

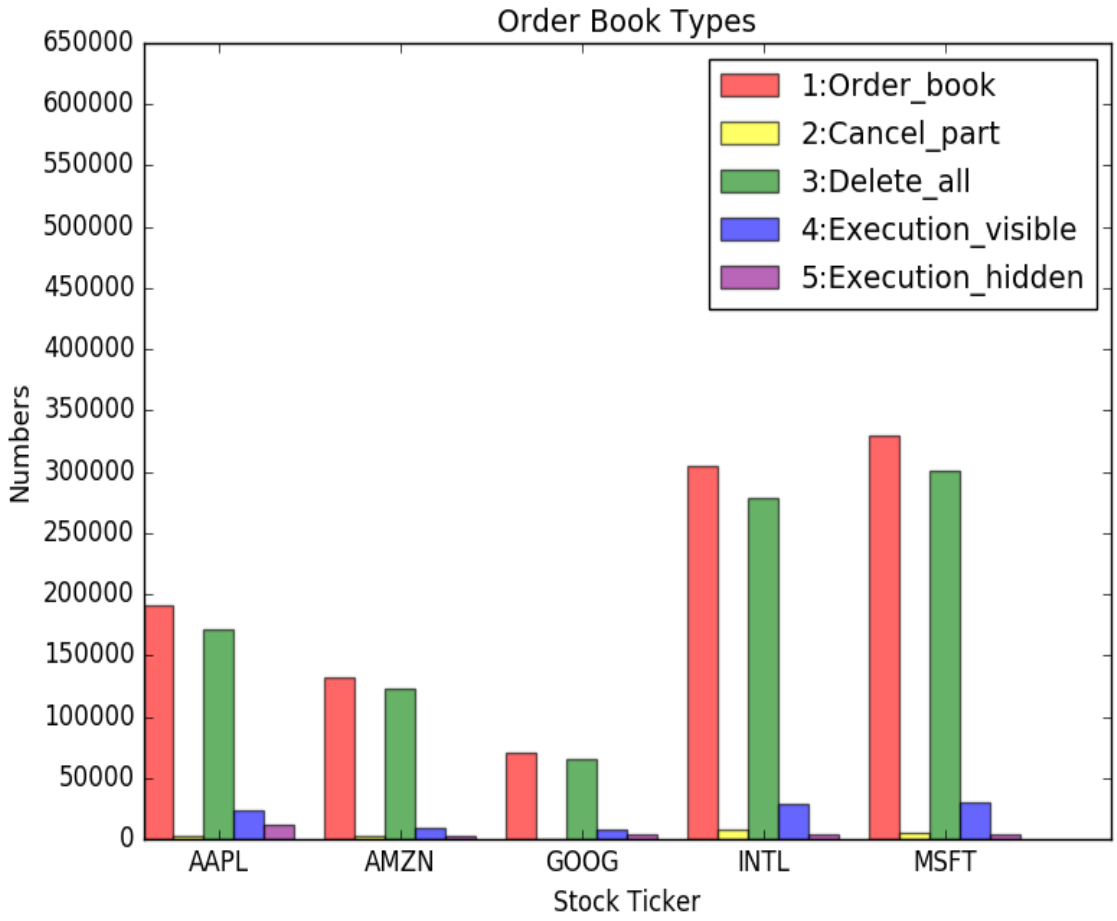


Figure 6.8: Order book type comparison for five stocks in 2012-06-21, X-axis is the stock ticker and Y-axis is the number of different order book types

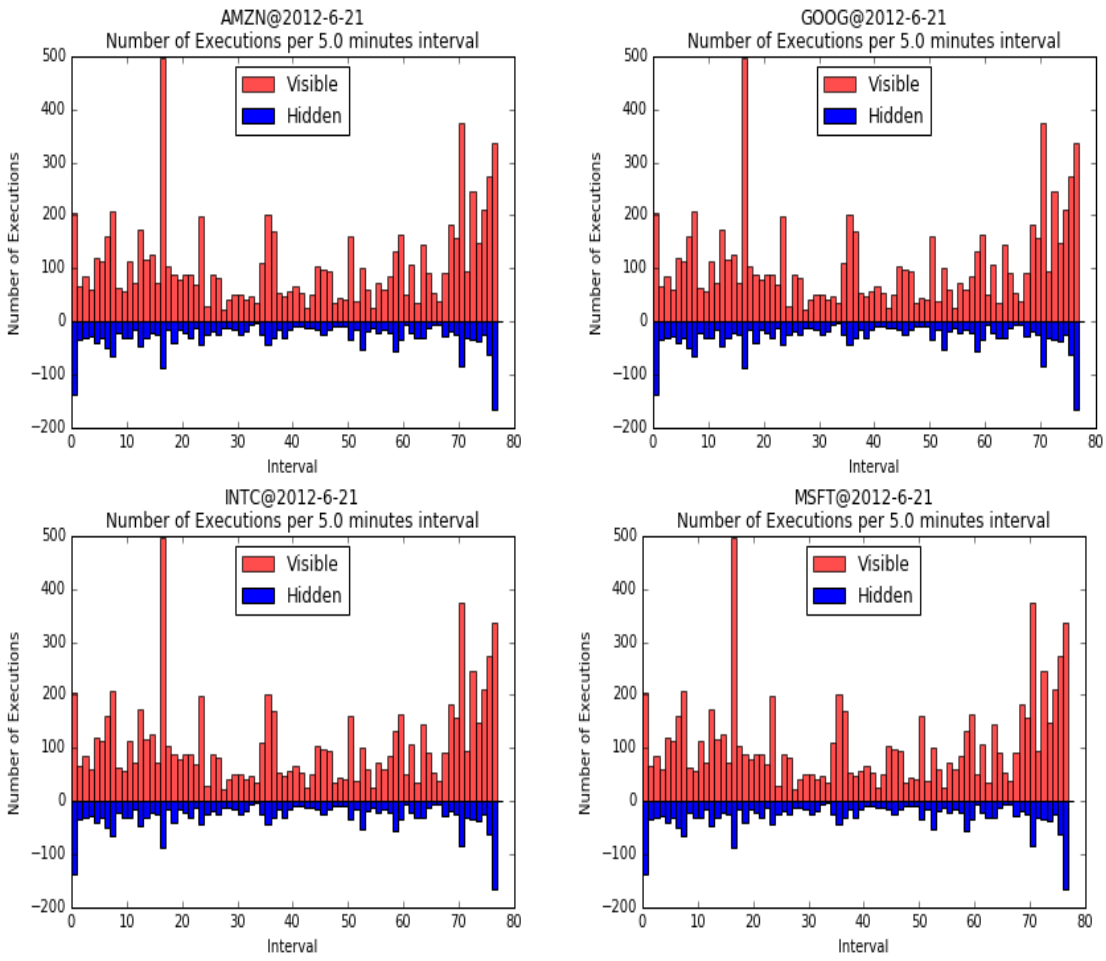


Figure 6.9: Number of trading in 2012-06-21, X-axis is the time interval and Y-axis is the number of executions. Each band in time interval represents 5 minutes.The top left panel shows the result of AMZN, the top right panel shows the result of GOOG, the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

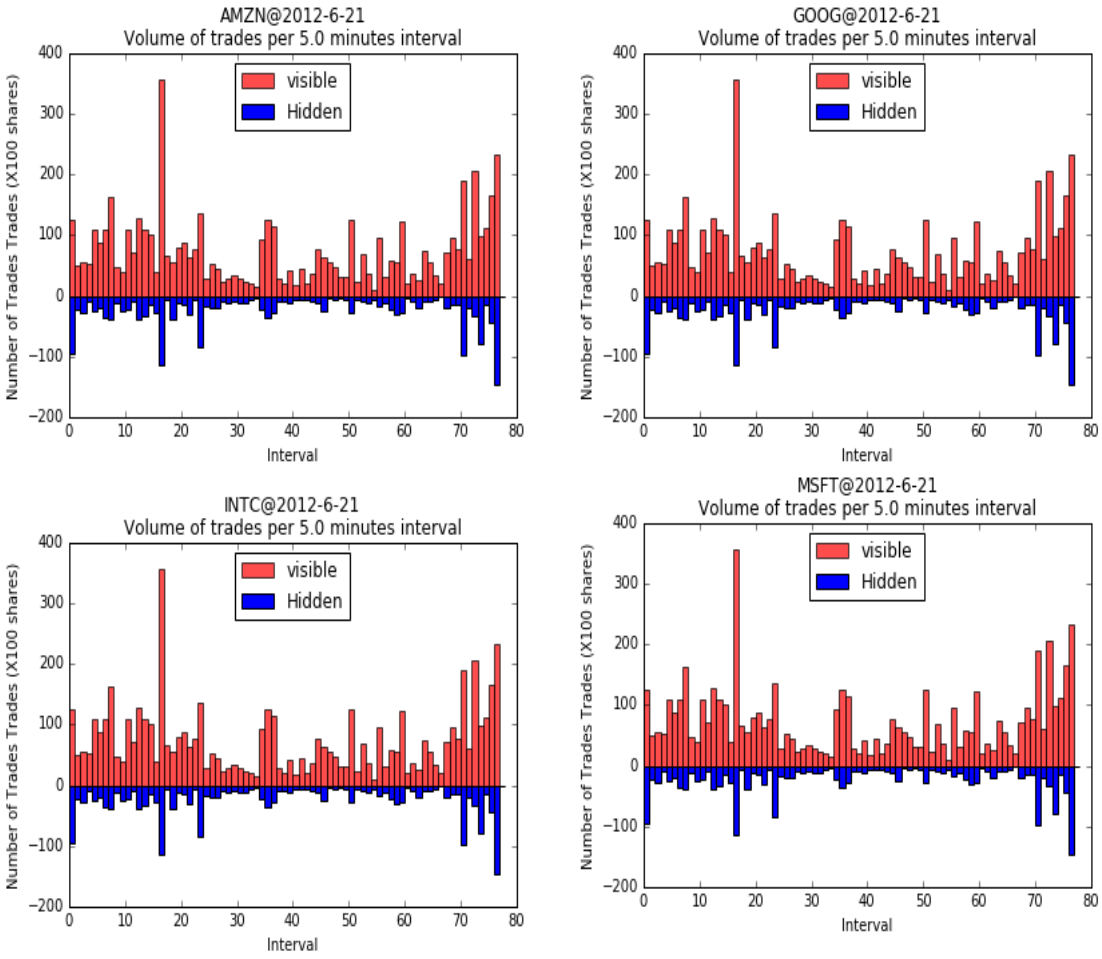


Figure 6.10: Volume of trading in 2012-06-21, X-axis is the time interval and Y-axis is the number of executions. Each band in time interval represents 5 minutes.The top left panel shows the result of AMZN, the top right panel shows the result of GOOG, the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

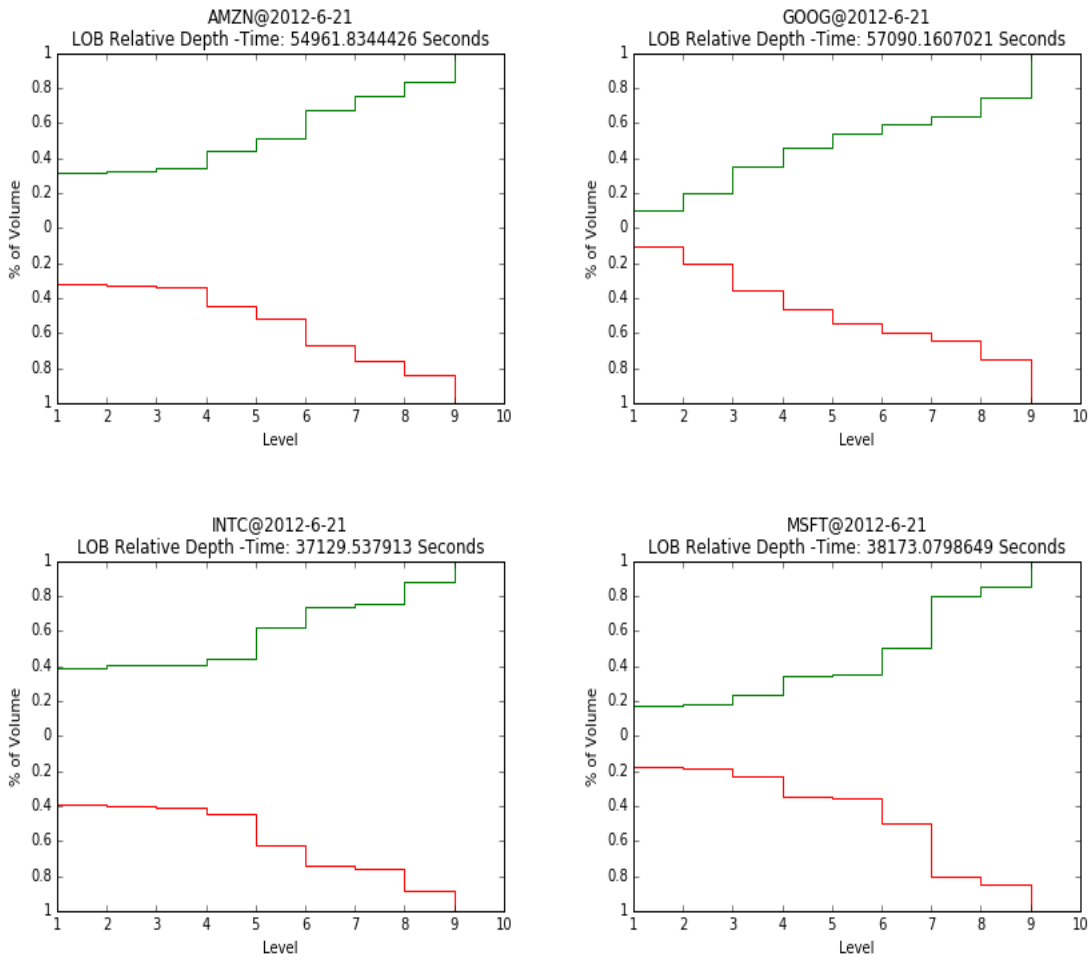


Figure 6.11: Relative depth in 2012-06-21, X-axis is the price level and Y-axis is the percentage of total volume. The top left panel shows the result of AMZN, the top right panel shows the result of GOOG, the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

## 6.3 Model framework

Our model aims to predict the arbitrage opportunities of limit order book price change. There are two kinds of arbitrage in our case: ask price lower arbitrage, denoted by ask-low, and bid price higher arbitrage, denoted by bid-high. We know that at any given time, the ask price will always higher than the bid price, so there is no arbitrage. However if the ask price at a future $\Delta t$ time is lower than the current bid price, then there exists an arbitrage. How to realize it? The arbitrageur can sell short the stock at current bid price $b_t$ and wait for $\Delta t$ time, buy the stock from the market with the price of ask price $a_{t+\Delta t}$, and return the stock required by the short positions, the profit he can earn without risk is $b_t - a_{t+\Delta t}$. On the bid-high situation, the arbitrage strategy is similar. Figure 6.12, figure 6.13, and figure 6.14 give us examples of ask-low arbitrage, bid-high arbitrage, and no arbitrage cases respectively.

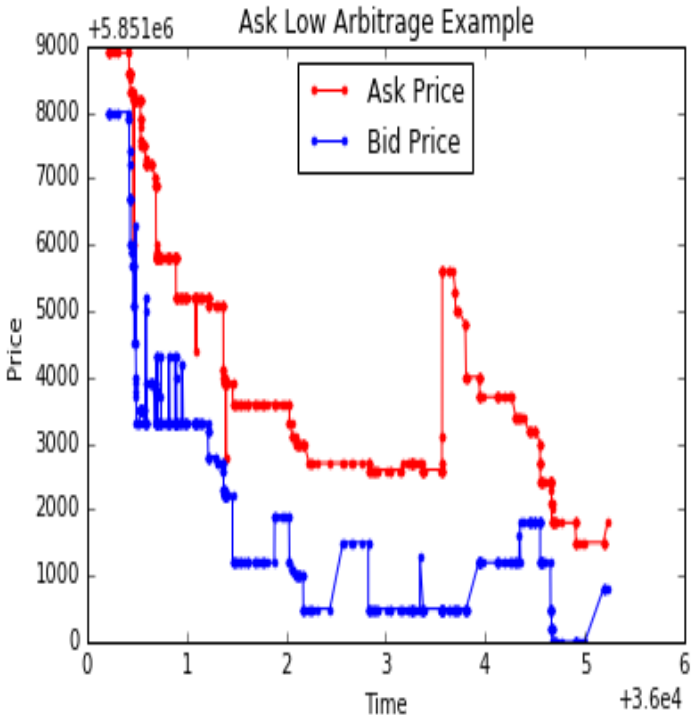


Figure 6.12: Ask-low arbitrage example, X-axis represents time in second and Y-axis represents the price times 10000 in dollar, the red line is the ask price and the blue line is the bid price. The ask price after 10 seconds is lower than the current bid price.

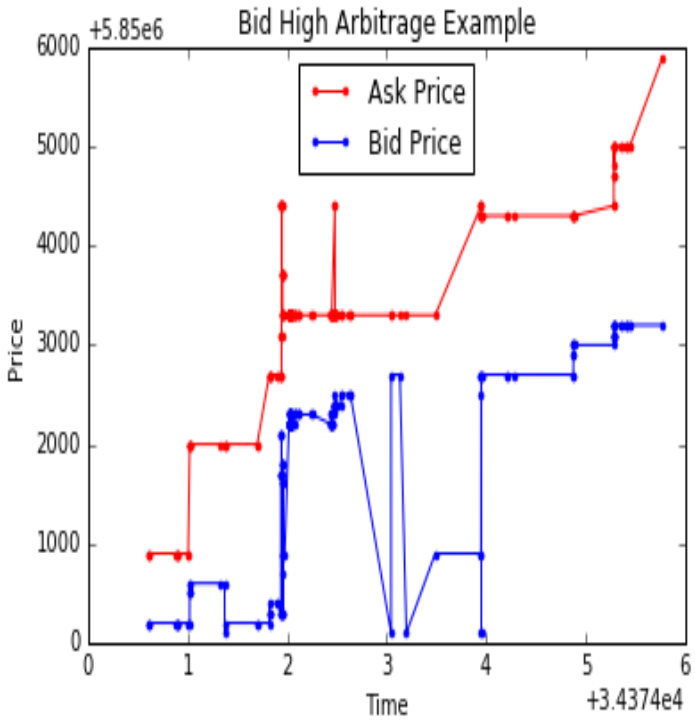


Figure 6.13: Bid-high arbitrage example,X-axis represents time in second and Y-axis represents the price times 10000 in dollar, the red line is the ask price and the blue line is the bid price. The bid price after 5 seconds is higher than the current ask price

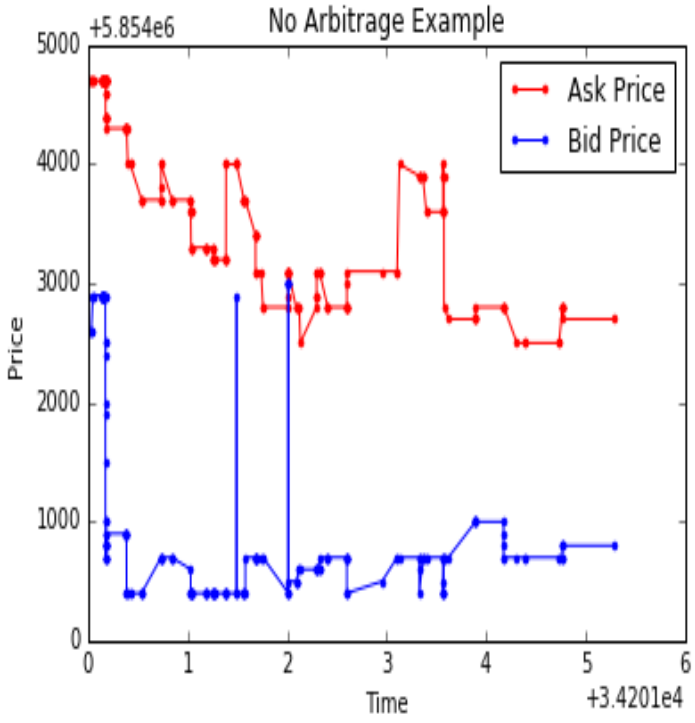


Figure 6.14: No arbitrage example,X-axis represents time in second and Y-axis represents the price times 10000 in dollar, the red line is the ask price and the blue line is the bid price. There is no price spread crossing after 5 seconds.

Instead of predicting the price change of future events, as most past papers did, we focus on predict the price change in a fixed future time interval, e.g. 5 seconds later.

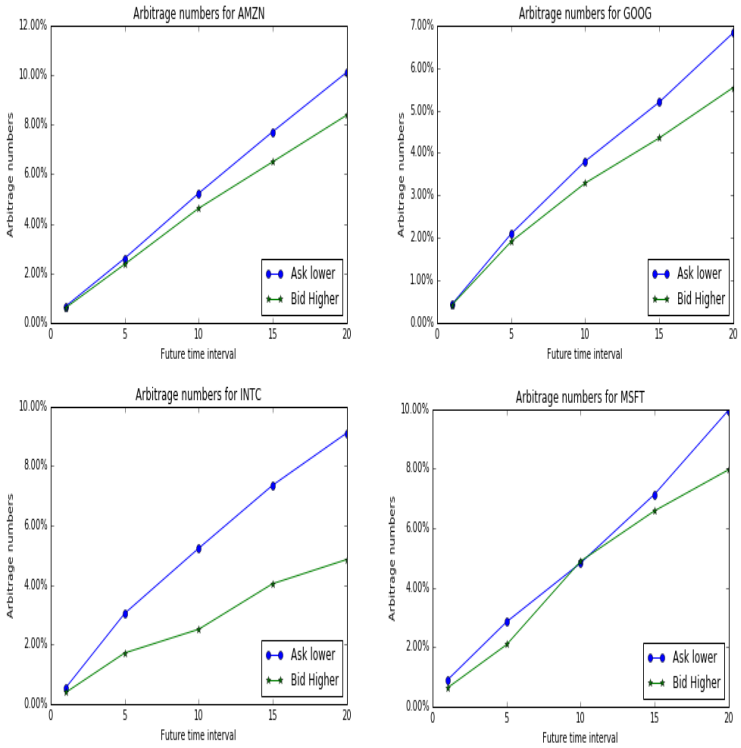


Figure 6.15: Arbitrage numbers based on future time interval, X-axis represents different time intervals and Y-axis is percentage of arbitrage opportunities among all the transactions.Blue line is ask-low cases and green line is bid-high cases

From figure 6.15, we can find that the arbitrage numbers for both bid-high and ask-low opportunities are increasing. For the five seconds time interval, the percentage of arbitrage opportunities among all the transaction is around 2% for each stock listed in the figure. For the ten seconds time interval, the arbitrage percentage increases to around 4%, which almost doubled.

Generally speaking, separate learning models should be built for different metrics depicting limit order book dynamics. To build a machine learning model for a specific metric, the following four-phase process is employed. 1) Features representation: transforming the data in the order book and message book to a format suitable for machine learning methods to manipulate. 2) Learning model construction: building models of AdaBoost and random forest with the given features from

step 1. 3) Learning model validation: Evaluating and validating the model by means of particular performance measurements. 4) Unseen-data classification: applying the constructed learning model to forecast the chosen testing set in real time.

## 6.4 Model measurement and numerical results

As we mentioned before, the arbitrage opportunities are rare events among all the placement of order books. For example, the percentages of arbitrage opportunities of five seconds interval is only around 2 %, so the accuracy rate of a model is not a good measurement. Because if we define all the results of a testing sample as no arbitrage, we can still get a very high accuracy rate, which is around 98 %. Therefore we introduce precision, recall and f1 score to deal with this problem.
Some terms here:
Positive (P): Observation is positive. In our case, it means that arbitrage opportunity will occur in the future.
Negative (N): Observation is negation. In our case, it means that there is no arbitrage in the future.
True positive(TP): Observation is positive and is classified as positive. In our case, it means that the detected arbitrage opportunity is a real arbitrage opportunity.
False negative(FN):Observation is positive but is classified as negative, in our case, we do not detect a real arbitrage opportunity
True negative(TN): Observation is negative and is classified as negative, in our case, it means the detected no arbitrage case is actually no arbitrage
False positive(FP): Observation is negative but is classified as positive, in our case, it means that the detected arbitrage case is actually no arbitrage.
The precision is:

$$Precision = \frac{TP}{TP + FP} = \frac{positive\ predicted\ correctly}{all\ positive\ predictions} \tag{6.1}$$

The recall is:

$$Recall = \frac{TP}{TP + FN} = \frac{positive\ predicted\ correctly}{all\ positive\ observations} \tag{6.2}$$

F1 score:

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall} \tag{6.3}$$

From equation 6.3, $F1$ score is the harmonic mean of precision and recall

Table 6.7 shows the results of predicting ask-lower opportunity of stock AMZN.

Table 6.7: AMZN ask-low arbitrage opportunities prediction(5 seconds)

| Model | Training time(s) | Training F1 score | Test time(s) | Test Recall | Test Precision | Test F1 score |
|---|---|---|---|---|---|---|
| Logistic regression(Lasso penalty) | 260.7 | 8.8 % | 0.002 | 2.9% | 75.0% | 5.6% |
| Logistic regression(Ridge penalty) | 7.2 | 8.8 % | 0.01 | 2.9% | 75.0% | 5.6% |
| SVM(Poly 2 kernal, 5000 estimator) | 75.7 | 61.5 % | 4.3 | 29.1% | 96.8% | 44.8% |
| Decision Tree(no pruning) | 3.9 | 61.8 % | 0.003 | 30.1% | 91.2% | 45.3% |
| AdaBoost(number of estimate=100) | 30.0 | 96.5 % | 0.04 | 73.8 % | 92.7% | 82.2% |
| Random forest(number of estimate=100) | 37.5 | 99.1 % | 0.11 | 72.8 % | 96.2% | 82.9% |

Here the number of training samples is 90000 and the number of testing samples are 10000. Computer has 8G memory and Intel Xeon E3 processor. The results for other stocks can be found in Appendix A

## 6.5 Multiclass prediction problem

There are three cases of arbitrage opportunities in our research including ask-low, bid-high, and no arbitrage. If we want to predict these three cases at the same time, then our model is transformed into a multi-category classification problem. For this kind of problem, we focus on learning a predictor $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ represents a set of categories and $\mathcal{X}$ is the set of independent variables. The most widely accepted way that deals with the multiclass learning problem, is to reduce the multiclass classification to binary classification. Two straightforward methods, one vs. rest and one vs. one, are introduced in the following:

In the one vs. rest method, we train $k$ binary classifiers, where $k$ is the number of classes for response $y$. Each binary classifier distinguishes the dependent variable $y$ between one class and the

rest of the other classes. More clearly, given a training set $S = (x_1, y_1), ..., (x_m, y_m)$, where each $y_i$ is in $\mathcal{Y}$ and $m$ is the amount of data sample, we build $k$ binary training sets, $S_1, ..., S_k$, where $S_i = (x_1, (-1)^{\mathbb{1}_{[y_1 \neq i]}}), ..., (x_m, (-1)^{\mathbb{1}_{[y_m \neq i]}})$. In word description, $S_i$ is the set of data labeled 1 if their class is i, and -1 otherwise. For each $i \in [k]$, we train a binary predictor $h_i : \mathcal{X} \to \{+1, -1\}$ based on $S_i$ and expect that the result of $h_i(x)$ will equal to 1 if and only if $x$ is in class $i$. Therefore, given $h_1, ..., h_k$, we build a multiclass predictor using the principle:

$$h(x) \in \underset{i \in [k]}{\operatorname{argmax}}\ h_i(x) \tag{6.4}$$

When there are more than one predictors that predict "1", we can break the ties by choosing the minimal index in $argmax_i h_i(x)$. The pseudo-code of the one vs.rest method is given in algorithm 6.1.

**Algorithm 6.1:** One vs. rest algorithm,

**Input:**;
training set $S = (x_1, y_1), ..., (x_m, y_m)$;
algorithm for binary classification $A$;
**for** $i \in \mathcal{Y}$ **do**
  let $S_i = (x_1, (-1)^{\mathbb{1}_{[y_1 \neq i]}}), ..., (x_m, (-1)^{\mathbb{1}_{[y_m \neq i]}})$. ;
  let $h_i = A(S_i)$ ;
**Output:** ;
the multiclass hypothesis defined by $h(x) \in \underset{i \in \mathcal{Y}}{\operatorname{argmax}}\ h_i(x)$;

Another widely used multiclass approach is one vs. one, which compares all pairs of classes with each other. That is, given a training set $S = (x_1, y_1), ..., (x_m, y_m)$, where every $y_i$ is in $[k]$, we build a binary training sequence, $S_{ij}$, for every $1 \leq i < j \leq k$. The set $S_{ij}$ include all instances from $S$ whose label is either $i$ or $j$. In each $S_{ij}$, we set the label of $S_{ij}$ as $+1$ if the class label in $S$ is $i$ and -1 if the class label in $S$ is $j$. Wo conduct the same process on every $S_{ij}$ to get a multiclass classifier $h_{ij}$. The final result of the multiclass classifier is obtained by a highest number of "wins".

The pseudocode of the one vs. one method is given in algorithm 6.2:

**Algorithm 6.2:** One vs. one method

**Input:**
Training set $S = (x_1, y_1), ..., (x_m, y_m)$;
Algorithm for binary classification $A$;
**for** *each* $i, j$ *in* $\mathcal{Y}$, *such that* $i < j$ **do**
    Initialize $S_{ij}$ to be the empty sequence;
    **for** $t = 1, ..., m$ **do**
        If $y_t = i$ add $(\mathrm{x}_t, +1)$ *to* $\mathrm{S}_{ij}$;
        If $y_t = j$ add $(\mathrm{x}_t, -1)$ *to* $\mathrm{S}_{ij}$;
    Let $h_{ij} = A(S_{ij})$;
**Output:**
The multiclass hypothesis defined by $h(x) \in argmax_{i \in \mathcal{Y}}(\sum_{j \in \mathcal{Y}} sign(j - i) h_{ij}(x))$ ;

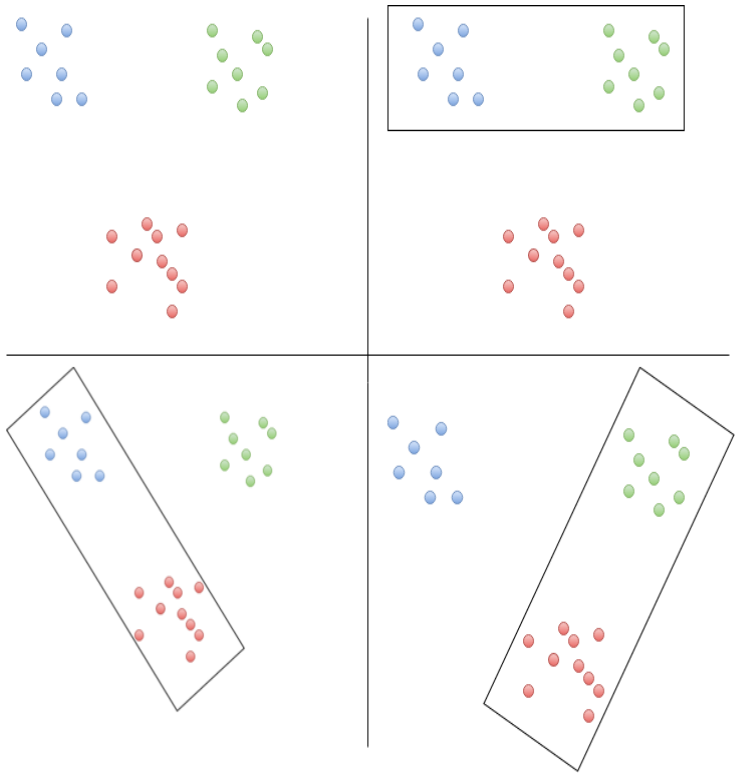

Figure 6.16: One vs. one algorithm, the upper left figure shows the original three classes(represent as red, green, and blue), the upper right figure only considers blue and green classes, the bottom left figure only considers blue and red classes, and bottom right figure only considers red and green classes.

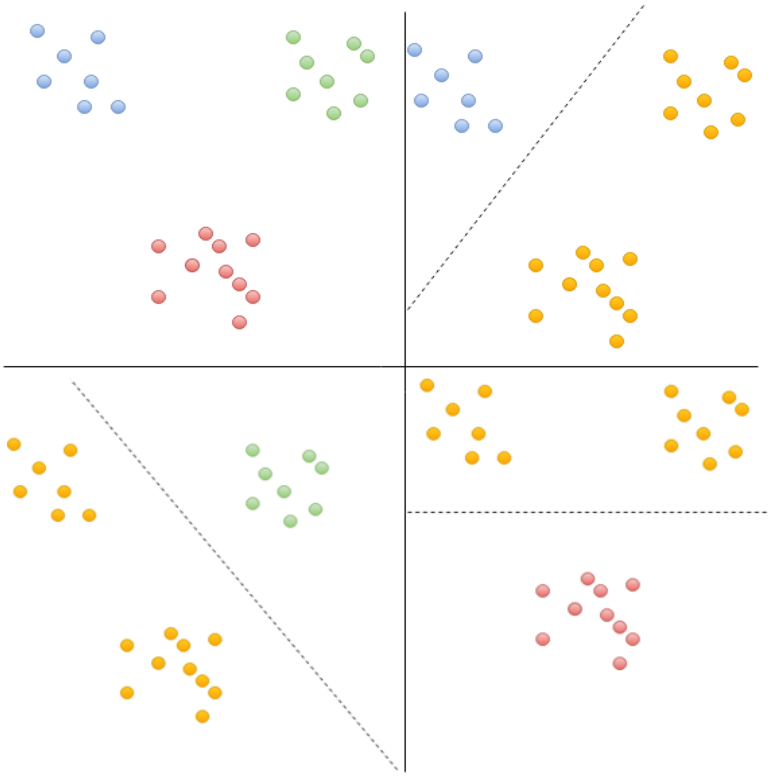

Figure 6.17: One vs. rest algorithm, the upper left figure shows the original three classes(represent as red, green, and blue), the upper right figure considers blue and the rest classes, the bottom left figure considers green and the rest classes, and bottom right figure considers red and the rest classes.

## 6.6 Feature importance

In statistical and machine learning area, feature and variable selection play a very important role. There are some benefits of the feature selection to researchers: 1) It can improve the prediction performance of the predictors. 2) The predictors can be faster and more cost-efficiency. 3) It help people to better understand the meaning hidden behind models to generate the data. Breiman (2001) proposes a formula to evaluate the importance of a variable $X_m$ for predicting $Y$ by adding up the weighted impurity decreases $p(t)\Delta_i(s_t, t)$ for all nodes $t$ where $X_m$ is used, averaged over all $N_T$ trees in the forest:

$$Imp(X_m) = \frac{1}{N_T} \sum_{T} \sum_{t \in T: v(s_t) = X_m} p(t)\Delta_i(s_t, t)$$
$$\Delta_i(s, t) = i(t) - p_L i(t_l) - p_R i(t_R) \tag{6.5}$$

where $i(t)$ is a given impurity measure, $p(t)$ is the proportion $N_t/N$ of samples reaching $t$ , $v(s_t)$ is the variable used in split $s_t$, $p_L = N_{t_L}/N_t$ and $p_R = N_{t_R}/N_t$. The equation 6.5 is referred as *Mean Decrease Impurity importance* (MDI). Figure 6.18 and table 6.8 show the results of feature importance of the four stocks based on random forest models. We find that feature importance varies from stock to stock, there is no obvious rule according to it. To look more closely, we list the top 3 features for each stock. For stock AMZN, the first three important features are 107, 106 and 116 which represents $\partial P_5^{ask}/\partial t$, $\partial P_6^{ask}/\partial t$, and $\partial P_4^{bid}/\partial t$ respectively. The first 3 import stock for GOOS are 41, 3, 51 which represent $P_1^{ask} - P_1^{bid}$, $P_3^{ask}$, and $P_1^{ask} + P_1^{bid}$. For stock INTC, they are 16, 36, 40 which represents $V_6^{ask}, V_6^{bid}$, and $V_{10}^{bid}$. For stock MSFT, they are 28, 12 and 16 which represent $P_8^{bid}, V_2^{ask}$, and $V_6^{ask}$. We can see that the derivative of price to time is important to stock AMZN. For stock GOOG, the relationship of bid price and ask price becomes more important. Ask volume and bid volume are essential for stock INTC, price and volume are critical to stock MSFT.

## 6.7 Trading strategy

There are some measurements to test the performance of trading strategy, in our research we choose the Profit-and-Loss(PnL). According to Zhou et al. (2015), suppose we have $y_t = S_{t+H} - S_t$ as the price change from time $t$ to $t + H$, and $\hat{y}_t$ as the prediction result of our model, $c$ is the

Table 6.8: Feature importance based on random forest model

| AMZN | | GOOG | | INTC | | MSFT | |
|---|---|---|---|---|---|---|---|
| Feature index | Importance | Feature index | Importance | Feature index | Importance | Feature index | Importance |
| 107 | 3.997699 | 41 | 2.497344 | 16 | 4.930035 | 28 | 3.777894 |
| 108 | 2.531017 | 3 | 2.077711 | 36 | 3.584072 | 12 | 3.668482 |
| 116 | 1.72645 | 51 | 1.911522 | 40 | 3.271213 | 16 | 3.462065 |
| 12 | 1.523849 | 71 | 1.860316 | 14 | 2.930865 | 36 | 3.307528 |
| 41 | 1.489288 | 111 | 1.7199 | 30 | 2.830366 | 8 | 3.134871 |
| 52 | 1.401763 | 70 | 1.511696 | 20 | 2.694488 | 26 | 2.863927 |
| 61 | 1.390584 | 82 | 1.503223 | 22 | 2.594267 | 30 | 2.783838 |
| 51 | 1.296778 | 42 | 1.476138 | 38 | 2.575397 | 20 | 2.782014 |
| 1 | 1.26687 | 7 | 1.446197 | 28 | 2.565284 | 24 | 2.752804 |
| 81 | 1.24307 | 52 | 1.413531 | 32 | 2.510037 | 32 | 2.722132 |
| 3 | 1.240166 | 53 | 1.410323 | 18 | 2.485869 | 38 | 2.660868 |
| 9 | 1.222178 | 80 | 1.379994 | 24 | 2.388528 | 81 | 2.45301 |
| 79 | 1.187279 | 8 | 1.3286 | 82 | 2.289012 | 82 | 2.312318 |
| 80 | 1.168383 | 84 | 1.316409 | 12 | 2.165716 | 40 | 2.276727 |
| 53 | 1.16105 | 24 | 1.246773 | 26 | 2.119569 | 34 | 2.149401 |
| 54 | 1.150503 | 15 | 1.223152 | 4 | 2.032679 | 10 | 1.790342 |
| 82 | 1.150272 | 55 | 1.214248 | 8 | 2.007556 | 18 | 1.765228 |
| 118 | 1.133169 | 16 | 1.211959 | 34 | 1.950585 | 14 | 1.727172 |
| 7 | 1.131877 | 57 | 1.199527 | 81 | 1.896318 | 22 | 1.649081 |
| 84 | 1.099623 | 1 | 1.178082 | 6 | 1.697322 | 84 | 1.515854 |

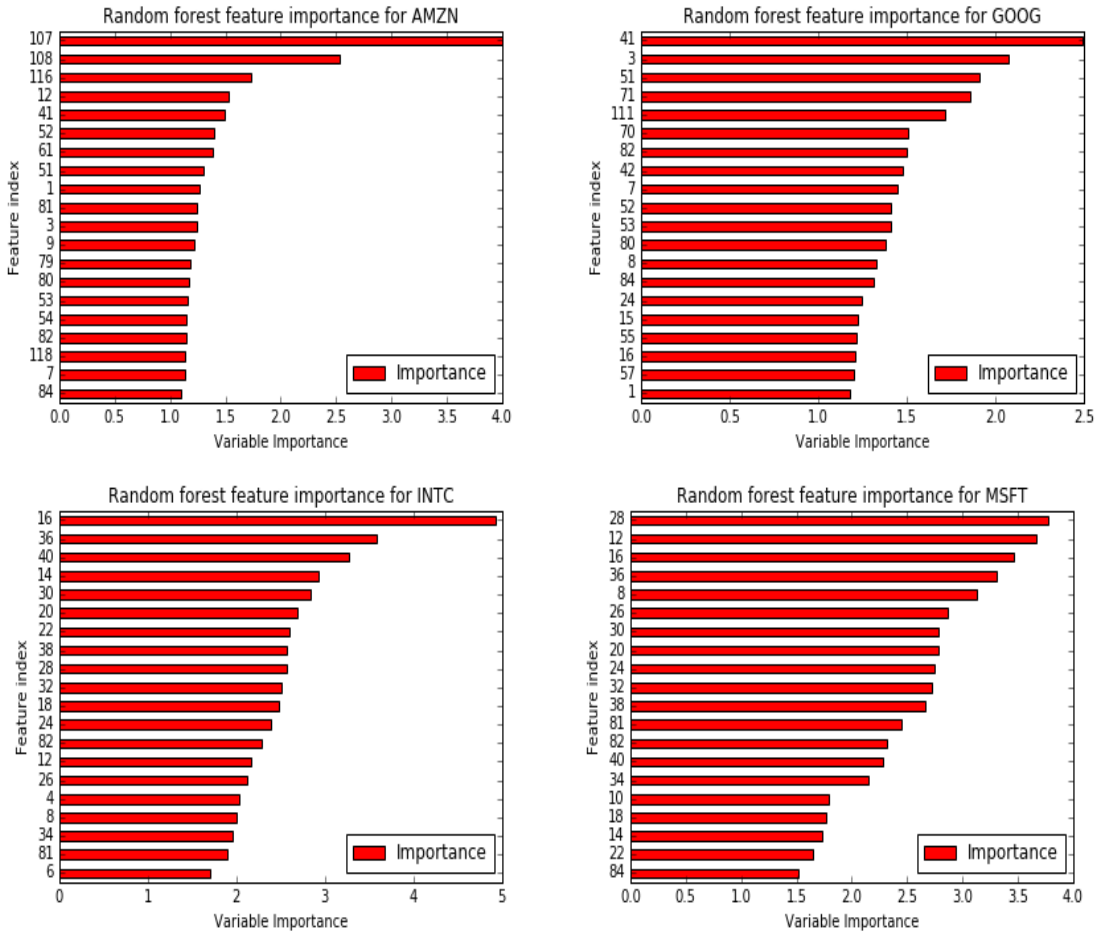


Figure 6.18: Feature importance. X-axis is the feature importance and Y-axis is the corresponding feature index.The top left panel shows the result of AMZN, the top right panel shows the result of GOOG, the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

trading cost. Assume that we will make a trading decision if the profit is bigger than a significant level, $\hat{y_t} > \alpha$, or conversely, the loss is bigger than a significant level, $\hat{y_t} < -\alpha$.The PnL within the time interval $[t, t+H]$ is the profit and loss through a transaction which can be written as follows:

$$PnL = \begin{cases} y - c & y >= \alpha, buy\ action \\ -y - c & y <= -\alpha, sell\ short\ action \\ 0 & otherwise \end{cases} \tag{6.6}$$

where y is the net capital gain from a transaction, $\alpha$ is significant level and $c$ is the trading cost. The liquidity of taking the orders and information acquisition usually influence the trading cost. In addition, when a large number of volumes come to the market, the trading cost tends to increases. The stocks that we studied are all big information technology companies, so the liquidity of those companies is relatively high. Usually, the trading cost of liquid products is low than one bid-ask spread, which is around \$0.02 per trade on NASDAQ. Therefore it is feasible that we assume our trading cost is \$0.02. In the following, we define a simple buy-low and sell-high strategy. The details are in algorithm 6.3

**Algorithm 6.3:** Naive trading algorithm,

```
initialize: PnL=0
for i =1 to length(test_set) do
    input test_set[i] features into model and get result of Predict[i]
    if Predict[i]==1(Ask low) then
        Sell short at bid price
        Clear the short option Δt seconds later
        PnL+=Bid_price_t − Ask_price_{t+Δt} − cost
    else if Predicted[i]==-1(Bid high) then
        Buy at ask price
        Sell at bid price Δt seconds later
        PnL+=Bid_price_{t+Δt} − Ask_price_t − cost
    else
        Take no action
    return PnL
```

Figure 6.19 gives us an example of how to conduct a naive arbitrage trading, when the ask-low cases (green triangles) occur, we can sell short at the current bid price and buy the stock back at

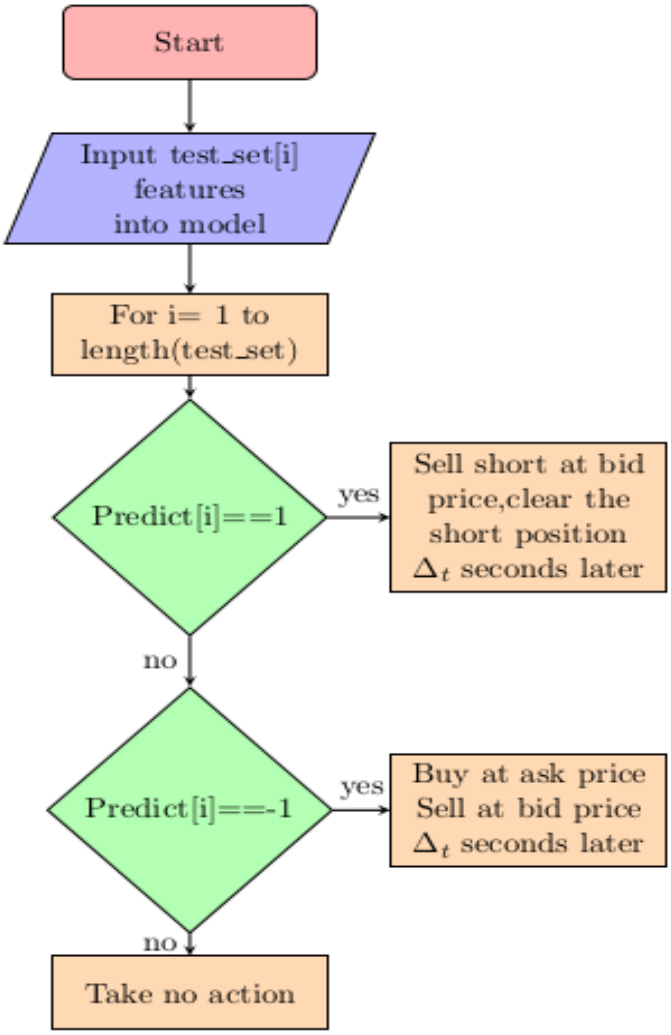


Figure 6.19: Naive trading strategy framework. When the ask-low cases (prediction result as 1) occur, we can sell short at the current bid price and buy the stock back at the future ask price. Similarly, when the bid-high cases (prediction result as -1) occurs, we can buy the stock at the current ask price and sell the stock to the market at the future bid price.

the future ask price. Similarly, when the bid-high cases (red triangles) occurs, we can buy the stock at the current ask price and sell the stock to the market at the future bid price.

In the following, we show the results of prediction results of stock AMZN, random forest and AdaBoost methods are used. Additionally, for multi-class problems, one vs. one and one vs. rest methods are utilized.

Prediction results of random forest one vs. one method for stock AMZN:

$$\begin{bmatrix} 109 & 27 & 0 \\ 0 & 9736 & 3 \\ 0 & 33 & 92 \end{bmatrix}$$

Prediction results of random forest one vs rest method for stock AMZN:

$$\begin{bmatrix} 109 & 27 & 0 \\ 0 & 9737 & 2 \\ 0 & 38 & 87 \end{bmatrix}$$

Prediction results of AdaBoost one vs one method for stock AMZN:

$$\begin{bmatrix} 128 & 8 & 0 \\ 2 & 9726 & 11 \\ 0 & 18 & 107 \end{bmatrix}$$

Prediction results of AdaBoost one vs rest method for stock AMZN:

$$\begin{bmatrix} 120 & 16 & 0 \\ 0 & 9736 & 3 \\ 0 & 27 & 98 \end{bmatrix}$$

Visualized confusion matrix can be found in figure 6.20. More results for all stocks are listed in appendix A.

Figure 6.21 shows the PnL for stock AMZN under random forest and AdaBoost methods. The red dots represent the bid-high cases and the blue dots represent the ask-low cases. If the dot lies above the X-axis, it means that this transaction will earn money and vice versa. The trading cost is \$0.02, and we can see that most transactions are profitable. From figure 6.22 we can find that our cumulative PnL shows a very good performance of our trading strategy. One reason to explain this is that our model returns a very good precision score. We will take action if our prediction

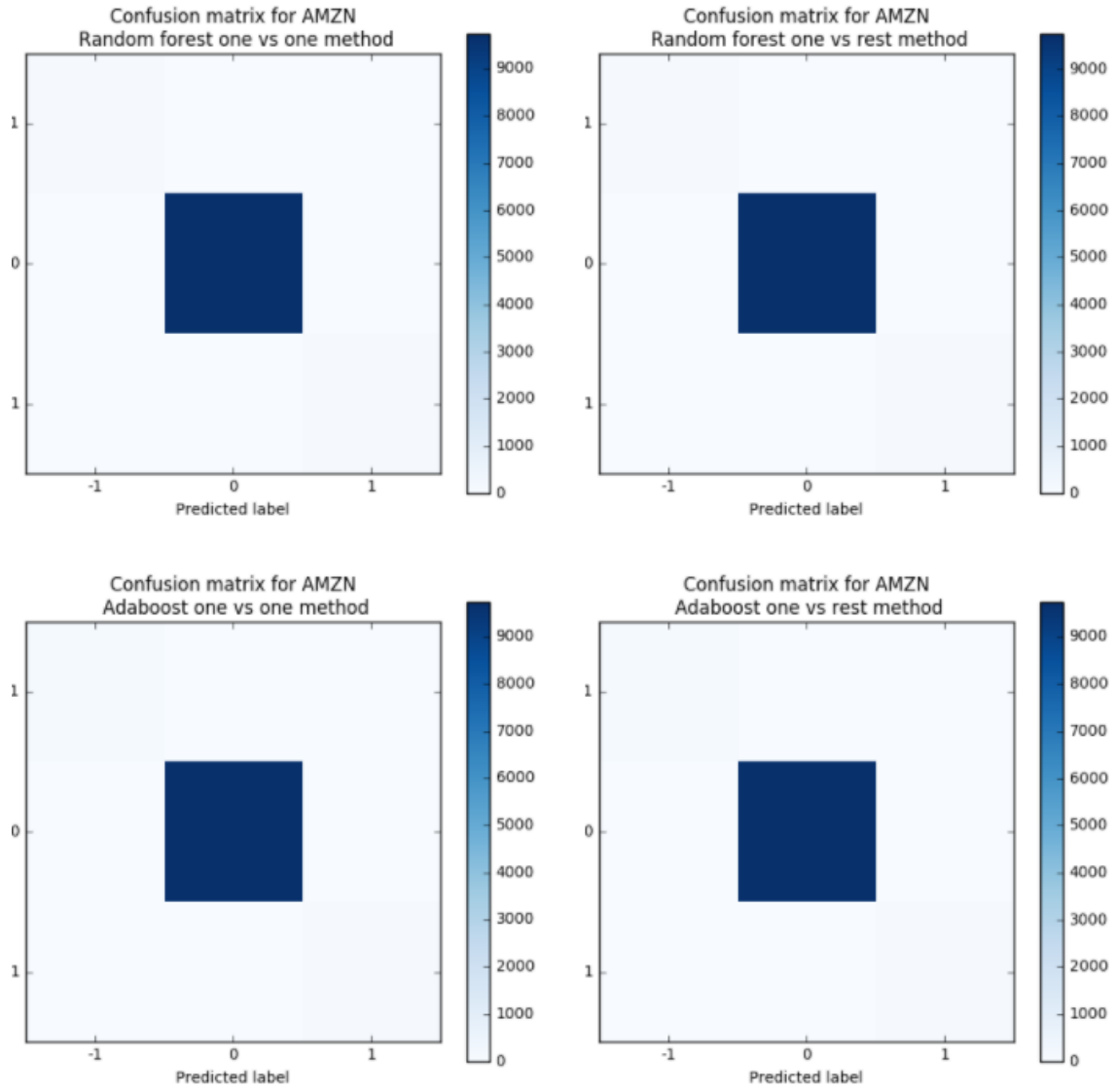


Figure 6.20: Confusion matrix of stock AMZN, -1 represents bid-high arbitrage, 1 represents ask-low arbitrage and 0 means no arbitrage. X-axis is the predicted label and Y-axis is the true label

result is non-zero, so high precision is very helpful. If our model predicts no arbitrage, then we will take no action, so misclassifying of an arbitrage case to a nonarbitrage case will not impact our profitability. Therefore, the recall score is not as important as the precision score in our strategy. The PnL results for other stocks are listed in the appendix A.

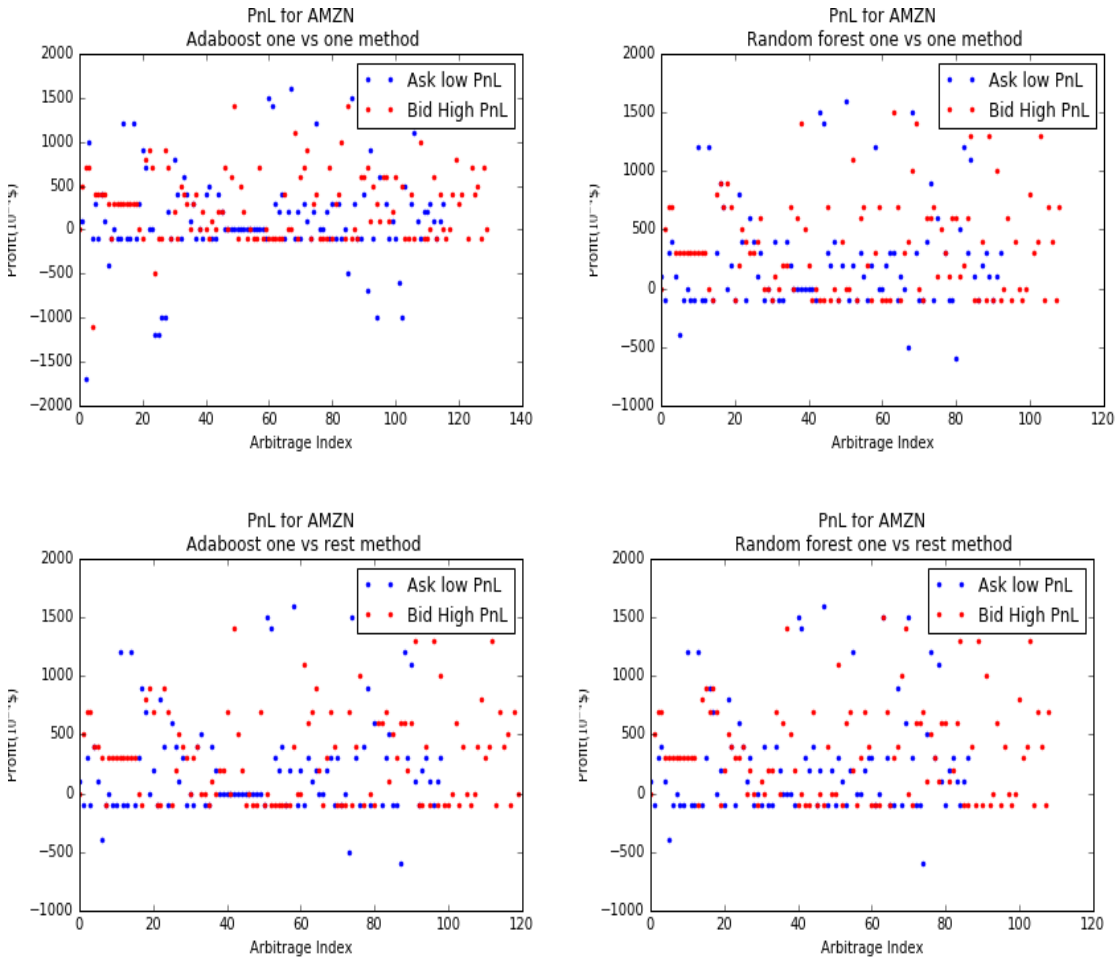


Figure 6.21: PnL for AMAN based on random forest and AdaBoost methods, X-axis represents the predicted arbitrage index and Y-axis is profit or loss for each transaction.The top left panel shows the result of AdaBoost one vs. one method, the top right panel shows the result of random forest one vs. one method, the bottom left panel shows the result of AdaBoost one vs. rest method, and the bottom right panel shows the result of random forest one vs. rest method. Trading cost is $0.02

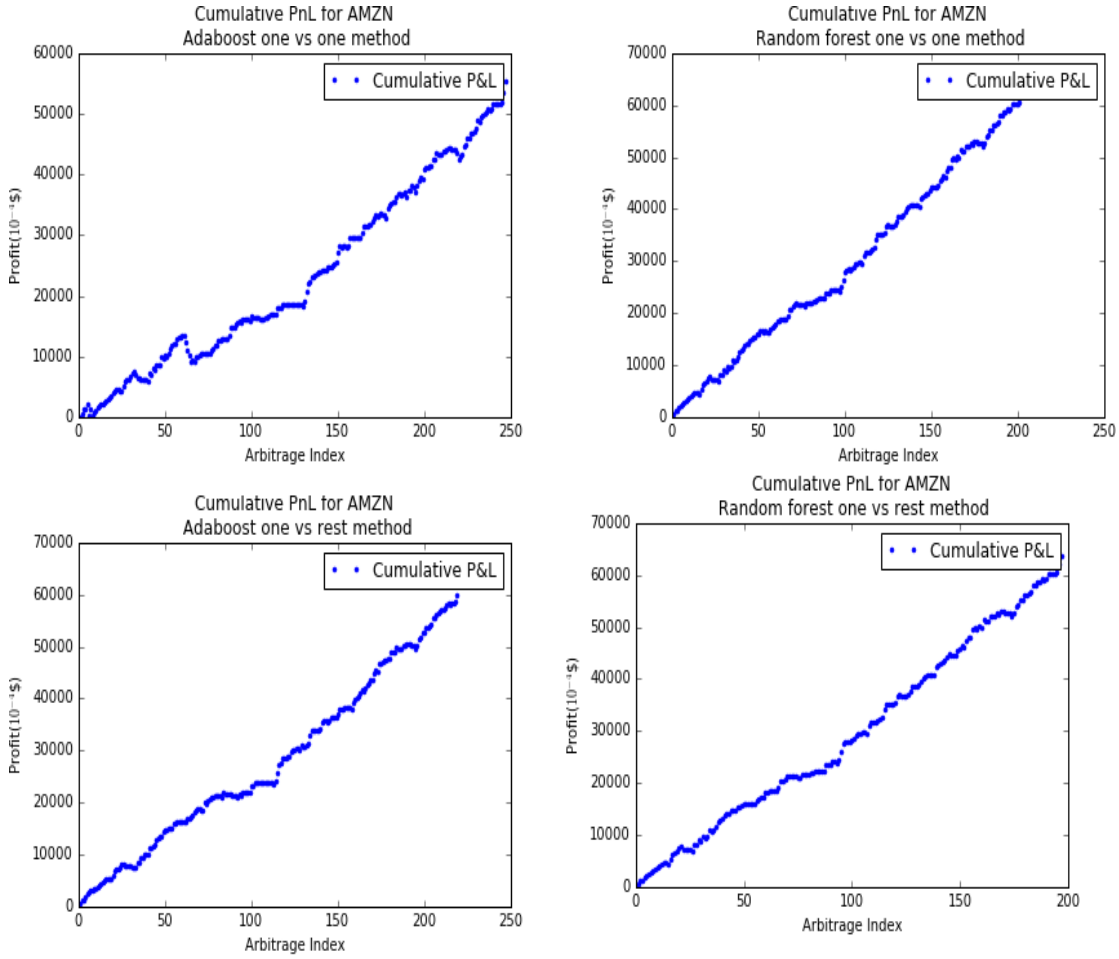


Figure 6.22: Cumulated PnL for AMAN based on random forest and AdaBoost methods, X-axis represents the predicted arbitrage index and Y-axis is profit or loss for each transaction.The top left panel shows the result of AdaBoost one vs. one method, the top right panel shows the result of random forest one vs. one method, the bottom left panel shows the result of AdaBoost one vs. rest method, and the bottom right panel shows the result of random forest one vs. rest method. Trading cost is $0.02

# CHAPTER 7

# CONCLUSION AND FUTURE WORKS

This paper designs a framework for exploring the dynamics of LOBs based on ensemble machine learning methods. With significantly better performance relative to the support vector machine, logistic regression with lasso and ridge penalty, decision tree, and ensemble methods are found to have good applicability in predicting price spread crossing opportunities of LOBs. The strong outperformance of ensemble methods over other models suggests that the existing financial markets' arbitrage strategies can be strengthened by using ensemble methods. Our models are trained and tested with substantial data samples including the total transaction histories for one trading day of five information technology stocks. The amount of data for these five stocks ranges from two hundred thousand to six hundred thousand. To deal with the imbalanced data problem, we use precision, recall, and F1 scores to measure the performance of different models. In order to deal with the multi-category classification problem, we use one vs. one and one vs. rest methods. The confusion matrices of ensemble methods based on our problem show that the framework of our research also works well for the multi-classification problem. The experimental results with real-time stock data show that the proposed frameworks based on ensemble methods are effective. In addition, a simple trading strategy that uses the predicting results of our models can make a significant profit with low risks, indicating that our model is beneficial for the actual transaction.

The framework of our research can be extended in several ways. One possible research direction is to add more features that related to changes in stock prices. For example, Wahal and Yavuz (2013) show that some variables, such as book-to-market ratio and size are helpful for predicting stock prices. Hence, there are a lot of potential alternative features for future research and how to avoid over-fitting becomes an interesting problem. Some past related issues can be found in Fan and Lv (2008), Fan et al. (2010), and Bühlmann et al. (2013).

Another possible future research orientation is to apply our models to asset allocation and portfolio management. In our paper, the PnL of a trading strategy based on our model in five seconds is profitable. We can test the PnL based on different holding periods such as one second,

ten seconds, and one minute. If our models show a profitable performance for all the holding periods, we can design a trading strategy that uses our model in different trading horizons, which may diversify risk by matching the duration of different assets, as well as obtaining considerable profit.

We can also introduce some of the latest machine learning tools in the future. For example, deep learning and reinforcement learning will be excellent candidates. Sirignano (2016) gives some demonstration of how to use deep learning for modeling the spatial distribution of LOBs. Finally, it is feasible to apply the similar framework in order to model the trends of other financial markets, such as options, futures, and currency exchanges. We can compare the profit results of our framework based on different financial markets in order to test the model's robustness.

# APPENDIX A

# TWO CLASSES ARBITRAGE PREDICTION RESULTS

In chapter 6.4, we list the result of ask-low arbitrage opportunity for stock AMZN based on Logistic regression(Lasso penalty), Logistic regression(Ridge penalty),support vector machine,decision tree, Adaboost, and Random forest. In this appendix, we show the results of the other four stocks including Apple(AAPL),Google(GOOG),Intel(INTC), and Microsoft(MSFT).

Table A.1: AAPL ask-low arbitrage opportunity prediction(5 seconds)

| Model | Training time(s) | Training F1 score | Test time(s) | Test Recall | Test Precision | Test F1 score |
|---|---|---|---|---|---|---|
| Logistic regression(Lasso penalty) | 389.8 | 9.3 % | 0.002 | 3.0% | 41.7% | 5.5% |
| Logistic regression(Ridge penalty) | 5.7 | 9.3 % | 0.01 | 3.0% | 41.7% | 5.5% |
| SVM(Poly 2 kernal,5000 estimator) | 105.7 | 68.7 % | 8.6 | 36.1% | 96.1% | 52.5% |
| Decision Tree(no pruning) | 4.3 | 50.6 % | 0.003 | 20.4% | 97.2% | 33.7% |
| Adaboost(number of estimate=100) | 32.2 | 90.1 % | 0.03 | 71.0 % | 84.5% | 77.2% |
| Random forest(number of estimate=100) | 45.9 | 99.5 % | 0.13 | 71.3 % | 98.4% | 82.7% |

Table A.2: GOOG ask-low arbitrage opportunity prediction(5 seconds)

| Model | Training time(s) | Training F1 score | Test time(s) | Test Recall | Test Precision | Test F1 score |
|---|---|---|---|---|---|---|
| Logistic regression(Lasso penalty) | 392.9 | 3.8 % | 0.002 | 4.3% | 83.3% | 8.2% |
| Logistic regression(Ridge penalty) | 5.6 | 3.8 % | 0.01 | 4.3% | 83.3% | 9.2% |
| SVM(Poly 2 kernal,5000 estimator) | 64.2 | 36.8 % | 3.3 | 20.7% | 100.0% | 34.3% |
| Decision Tree(no pruning) | 4.6 | 54.6 % | 0.003 | 50.0% | 96.7% | 65.9% |
| Adaboost(number of estimate=100) | 40.2 | 96.1 % | 0.03 | 76.7 % | 97.8 | 86.0% |
| Random forest(number of estimate=100) | 4.4 | 79.5 % | 0.01 | 77.6 % | 98.9% | 87.0% |

Table A.3: INTC ask-low arbitrage opportunity prediction(5 seconds)

| Model | Training time(s) | Training F1 score | Test time(s) | Test Recall | Test Precision | Test F1 score |
|---|---|---|---|---|---|---|
| Logistic regression(Lasso penalty) | 337.7 | 6.3 % | 0.002 | 3.2% | 1.4% | 2.0% |
| Logistic regression(Ridge penalty) | 12.1 | 62.8 % | 0.01 | 3.2% | 15.8% | 2.1% |
| SVM(Poly 2 kernal,5000 estimator) | 42.9 | 74.3 % | 3.0 | 3.2% | 10.0% | 4.8% |
| Decision Tree(no pruning) | 2.3 | 91.3 % | 0.003 | 56.8% | 69.6% | 62.5% |
| Adaboost(number of estimate=100) | 198.0 | 99.9 % | 0.20 | 85.2 % | 83.9 | 84.6% |
| Random forest(number of estimate=100) | 12.5 | 99.9 % | 0.1 | 65.6 % | 80.4% | 72.2% |

Table A.4: MSFT ask-low arbitrage opportunity prediction(5 seconds)

| Model | Training time(s) | Training F1 score | Test time(s) | Test Recall | Test Precision | Test F1 score |
|---|---|---|---|---|---|---|
| Logistic regression(Lasso penalty) | 358.6 | 39.5 % | 0.03 | 15.9% | 50.0% | 24.1% |
| Logistic regression(Ridge penalty) | 10.2 | 39.5 % | 0.002 | 15.9% | 50% | 24.1% |
| SVM(Poly 2 kernal,5000 estimator) | 88.3 | 77.8 % | 6.3 | 36.4% | 94.1% | 52.5% |
| Decision Tree(no pruning) | 1.9 | 88.1 % | 0.004 | 52.2% | 15.9% | 24.3% |
| Adaboost(number of estimate=100) | 132.3 | 99.9 % | 0.18 | 59.1 % | 100.0 | 74.28% |
| Random forest(number of estimate=100) | 16.4 | 99.9 % | 0.1 | 59.1 % | 100.0% | 74.28% |

# APPENDIX B

# PNL AND CUMULATIVE PNL

Figure B.1 and figure B.2 show the profit and loss results for four stocks, the trading cost here is set to 0. Methodology is random forest one vs. rest algorithm

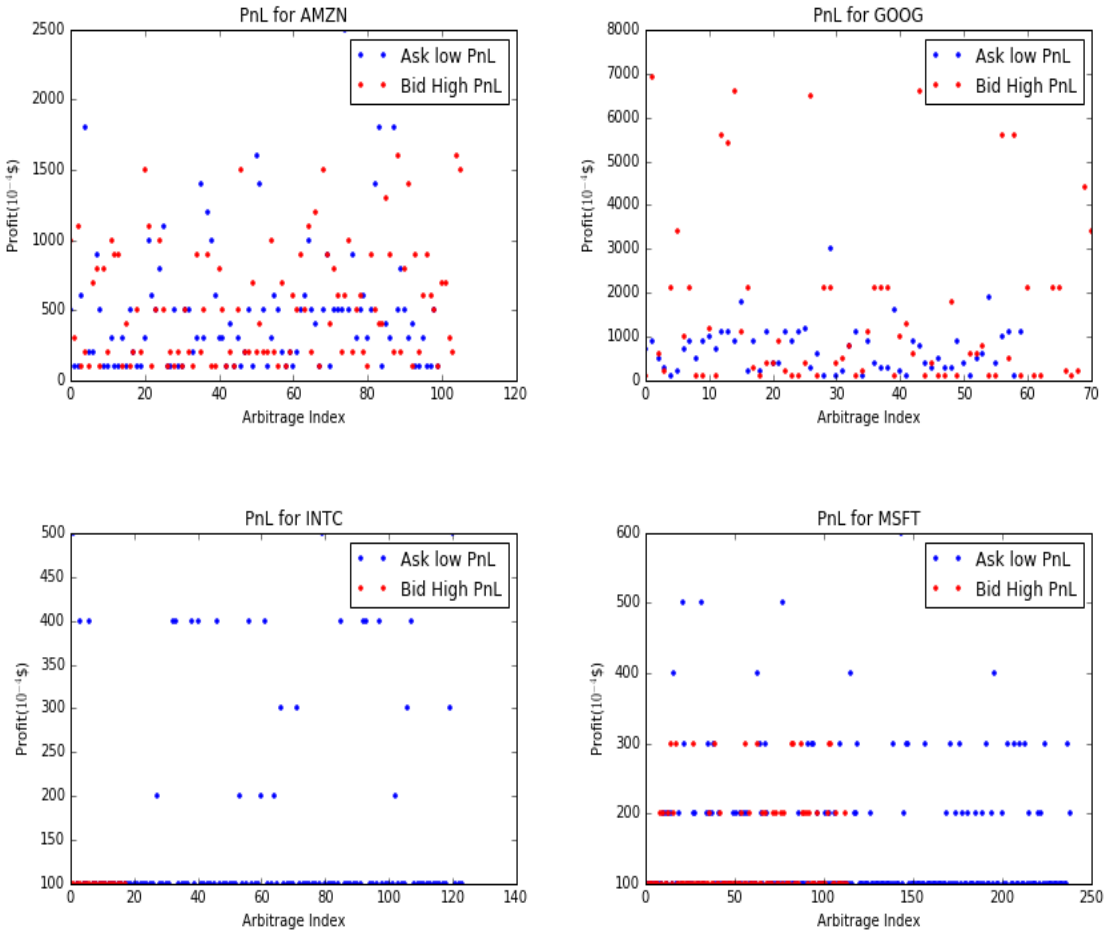


Figure B.1: Profit and Loss, X-axis represents the predicted arbitrage index and Y-axis is profit or loss for each transaction.The top left panel shows the result of AMZN, the top right panel shows the result of GOOG,the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

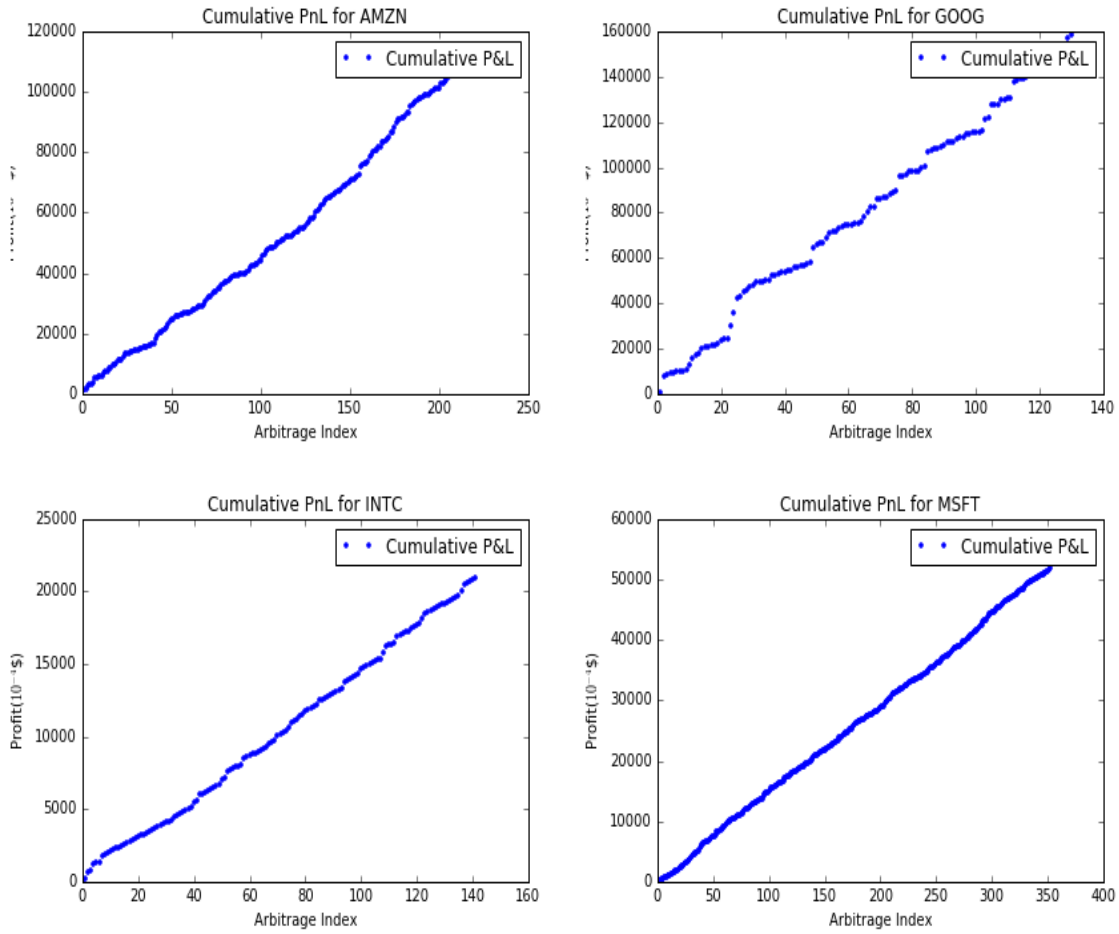


Figure B.2: Profit and Loss, X-axis represents the predicted arbitrage index and Y-axis is profit or loss for each transaction.The top left panel shows the result of AMZN, the top right panel shows the result of GOOG,the bottom left panel shows the result of INTC, and the bottom right panel shows the result of MSFT.

# APPENDIX C

# MAIN PART OF CODE

Our models are established based on language Python. In this appendix, we list the main part of our codes, which includes seven components:

1) Model preparation: Read order books and message books data of five stocks into python environment. Build features and responses based on different time interval.

2) Data split: Divide the features and responses data into training data and testing data based on the ratio of nine to one.

3) Two-class problem: Create binary classification models( predicting the future ask-low case of LOBs), which include logistic regression, lasso regression, ridge regression, support vector machine, decision tree, random forest, and AdaBoost.

4) Multi-class problem: Create binary classification models, which include logistic regression, lasso regression, ridge regression, support vector machine, decision tree, random forest, and AdaBoost, by use of one vs. one and one vs. rest methods.

5) PnL calculation: Design a simple trading strategy and calculate the PnL based on ensemble predicting models.

6) Order book structure plotting: Plot the structures of LOBs which include volume, size, snapshot and relative depth of LOBs.

7) Statistical properties of LOBs plotting: Plot the statistical properties of LOBs which contain distribution, average shape, intraday seasonality of LOBs.

## Limit order book

author: Jian Wang

time: 2017-06-19

## Model prepare

In [ ]:

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from sklearn import linear_model
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import tree
from sklearn import ensemble
import time
import matplotlib.pyplot as plt

#Set default parameters
ticker_list=["AAPL","AMZN","GOOG","INTC","MSFT"]
start_ind=10*3600
end_ind=15.5*3600
data_order_list=[]
data_mess_list=[]
time_index_list=[]
path_save='/media/jianwang/Study1/Research/order_book/'
path_load="/media/jianwang/Study1/Research/order_book/"

## set random seed to produce the same results

np.random.seed(987612345)

#read the stock ticker
#totally 5 dataset

for i in range(len(ticker_list)):
    #get the path for the csv files
    # name_order is for the order book and name_mess for the message book
    name_order='_2012-06-21_34200000_57600000_orderbook_10.csv'
    name_mess='_2012-06-21_34200000_57600000_message_10.csv'
    # calculate the cputime for reading the data
    t=time.time()
    # header =-1 means that the first line is not the header, otherwise, the first line will be header
    # data_order is for order book and data mess is for message book
    data_order_list.append(np.array(pd.read_csv(path_load+ticker_list[i]+name_order,header=-1),dtype="f
loat64"))
    data_mess_list.append(np.array(pd.read_csv(path_load+ticker_list[i]+name_mess,header=-1),dtype="flo
at64"))
    print("Time for importing the "+ticker_list[i]+" data is:",time.time()-t)
    print("The shape of the order data is: ",data_order_list[i].shape, " of message data is: ", data_me
ss_list[i].shape)
    # get the time index
    time_index_list.append(data_mess__list[i][:,0])


#print the sample of data
print("Check the original data:")

for i in range(len(ticker_list)):
    print()
    print("The first five sampe of "+ticker_list[i]+" is: ",data_order_list[i][:3])


# load the feature
#%%
```

```
import time
t=time.time()
feature_array_list=[]
for ticker_ind in range(len(ticker_list)):
    feature_array_list.append(np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_feature_array.tx
t',\
                                                   sep=' ',header=-1)))
print(time.time()-t)

# this function used to build the y
# ask_low as 1 bad high as -1 and no arbitrage as 0
# option=1 return ask low, option =2 return bid high, option =3 return no arbi, option =4 return total(
ask_low=1,
# bid_high =-1 and no arbi =0)
#%%
def build_y(ask_low,bid_high,no_arbi,option):
    if (option==1):
        return ask_low
    elif option==2:
        return bid_high
    elif option==3:
        return no_arbi
    elif option==4:
        return ask_low-bid_high
    else:
        print("option should be 1,2,3,4")


## load y data
#%%
response_list=[]
for ticker_ind in range(len(ticker_list)):
    response_list.append((np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_response.txt',header
=-1))))


## print the shape of the response
## note it is the total response
#%%
print("The shape of the total response is:\n")

for ticker_ind in range(len(ticker_list)):
    print(response_list[ticker_ind].shape)

# need to get the response from 10 to 15:30
# the shape of the response and the feature array should be equal
response_reduced_list=[]
for ticker_ind in range(len(ticker_list)):
    first_ind = np.where(time_index_list[ticker_ind]>=start_ind)[0][0]
    last_ind=np.where(time_index_list[ticker_ind]<=end_ind)[0][-1]
    response_reduced_list.append(response_list[ticker_ind][first_ind:last_ind+1])

print("The shape of the reduced response is:\n")

## print the shape of reduced response
## response reduced is used for testing and training the model
for ticker_ind in range(len(ticker_list)):
    print(response_reduced_list[ticker_ind].shape)
```

## Data split

In [ ]:

```
ticker_ind=1
size=100000
random_ratio=0.5
# combine the feature and response array to random sample
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind]),axis=1)[:
size,:]

total_array=total_array[random_choice(list(range(size)),int(size*random_ratio)),:]


train_num_index=int(len(total_array)*0.9)
```

```
print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
# scale data
#%%

# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))

# -*- coding: utf-8 -*-

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)
```

## Two classes training

Logistic regression, lasso regression, ridge regression, support vector machine, decision tree, random forest and Adaboosting are used here.

In [ ]:

```
#----------------
# logistic l1
#-----------------


from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l1', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
```

```
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()

#----------------
# logistic l2
#-----------------


from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)
```

```
        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l2', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

## Multi-class predict

One vs One and One vs rest are used here

### load the arbitrage time txt data

In [ ]:

```
ask_low_time_list=[]
bid_high_time_list=[]
no_arbi_time_list=[]
time_list=[1,5,10,15,20]
import time
t=time.time()
for ticker_ind in range(5):
    ask_low_time_list.append([])
    bid_high_time_list.append([])
    no_arbi_time_list.append([])
    for time_ind in range(len(time_list)):
        ask_low_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_ask_low_time_'+str(time_list[time_
ind])+'.txt',header=-1)))
        bid_high_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_bid_high_time_'+str(time_list[time
_ind])+'.txt',header=-1)))
        no_arbi_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_no_arbi_time_'+str(time_list[time_
ind])+'.txt',header=-1)))

print(time.time()-t)
```

### Deal with the data

In [ ]:

```
def build_y(ask_low,bid_high,no_arbi,option):
    if (option==1):
        return ask_low
    elif option==2:
        return bid_high
    elif option==3:
        return no_arbi
    elif option==4:
        return ask_low-bid_high
    else:
        print("option should be 1,2,3,4")

for ticker_ind in range(len(ticker_list)):
    response=build_y(ask_low_time_list[ticker_ind][1],bid_high_time_list[ticker_ind][1],\
                                 no_arbi_time_list[ticker_ind][1],option=4)
    np.savetxt(path_save+ticker_list[ticker_ind]+'_multiresponse.txt',response)

response_list=[]
for ticker_ind in range(len(ticker_list)):
    response_list.append((np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_multiresponse.txt',h
eader=-1))))

    ## print the shape of the response
## note it is the total response
print("The shape of the total response is:\n")

for ticker_ind in range(len(ticker_list)):
    print(response_list[ticker_ind].shape)

# need to get the response from 10 to 15:30
# the shape of the response and the feature array should be equal
response_reduced_list=[]
for ticker_ind in range(len(ticker_list)):
    first_ind = np.where(time_index_list[ticker_ind]>=start_ind)[0][0]
    last_ind=np.where(time_index_list[ticker_ind]<=end_ind)[0][-1]
    response_reduced_list.append(response_list[ticker_ind][first_ind:last_ind+1])

print("The shape of the reduced response is:\n")

## print the shape of reduced response
## response reduced is used for testing and training the model
for ticker_ind in range(len(ticker_list)):
    print(response_reduced_list[ticker_ind].shape)
```

```
    # random split data
```

### Split the data

In [ ]:

```
#time series split
#%%-----------------------------------------------------------------------------------------------

ticker_ind=0
size =100000
random_ratio=0.6

time_index=time_index_list[ticker_ind]
# combine the feature and response array to random sample
time_index_reduced=time_index[(time_index>=start_ind)&(time_index<=end_ind)]
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                            time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)[:size,:]

total_array=total_array[random_choice(list(range(size)),int(size*random_ratio)),:]


train_num_index=int(len(total_array)*0.9)

print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
# scale the data
# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))
```

### One vs One

In [ ]:

```
# only run for random forest method
# one vs one case
# random forest
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier
from sklearn.ensemble import RandomForestClassifier


## sample weights
#sample_weights=[]
#ratio=len(train_y)/sum(train_y==1)/10
#for i in range(len(train_x)):
#    if train_y[i]==0:
#        sample_weights.append(1)
#    else: sample_weights.append(ratio)
```

```
# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsOneClassifier(RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345)
)
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)

print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

%matplotlib inline
# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_one.png")
plt.show()
```

### One Vs rest

In [ ]:

```
# only run for random forest method
# one vs rest case
```

```
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier




# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsRestClassifier(RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345
))
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to pbrefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)
print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_rest.png")
plt.show()
```

## PnL Calculation

In [ ]:

```
def get_index(index, value):
```

```
    i=0
    while index[i] <value:
        i=i+1
    return i
## for AMZN
ticker_ind =1
train_ratio=0.9
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]

time_index=data_mess[:,0]
data_order_reduced=data_order[(time_index>= start_ind) & (time_index<= end_ind)]
time_index_reduced=time_index[(time_index>= start_ind) & (time_index<= end_ind)]
total_array_old=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                                time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)
import matplotlib.pyplot as plt

plt.plot(time_index_test[:10000],data_order_test[:10000,0],"r-",label="Ask price")
plt.plot(time_index_test[:10000],data_order_test[:10000,2],"b-",label="Bid price")

x_ask_low_choose=time_index_test[test_y_unrandom==1]
y_ask_low_choose=data_order_test[test_y_unrandom==1,0]
x_bid_high_choose=time_index_test[test_y_unrandom==-1]
y_bid_high_choose=data_order_test[test_y_unrandom==-1,2]

plt.plot(x_ask_low_choose[:30],y_ask_low_choose[:30],"gv",markersize=8,label="Ask low")
plt.plot(x_bid_high_choose[:30],y_bid_high_choose[:30],"r^",markersize=8,label="Bid high")
plt.xlabel("Time(s)")
plt.ylabel("Price($10^{-4}$\$)")
plt.legend(bbox_to_anchor=[1.4, 1])
plt.title("Arbitrage opportunities for "+ticker_list[ticker_ind]+"(5s)")
plt.savefig("arbitrage_plot.png")
plt.show()
time_index_test=total_array[:,135][int(size*train_ratio):size]
# find the arbitrage occuring index
arbi_index=list(np.where(predict_y_test!=0)[0])
# find the index that 5 seconds later
arbi_future_index=[]
for i in arbi_index:
    arbi_future_index.append(get_index(time_index_reduced,time_index_test[i]+5))

total_array_test=total_array[int(size*train_ratio):size,:]
future_price=[]
current_price=[]
pnl=[]
cost=200
for i in range(len(arbi_index)):
    #ask low
    if predict_y_test[arbi_index[i]]==1 :
        future_price=data_order_reduced[arbi_future_index[i],0]
        current_price=total_array_test[arbi_index[i],2]
        pnl.append(current_price-future_price-cost)
    # bid high
    else:
        future_price=data_order_reduced[arbi_future_index[i],2]
        current_price=total_array_test[arbi_index[i],0]
        pnl.append(future_price-current_price-cost)

pnl=np.array(pnl)
predict_arbi=predict_y_test[predict_y_test!=0]
plt.plot(pnl[predict_arbi==1],"b.",label="Ask low PnL")
plt.plot(pnl[predict_arbi==-1],"r.",label="Bid High PnL")

plt.xlabel("Arbitrage Index")
plt.ylabel("Profit($10^{-4}$\$)")
plt.title("PnL for "+ticker_list[ticker_ind])
plt.legend()
plt.savefig(ticker_list[ticker_ind]+"_pnl.png")
plt.show()

cum_pnl=np.cumsum(pnl)
plt.plot(cum_pnl,"b.",label="Cumulative P&L")
plt.xlabel("Arbitrage Index")
plt.ylabel("Profit($10^{-4}$\$)")
plt.title("Cumulative PnL for "+ticker_list[ticker_ind])
plt.legend()
plt.savefig(ticker_list[ticker_ind]+"_cum_pnl.png")
```

```
plt.show()
```

## Order book plot

In [ ]:

```
# fun for total

#---------------------------------------------------------------------------
#%%  set the parameters
#Stock name
#---------------------------------------------------------------------------
ticker ="INTC"


#---------------------------------------------------------------------------
# Levels
#---------------------------------------------------------------------------
lvl= 10

#---------------------------------------------------------------------------
# File names
#---------------------------------------------------------------------------
path='/media/jianwang/Study1/Research/order_book/'
path_save='/media/jianwang/Study1/Research/order_book/'
path_save='/media/jianwang/Study1/Research/order_book/'
name_book    = 'AMZN_2012-06-21_34200000_57600000_orderbook_10.csv'
name_mess    = 'AMZN_2012-06-21_34200000_57600000_message_10.csv'

#---------------------------------------------------------------------------
# Date of files
#---------------------------------------------------------------------------
demo_date    = [2012,6,21]    #year, month, day

#---------------------------------------------------------------------------
# Load Messsage File
#---------------------------------------------------------------------------
#  Load data
t=time.time()
mess = np.array(pd.read_csv(path+name_mess))
print("The time for reading the CSV file",time.time()-t)
#
#
#% Message file information:
#% -------------------------------------------------------
#%
#%   - Dimension:    (NumberEvents x 6)
#%
#%   - Structure:    Each row:
#%                   Time stamp (sec after midnight with decimal
#%                   precision of at least milliseconds and
#%                   up to nanoseconds depending on the period),
#%                   Event type, Order ID, Size (# of shares),
#%                   Price, Direction
#%
#%                   Event types:
#%                       - '1'   Submission new limit order
#%                       - '2'   Cancellation (partial)
#%                       - '3'   Deletion (total order)
#%                       - '4'   Execution of a visible limit order
#%                       - '5'   Execution of a hidden limit order
#%                               liquidity
#%                       - '7'   Trading Halt (Detailed
#%                               information below)
#%
#%                   Direction:
#%                       - '-1'  Sell limit order
#%                       - '-2'  Buy limit order
#%                       - NOTE: Execution of a sell (buy)
#%                               limit order corresponds to
#%                               a buyer-(seller-) initiated
#%                               trade, i.e. a BUY (SELL) trade.
#%
#% -------------------------------------------------------
#% Data Preparation - Message File
```

```
#
#% Trading hours (start & end)

#%% deal with the message data
#Remove observations outside the official trading hours
# -------------------------------------------------------

#% Trading hours (start & end)
start_trad   = 9.5*60*60       # 9:30:00 in sec
                               # after midnight
end_trad     = 16*60*60        # 16:00:00 in sec
                               # after midnight
# Get index of observations
time_idx=(mess[:,0]>= start_trad) & (mess[:,0]<= end_trad)
mess = mess[time_idx,:]


##-------------------------------------------------------------
#% Note: As the rows of the message and orderbook file
#%       correspond to each other, the time index of
#%       the message file can also be used to 'cut'
#%       the orderbook file.
#
#
#% Check for trading halts
#% -------------------------------------------------------------
trade_halt_idx = np.where(mess[:,1] == 7)

if (np.size(trade_halt_idx)>0):
    print(['Data contains trading halt! Trading halt, '+
    'quoting resume, and resume of trading indices in tradeHaltIdx'])
else:
    print('No trading halts detected.')
#
#
#%  When trading halts, a message of type '7' is written into the
#%  'message' file. The corresponding price and trade direction
#%  are set to '-1' and all other properties are set to '0'.
#%  Should the resume of quoting be indicated by an additional
#%  message in NASDAQ's Historical TotalView-ITCH files, another
#%  message of type '7' with price '0' is added to the 'message'
#%  file. Again, the trade direction is set to '-1' and all other
#%  fields are set to '0'.
#%  When trading resumes a message of type '7' and
#%  price '1' (Trade direction '-1' and all other
#%  entries '0') is written to the 'message' file. For messages
#%  of type '7', the corresponding order book rows contain a
#%  duplication of the preceding order book state. The reason
#%  for the trading halt is not included in the output.
#%
#%    Example: Stylized trading halt messages in 'message' file.
#%
#%    Halt:     36023 | 7 | 0 | 0 | -1 | -1
#%              ...
#%    Quoting:    36323  | 7 | 0 | 0 | 0  | -1
#%              ...
#%    Resume Trading:  36723   | 7 | 0 | 0 | 1  | -1
#%              ...
#%    The vertical bars indicate the different columns in the
#%    message file.
#
#% Set Bounds for Intraday Intervals
#
#% Define interval length

freq = 6.5*3600/(5*60)+1  # Interval length in sec, according to the python do not include the endpoint
                          # so add 1 in the last

time_interval=60*6.5/(freq-1)

# Set interval bounds
bounds = np.linspace(start_trad,end_trad,freq,endpoint=True)

# Number of intervals
bl = np.size(bounds,0)

# Indices for intervals
```

```
bound_idx = np.zeros([bl,1])



k1 = 0
for k2 in range(0,np.size(mess,0)):
    if mess[k2,0] >= bounds[k1]:
        bound_idx[k1,0] = k2
        k1 = k1+1
bound_idx[bl-1]=mess[len(mess)-1,0]


#
#% Plot - Number of Executions and Trade Volume by Interval
#
#% Note: Difference between trades and executions
#%
#%       The LOBSTER output records limit order executions
#%       and not what one might intuitively consider trades.
#%
#%       Imagine a volume of 1000 is posted at the best ask
#%       price. Further, an incoming market buy order of
#%       volume 1000 is executed against the quote.
#%
#%       The LOBSTER output of this trade depends on the
#%       composition of the volume at the best ask price.
#%       Take the following two scenarios with the best ask
#%       volume consisting of ...
#%       (a) 1 sell limit order with volume 1000
#%       (b) 5 sell limit orders with volume 200 each
#%           (ordered according to time of submission)
#%
#%       The LOBSTER output for case ...
#%       (a) shows one execution of volume 1000. If the
#%           incoming market order is matched with one
#%           standing limit order, execution and trade
#%           coincide.
#%       (b) shows 5 executions of volume 200 each with the
#%           same time stamp. The incoming order is matched
#%           with 5 standing limit orders and triggers 5
#%           executions.
#%
#%       Bottom line:
#%       LOBSTER records the exact limit orders against
#%       which incoming market orders are executed. What
#%       might be called 'economic' trade size has to be
#%       inferred from the executions.

#% Collection matrix
trades_info = np.zeros([bl-1,4])
#    % Note: Number visible executions, volume visible
#    %       trades, number hidden executions,
#    %       volume hidden trades


for k1 in range(0,bl-1):


    temp = mess[int(bound_idx[k1]+1):int(bound_idx[k1+1]),[1,3]]

    temp_vis = temp[temp[:,0]==4,1]    # Visible


    #% Hidden
    temp_hid = temp[temp[:,0]==5,1];

    # Collect information
    trades_info[k1,:] = [np.size(temp_vis,0), np.sum(temp_vis),np.size(temp_hid,0), np.sum(temp_hid)]

    del temp, temp_vis, temp_hid

#%% plot the data
#Plot number of executions
#-------------------------------------------------------------------------------

%matplotlib inline
fig, ax = plt.subplots()
```

```
ind=np.arange(np.size(trades_info,0))
width=1
color=["red","blue"]4
   #% Visible ...
ax.bar(ind,trades_info[:,0],width=width, color=color[0],label="Visible",alpha=0.7)
#        title({[ticker ' // ' ...
#            datestr(datenum(demoDate),'yyyy-mmm-dd')] ...
#            ['Number of Executions per ' ...
#            num2str(freq./60) ' min Interval ']});
ax.set_xlabel('Interval')
ax.set_ylabel('Number of Executions')
ax.set_title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nNumber of Executions per "+str(time_interval)+" minutes interval")
ax.bar(ind,-trades_info[:,2],width=width,color=color[1],label="Hidden");
ax.legend(loc="upper center")
plt.savefig(ticker+"_num_exec.png")

#-------------------------------------------------------------------------
#plot the volume of traders
#-------------------------------------------------------------------------
fig, ax = plt.subplots()
ind=np.arange(np.size(trades_info,0))
width=1
color=["red","blue"]
   #% Visible ...
ax.bar(ind,trades_info[:,1]/100,width=width, color=color[0],label="visible",alpha=0.7)

ax.set_xlabel('Interval')
ax.set_ylabel('Number of Trades Trades (X100 shares)')
ax.set_title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nVolume of trades per "+str(time_interval)+" minutes interval")
ax.bar(ind,-trades_info[:,3]/100,width=width,color=color[1],label="Hidden");
ax.legend(loc="upper center")
plt.savefig(ticker+"_num_trade.png")
plt.show()

t=time.time()
book = np.array(pd.read_csv(path+name_book,dtype ="float64"))
print("The time for reading the CSV file",time.time()-t)
book = book[time_idx,:]
book[:,::2]=book[:,::2]/10000

#%% plot the snapshot of the limit order book
#-------------------------------------------------------------------------
#select a random event to show
event_idx= np.random.randint(0, len(book))# note that the randint will not generate the last value

ask_price_pos=list(range(0,lvl*4,4))

# Note: Pick a randmom row/ event from the order book.
# position of variables in the book

ask_price_pos = list(range(0,lvl*4,4))

ask_vol_pos= [i+1 for i in ask_price_pos]

bid_price_pos=[i+2 for i in ask_price_pos]

bid_vol_pos=[i+1 for i in bid_price_pos]

vol= list(range(1,lvl*4,2))

max_price = book[event_idx, ask_price_pos[lvl-1]]+0.01
min_price=book[event_idx,bid_price_pos[lvl-1]]-0.01

max_vol=max(book[event_idx,vol])

mid=0.5*(sum(book[event_idx,[0,2]],2))

#%%plot the Snapshot of the Limit Order Book
#-------------------------------------------------------------------------
plt.figure()
#ask price
color=["red","blue"]
y_pos=np.arange(11,21)
y_value=book[event_idx,ask_vol_pos]
plt.barh(y_pos, y_value,alpha=0.7,color=color[0],align="center",label="Ask")
```

```
#mid price
plt.plot([10,40],[10,10],'<g',markersize=10,fillstyle="full",label="Mid_price")
#bid price
y_pos=np.arange(0,10)
y_value=book[event_idx,bid_vol_pos][::-1]
plt.barh(y_pos,y_value,alpha=0.7,color=color[1],align="center",label="Bid")
#set style
y_pos=np.arange(0,21)
y_ticks=np.concatenate((book[event_idx,bid_price_pos][::-1],np.array([mid]),book[event_idx,ask_price_po
s]),0)
plt.yticks(y_pos,y_ticks)
plt.xlabel('Volumne')
plt.title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nLOB Snapshot -Time: "+str(mess[event_idx,0])+" Seconds")
plt.ylim([-1,21])
plt.legend()
plt.savefig(ticker+"_snapshot.png")

plt.show()

#%%plot the relative depth in the Limit Oeder Book
#------------------------------------------------------------------------------



#% Relative volume ...

#% Ask
book_vol_ask = np.cumsum(book[event_idx,ask_vol_pos])
book_vol_ask = book_vol_ask/book_vol_ask[-1]

#% Bid
book_vol_bid = np.cumsum(book[event_idx,bid_vol_pos])
book_vol_bid = book_vol_bid/book_vol_bid[-1]

plt.figure()
#% Ask
plt.step(list(range(1,11)),book_vol_ask,color="g",label="Ask Depth")

plt.title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nLOB Relative Depth -Time: "+str(mess[event_idx,0])+" Seconds")

plt.ylabel('% of Volume')
plt.xlabel('Level')

plt.xlim([1,10])

#Bid
plt.step(list(range(1,11)),-book_vol_ask,color="r",label="Bid Depth")

#y_pos=np.arange(0,21)
y_pos=np.linspace(-1,1,11)
plt.yticks(y_pos,[1,0.8,0.6,0.4,0.2,0,0.2,0.4,0.6,0.8,1])
plt.ylim([-1,1])
plt.savefig(ticker+"_depth.png")

plt.show()
```

## Statistical properties plotting

In [ ]:

```
### 1)Cumulative distribution function for arrival time¶
```

In [ ]:

```
ticker_ind=2
data=data_mess_list[ticker_ind]
# we use the market order
data_order=data[(data[:,1]==4) | (data[:,1]==5)]

arrival_time=data_order[1:,0]-data_order[0:-1,0]
#delete the zero intra arrival time
arrival_time=arrival_time[arrival_time>0]
```

```
mu_log=np.mean(np.log(arrival_time))
std_log=np.std(np.log(arrival_time))
data_log=np.random.lognormal(mu_log,std_log,arrival_time.shape)

mu_exp=np.mean(arrival_time)
data_exp=np.random.exponential(mu_exp,arrival_time.shape)

data_weibull=np.random.weibull(0.38,arrival_time.shape)
beta=np.var(arrival_time)/np.mean(arrival_time)
alpha=np.mean(arrival_time)/beta
data_gamma=np.random.gamma(alpha,beta,arrival_time.shape)
```

In [ ]:

```
%matplotlib inline
import statsmodels.api as sm
from scipy.stats.kde import gaussian_kde

from scipy.interpolate import UnivariateSpline
from scipy.stats import lognorm
ecdf = sm.distributions.ECDF(arrival_time,)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"b",label="Original data")

ecdf = sm.distributions.ECDF(data_log)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"g",label="Lognormal Distribution")

ecdf = sm.distributions.ECDF(data_exp)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"y",label="Exponential distribution")

ecdf = sm.distributions.ECDF(data_weibull)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"r",label="Weibull distribution")


ecdf = sm.distributions.ECDF(data_gamma)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"purple",label="Gamma distribution")


plt.xlabel("Intra-arrival time")
plt.ylabel("Probability")
plt.legend(loc="lower right")
plt.title("Cumulative distribution function of order arrival time")
plt.show()
```

### 2) Volume plotting

In [ ]:

```
%matplotlib inline
from scipy.interpolate import UnivariateSpline
from scipy.stats import lognorm
import seaborn as sns
ticker_ind=0
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]

data_market=data[(data[:,1]==4) | (data[:,1]==5)]
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order
Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")

plt.xlim([0,5])
plt.ylim([0,1])
```

```
plt.ylim([0,1])
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_AAPL.png")
plt.show()
```

### 3) Intraday seasonality

In [ ]:

```
ticker_ind=0
data_mess=data_mess_list[ticker_ind]
data_mess_limit=data_mess[data_mess[:,1]==1,:]

# calute the volume of limit order book in each time interval

time_interval=np.linspace(data_mess_limit[:,0].min(),data_mess_limit[:,0].max(),78)
vol=0
vol_time=[]
j=1

for i in range(len(data_mess_limit)):
    if  data_mess_limit[i,0]<=time_interval[j]:
        vol=vol+data_mess_limit[i,3]
    else:
        j=j+1
        vol_time.append(vol)
        vol=data_mess_limit[i,3]

# plot the quadratic fit and vol_time
x=range(76)
plt.plot(x,vol_time,label=ticker_list[ticker_ind])
qua_fit=np.poly1d(np.polyfit(x, vol_time, 2))(x)
plt.plot(x,qua_fit,label=ticker_list[ticker_ind]+" quadratic fit")
plt.legend(loc="lower right")
xticks=np.arange(34200,57600,2400)
plt.xticks(x[::8],xticks)
plt.show()
```

### 4) average shape of order books

In [ ]:

```
### 4) average shape of the order books
%matplotlib inline
import seaborn as sns

ticker_ind=1
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]
data_order_limit_ask_vol=data_order[data_mess[:,1]==1,1:40:4]
data_order_limit_bid_vol=data_order[data_mess[:,1]==1,3:40:4]

vol_ask=np.sum(data_order_limit_ask_vol,axis=0)/np.mean(np.sum(data_order_limit_ask_vol,axis=0))
vol_bid=np.sum(data_order_limit_bid_vol,axis=0)/np.mean(np.sum(data_order_limit_bid_vol,axis=0))
plt.plot(list(range(-10,0)),vol_bid)
plt.plot(list(range(1,11)),vol_ask)
```

In [ ]:

```
### 5) placement of orders
ticker_ind=2
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]

data_mess_limit=data_mess[data_mess[:,1]==1,:]
data_order_limit=data_order[data_mess[:,1]==1,:]
spread_list=[]
for i in range(1,len(data_mess_limit)):
    if data_mess_limit[i,5]==-1:
        spread=data_mess_limit[i,4]-data_order_limit[i-1,0]
    else:
        spread=data_order_limit[i-1,2]-data_mess_limit[i,4]
```

```
        spread=data_order_limit[i-1,2]-data_mess_limit[i,4]
    spread_list.append(spread)
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
import math

Se_u=pd.Series(np.array(spread_list))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")
mu = 0
variance = np.var(spread_list)
sigma = math.sqrt(variance)
x = np.linspace(min(spread_list), max(spread_list), 100)
plt.plot(x,mlab.normpdf(x, mu, sigma),"r--",label="Gaussian")
plt.xlim([-10000,10000])
```

# BIBLIOGRAPHY


Anuj Agarwal. High frequency trading: Evolution and the future. 2012.

Bruno Biais, Pierre Hillion, and Chester Spatt. An empirical analysis of the limit order book and the order flow in the paris bourse. *the Journal of Finance*, 50(5):1655–1689, 1995.

Chester I Bliss. The method of probits. *Science*, 79(2037):38–39, 1934.

Ekkehart Boehmer, Robert Jennings, and Li Wei. Public disclosure and private decisions: Equity market execution quality and order routing. *Review of Financial Studies*, 20(2):315–358, 2007.

Peter J Bolland and Jerome T Connor. A constrained neural network kalman filter for price estimation in high frequency financial data. *International Journal of Neural Systems*, 8(04): 399–415, 1997.

PJ Bolland and JT Connor. A robust non-linear multivariate kalman filter for arbitrage identification in high frequency data. *Neural Networks in Financial Engineering (Proceedings of the NNCM-95), eds. AP. N. Refenes, Y. Abu-Mostafa, J. Moody and AS Weigend (World Scientific)*, pages 122–135, 1996.

Tim Bollerslev and Ian Domowitz. Trading patterns and prices in the interbank foreign exchange market. *The Journal of Finance*, 48(4):1421–1443, 1993.

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

Jean-Philippe Bouchaud, Marc Mézard, Marc Potters, et al. Statistical properties of stock order books: empirical results and models. *Quantitative finance*, 2(4):251–256, 2002.

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

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

Peter Bühlmann. Bagging, boosting and ensemble methods. In *Handbook of Computational Statistics*, pages 985–1022. Springer, 2012.

Peter Bühlmann, Philipp Rütimann, Sara van de Geer, and Cun-Hui Zhang. Correlated variables in regression: clustering and sparse estimation. *Journal of Statistical Planning and Inference*, 143(11):1835–1858, 2013.

Jan Bulla. Application of hidden markov models and hidden semi-markov models to financial time series. 2006.

Lijuan Cao. Support vector machines experts for time series forecasting. *Neurocomputing*, 51: 321–339, 2003.

Damien Challet and Robin Stinchcombe. Analyzing and modeling 1+ 1d markets. *Physica A: Statistical Mechanics and its Applications*, 300(1):285–299, 2001.

Stephan K Chalup and Andreas Mitschele. Kernel methods in finance. In *Handbook on information technology in finance*, pages 655–687. Springer, 2008.

Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

Peter K Clark. A subordinated stochastic process model with finite variance for speculative prices. *Econometrica: journal of the Econometric Society*, pages 135–155, 1973.

Rama Cont and Arseniy Kukanov. Optimal order placement in limit order markets. 2013.

Rama Cont, Sasha Stoikov, and Rishi Talreja. A stochastic model for order book dynamics. *Operations research*, 58(3):549–563, 2010.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Bradley Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, pages 1–26, 1979.

Robert F Engle. The econometrics of ultra-high-frequency data. *Econometrica*, 68(1):1–22, 2000.

Robert F Engle and Jeffrey R Russell. Forecasting the frequency of changes in quoted foreign exchange prices with the autoregressive conditional duration model. *Journal of empirical finance*, 4(2):187–212, 1997.

Robert F Engle, Robert Ferstenberg, and Jeffrey R Russell. Measuring and modeling execution cost and risk. 2006.

Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367, 2003.

Jianqing Fan and Jinchi Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5):849–911, 2008.

Jianqing Fan, Rui Song, et al. Sure independence screening in generalized linear models with np-dimensionality. *The Annals of Statistics*, 38(6):3567–3604, 2010.

Tristan Fletcher. Hybrid evolutionary techniques for fx arbitrage prediction. 2007.

Tristan Fletcher, Fabian Redpath, and Joe DAlessandro. Machine learning in fx carry basket prediction. In *Proceedings of the World Congress on Engineering*, volume 2, 2009.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

Kuzman Ganchev, Yuriy Nevmyvaka, Michael Kearns, and Jennifer Wortman Vaughan. Censored exploration and the dark pool problem. *Communications of the ACM*, 53(5):99–107, 2010.

Parameswaran Gopikrishnan, Vasiliki Plerou, Xavier Gabaix, and H Eugene Stanley. Statistical properties of share volume traded in financial markets. *Physical Review E*, 62(4):R4493, 2000.

Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

Xin Guo. Optimal placement in a limit order book. In *Theory Driven by Influential Applications*, pages 191–200. INFORMS, 2013.

Anthony D Hall, Nikolaus Hautsch, et al. *A continuous-time measurement of the buy-sell pressure in a limit order book market*. Institute of Economics, University of Copenhagen, 2004.

Yasushi Hamao and Joel Hasbrouck. Securities trading in the absence of dealers: Trades and quotes on the tokyo stock exchange. *Review of Financial Studies*, 8(3):849–878, 1995.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.

Larry Harris. *Trading and exchanges: Market microstructure for practitioners*. Oxford University Press, USA, 2003.

Sherif Hashem. Optimal linear combinations of neural networks. *Neural networks*, 10(4):599–614, 1997.

Md Rafiul Hassan and Baikunth Nath. Stock market forecasting using hidden markov model: a new approach. In *Intelligent Systems Design and Applications, 2005. ISDA'05. Proceedings. 5th International Conference on*, pages 192–196. IEEE, 2005.

Md Rafiul Hassan, Baikunth Nath, and Michael Kirley. A fusion model of hmm, ann and ga for stock market forecasting. *Expert Systems with Applications*, 33(1):171–180, 2007.

Neep Hazarika and John G Taylor. Predicting bonds using the linear relevance vector machine. *Neural networks and the financial markets*, pages 145–155, 2002.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Shian-Chang Huang and Tung-Kuang Wu. Wavelet-based relevance vector machines for stock index forecasting. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 603–609. IEEE, 2006.

Shian-Chang Huang and Tung-Kuang Wu. Combining wavelet-based feature extractions with relevance vector machines for stock index forecasting. *Expert Systems*, 25(2):133–149, 2008.

Wei Huang, Yoshiteru Nakamori, and Shou-Yang Wang. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522, 2005.

Weibing Huang, Charles-Albert Lehalle, and Mathieu Rosenbaum. Simulating and analyzing order book data: The queue-reactive model. *Journal of the American Statistical Association*, 110 (509):107–122, 2015.

James M Hutchinson, Andrew W Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3):851–889, 1994.

Nicolas Huth and Frédéric Abergel. The times change: Multivariate subordination. empirical facts. *Quantitative Finance*, 12(1):1–10, 2012.

Patrik Idvall and Conny Jonsson. Algorithmic trading: Hidden markov models on foreign exchange data, 2008.

Plamen Ch Ivanov, Ainslie Yuen, Boris Podobnik, and Youngki Lee. Common scaling patterns in intertrade times of us stocks. *Physical Review E*, 69(5):056107, 2004.

Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

Balázs Kégl. The return of adaboost. mh: multi-class hamming trees. *arXiv preprint arXiv:1312.6086*, 2013.

Alec N Kercheval and Yuan Zhang. Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329, 2015.

Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003.

Andrei A Kirilenko, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. The flash crash: The impact of high frequency trading on an electronic market. *Available at SSRN 1686004*, 2015.

Chung-Ming Kuan and Tung Liu. Forecasting exchange rates using feedforward and recurrent neural networks. *Journal of applied econometrics*, 10(4):347–364, 1995.

Camilla Landen. Bond pricing in a hidden markov model of the short rate. *Finance and Stochastics*, 4(4):371–389, 2000.

Sophie Laruelle, Charles-Albert Lehalle, and Gilles Pages. Optimal split of orders across liquidity pools: a stochastic algorithm approach. *SIAM Journal on Financial Mathematics*, 2(1): 1042–1076, 2011.

Hugh Luckock. A statistical model of a limit order market. *Sidney University preprint (September 2001)*, 2001.

Rogemar S Mamon and Robert J Elliott. *Hidden markov models in finance*, volume 4. Springer, 2007.

Sergei Maslov and Mark Mills. Price fluctuations from the order book perspectiveempirical facts and a simple model. *Physica A: Statistical Mechanics and its Applications*, 299(1):234–246, 2001.

Szabolcs Mike and J Doyne Farmer. An empirical behavioral model of liquidity and volatility. *Journal of Economic Dynamics and Control*, 32(1):200–234, 2008.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

Anna A Obizhaeva and Jiang Wang. Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1–32, 2013.

Beomsoo Park and Benjamin Van Roy. Adaptive execution: Exploration and learning of price impact. *Operations Research*, 63(5):1058–1076, 2015.

Fernando Pérez-Cruz, Julio A Afonso-Rodriguez, Javier Giner, et al. Estimating garch models using support vector machines*. *Quantitative Finance*, 3(3):163–172, 2003.

Marc Potters and Jean-Philippe Bouchaud. More statistical properties of order books and price impact. *Physica A: Statistical Mechanics and its Applications*, 324(1):133–140, 2003.

Alessandro Rossi and Giampiero M Gallo. Volatility estimation via hidden markov models. *Journal of Empirical Finance*, 13(2):203–230, 2006.

Ioanid Roşu. A dynamic model of the limit order book. *Review of Financial Studies*, 22(11): 4601–4641, 2009.

Ioanid Rosu. A dynamic model of the limit order book. *Forthcoming in The Review of Financial Studies*, 2009.

Tobias Rydén, Timo Teräsvirta, and Stefan Åsbrink. Stylized facts of daily return series and the hidden markov model. *Journal of applied econometrics*, pages 217–244, 1998.

Jimmy Shadbolt. *Neural Networks and the Financial Markets: Bpredicting, Combining, and Portfolio Optimisation*. Springer Science & Business Media, 2002.

A Christian Silva and Victor M Yakovenko. Stochastic volatility of financial markets as the fluctuating rate of trading: An empirical study. *Physica A: Statistical Mechanics and its Applications*, 382(1):278–285, 2007.

Justin Sirignano. Deep learning for limit order books. 2016.

Francis EH Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

Francis EH Tay and LJ Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1):847–861, 2002.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Peter Tino, Nikolay Nikolaev, and Xin Yao. Volatility forecasting with sparse bayesian kernel models. In *Proc. 4th International Conference on Computational Intelligence in Economics and Finance, Salt Lake City, UT*, pages 1150–1153, 2005.

Robert R Trippi and Efraim Turban. *Neural networks in finance and investing: Using artificial intelligence to improve real world performance*. McGraw-Hill, Inc., 1992.

Tony Van Gestel, Johan AK Suykens, D-E Baestaens, Annemie Lambrechts, Gert Lanckriet, Bruno Vandaele, Bart De Moor, and Joos Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on neural networks*, 12(4):809–821, 2001.

Sunil Wahal and M Deniz Yavuz. Style investing, comovement and return predictability. *Journal of Financial Economics*, 107(1):136–154, 2013.

Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of management information systems*, 17(4):203–222, 2001.

Philipp Weber and Bernd Rosenow*. Order book approach to price impact. *Quantitative Finance*, 5(4):357–364, 2005.

Yingjian Zhang. *Prediction of financial time series with Hidden Markov Models*. PhD thesis, Simon Fraser University, 2004.

Nan Zhou, Wen Cheng, Yichen Qin, and Zongcheng Yin. Evolution of high-frequency systematic trading: a performance-driven gradient boosting model. *Quantitative Finance*, 15(8):1387–1403, 2015.

Ilija Zovko, J Doyne Farmer, et al. The power of patience: a behavioural regularity in limit-order placement. *Quantitative finance*, 2(5):387–392, 2002.

# BIOGRAPHICAL SKETCH

Jian Wang is currently a PhD candidate majoring in Financial Mathematics at Department of Mathematics at Florida State University(FSU) in Tallahassee, Florida, USA. His research interests include machine learning, high frequency trading schemes, data mining and limit order book dynamics. He obtained a Master in Mathematics from FSU, a Bachelor and master degree of mathematics from Shanghai University, China, 2005. He is the member of society of actuaries. He also have around three years quantitative working experience in Sino-life insurance company, shanghai, China.