Investment Strategy Based on Machine Learning Models

by

Jiayue Zhang

A Master's Research Paper
presented to the University of Waterloo
in fulfillment of the
requirement for the degree of
Master of Science
in
Computational Mathematics

Supervised by Prof. Ken Seng Tan

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In recent years, applying machine learning algorithms to create a stock picking strategy has become popular. This essay highlights several innovations in this area and compares the general ideas with proposed improvements presented in this essay. Firstly, we use daily data between Jan.1st 2014 and Dec.31st 2018 on S&P 500 index to create lagged return features and generate labels according to the performance of each stock. After training Deep Neural Network (DNN), Random Forest (RAF), XGBoost and Support Vector Machine (SVM), daily one-day-ahead trading signals are generated based on the probability forecast of a stock to outperform its cross-sectional median. We long n stocks with the highest probabilities and short the lowest n stocks. To make further improvement, we compare 'Day-Trading', 'Normal-Trading', 'Vertical Ensemble Model' and 'Horizontal Ensemble Model', and try to reduce dimension of the models for the purpose of increasing a running speed. Moreover, we make an attempt to combine classification models and regression models, such as Recurrent Neural Network, which helps increase the annualized return by almost 25.8%. Finally, we duplicate the same ideas on Chinese Stock Market but only 'long' the highest $2n$ stocks without taking a 'short' position. Although we can earn a profit using this strategy, the efficiency on the Chinese market is lower than that on the U.S. market.

Keywords: XGBoost, Day-Trading, PCA, Recurrent Neural Network, LSTM

Acknowledgements

I would like to thank my supervisor, Professor Ken Seng Tan, for his guidance, support and encouragement throughout this year. Also, I want to express my sincere thanks to Professor Tony Wirjanto, for reading this research paper and providing valuable suggestions. To my parents and my boyfriend, for their love and inspiration throughout my life. To my dear friends, for always sharing ideas and making progress with me: Xuanrui, Greg, Meiyu, Michael, Lingyun. Last but not least, I want to thank our director, Prof. Jeff Orchard, Prof. Henry Wolkowicz and coordinator, Cherisse Mike, for their information and suggestion on daily life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data mining is a process of finding hidden patterns within data using automatic or semi-automatic learning algorithms. The accelerating development of computer technology and machine learning has generated increasing research interests in the innovative solution to traditional challenges in social sciences. In particular, machine learning (ML) techniques have shown impressive performance in solving real life classification problems in many different areas such as communications, internet traffic analysis, medical imaging, astronomy, document analysis, biology and time series analysis(Gerlein, 2016).[11] In term of finance applications, especially in the equity market, finding effective investment strategies based on computer science theoretical algorithms is becoming more popular in recent years.

Financial trading of securities using technical and quantitative analysis has been traditionally modelled by statistical techniques for time series analysis such as ARMA and ARIMA models, and more sophisticated ARCH models (Engle, 1982) [10]. In contrast to these statistical approaches, complex models coming from the ML field have emerged attempting to predict future movements of securities' prices [4]. The extensive literature has shown how some ML techniques specializing in classification and regression tasks have been shown to be well-suited for a quantitative analysis in the financial industry, as their capabilities of finding hidden patterns in large amounts of financial data may help in derivatives pricing, risk management and financial forecasting. In this essay, we compare prediction accuracies of our investment strategy based on different machine learning models and methods and make some innovations to improve the return rates.

1.1 Investment Strategy with Classification Models

Regression and classification are categorized under the same umbrella of supervised machine learning. Both share the same concept of utilizing known datasets (referred to as training datasets) to make predictions. In supervised learning, an algorithm is employed to learn a mapping function from a set of input variables, X, to an output variable, y; that is $y = f(X), where\ X = (x_1, x_2, ..., x_p)$. The objective of such a problem is to approximate the mapping function f as accurately as possible such that whenever there is a new input data x_0, the output variable y for the dataset can be predicted. The main difference between classification models and regression models is that the output variable in regression is numerical (or continuous) while that for classification is categorical (or discrete).

Most relevant researches about the investment strategy using machine learning models and methods mainly focus on classification models. Takeuchi and Lee (2013) [23] develop an enhanced momentum strategy on the U.S. stock market using the data from 1965 until 2009. They apply deep neural networks (DNN) as classifiers to calculate the probability for each stock to outperform the cross-sectional median return of all stocks in the holding month *t + 1*. Using standardized cumulative returns produces annualized returns of 45.93 percent in the out-of-sample testing period from 1990 until 2009. Heaton et al. (2016) [12] discuss the application of deep learning to financial prediction and classification, which is able to exploit empirical data to find the function relationship between an output variable and a group of input variables, that are not predicted by existing financial economic theory. Krauss and Huck (2016) [16] apply four kinds of classification models and their ensemble models to predict the probability of outperforming a cross-sectional median performance in next period, which shows the best result is obtained by *Tree-based* models. Culkin and Das (2017)[6] train a deep learning neural network to calculate option prices, which achieves a high degree of accuracy compared to the Black-Scholes option pricing formula. Nobre and Ferreira Neves(2018)[20] present an expert system in the financial area that combines Principal Component Analysis (PCA), Discrete Wavelet Transform (DWT), Extreme Gradient Boosting (XGBoost) and a Multi-Objective Optimization Genetic Algorithm (MOO-GA) in order to achieve high returns with a low level of risk.

1.2 Investment Strategy with Regression Models

Regression algorithms attempt to estimate the mapping function f from the input variables X to a numerical or continuous output variable. In financial applications, regression models are often used to predict the specific prices for next period, so that investors can make decisions according to the predicted return rates.

Kazema and Sharifia (2013) [14] apply forecasting model based on a chaotic mapping, firefly algorithm, and a SVR to predict the stock market price, which gives a better result than using an ARIMA-type model. Wang and Wang (2016) [24] attempt to improve the forecasting accuracy of crude oil price fluctuations by combining Multilayer perception and Elman recurrent neural networks (ERNN) with a stochastic time effective function. Naik and Mohan(2019) [18] collect data from India National Stock Exchange and use RNN with LSTM to forecast future stock returns for the purpose of gaining a higher excess trading profit.

In this essay, we also use supervised machine learning models to perform predictions of stocks return and find an investment strategy. Instead of only using classification models or only using regression models, we combine them with the goal to improve the excess return and the corresponding Sharpe Ratio. Also, PCA is used to reduce the dimension of the model for the purpose of decreasing the required computational time.

Chapter 2

Data and Features Generation

2.1 Data Collection

Motivated by computational feasibility, market efficiency, and liquidity, we choose the S&P 500 index in U.S. stock market and the CSI 300 index in Chinese stock market as our objects. The S&P 500 index consists of the leading 500 companies in the U.S. stock market, accounting for approximately 80 percent of available market capitalization. The CSI 300 index is a capitalization-weighted stock market index designed to replicate the performance of top 300 stocks traded in the Shanghai and Shenzhen stock exchanges. Firstly, we set the sample period range from January 2014 to December 2018. Secondly, following Krauss and Stubinger (2015) [17],we obtain all month end constituent lists for both the S&P 500 index and the CSI 300 index from Wind Financial Terminal. We consolidate these lists into one binary matrix, indicating whether the stock is a constituent of the index in the subsequent month or not. Finally, for all stocks having been a constituent of the index, we collect all of the daily transaction data for every stock, which include *transaction date, volume, open price, close price and adjusted closing price.*

2.2 Training and Testing Set

For the whole dataset from January 2014 to December 2018, training set is defined from January 2015 to December 2017 while the records in 2014 are used as basis to calculate return rates. We use the building models to test the data in 2018 between January 1st to December 31st and split the datasets using the same argument in both markets.

2.3 Features and Labels Generation

Features: For each stock in both markets, we generate features using their daily adjusted closing price. Let $\left(P_t^S\right)_{t\in T}$ denote the price process of stock S, then we define simple return rates $R_{t,m}^S$ for each stock S over m periods using the equation as $R_{t,m}^S = \frac{P_t^S}{P_{t-m}^S} - 1$. To capture useful information as much as possible, we consider the periods $m \in \big\{(1,2,3,...,20) \cup (40,60,80,\ldots,240)\big\}$. Normally, there are 20 trading days in one month, so $m \in (1,2,3,...,20)$ represents the lagged returns for each trading day in last month. Similarly, $m \in (40,60,80,...,240)$ takes records for returns in each month during the last year. As a result, we generate a total 31 features for each stock on every trading day.

Labels: We consider classification machine learning models at the initial step, so we need to generate a pair of binary labels for every record. If the one-period return $R_{t+1,1}^s$ is larger than the corresponding cross-sectional median, that means this stock outperforms and the label $Y_{t+1}^s = 1$. Otherwise, we set the label as 0. For a testing purpose, we also should forecast a probability $\mathcal{P}_{t+1|t}^s$ for each stock S to outperform the cross-sectional median in period $t+1$. After we get the predicted probability, we can compare the predicted label and the true label.

Chapter 3

Models Construction

3.1 Deep Neural Network

The deep neural network normally includes three parts, one input layer, one or more hidden layers, and one output layer[16]. In a fully connected feed-forward network, each node is connected to every node in the next layer. Associated with each edge between the i^{th} node in the previous layer and the j^{th} node in the current layer l is a weight $w_{ij}^{(l)}$. In order to find optimal weightings $\mathbf{w} := \left\{\mathbf{w}^{(l)}\right\}_{l:=1\to L}$ between nodes in a fully connected feed forward network with L layers, we seek to minimize a cross-entropy function of the form:

$$E(\mathbf{w}) = -\sum_{n=1}^{N_{\text{test}}} e_n(\mathbf{w}), \quad e_n(\mathbf{w}) := \sum_{k=1}^{K} y_{kn} \ln\left(\hat{y}_{kn}\right) \tag{3.1}$$

For clarity of exposition, we drop the subscript n. Here K denotes the total number of classes. The binary target vector y and binary output vector $\hat{y}$ have a $1-of-n_s$ encoding for each symbol, where n_s is the number of classes per symbol, so that each state associated with a symbol can be interpreted as a probabilistic weighting. Put formally, $y_k \in \{0,1\}, \forall k \in K$ and $\sum_{k\in\mathcal{K},y_k} = 1, \forall i$ where $\mathcal{K}_i$ is the set of n_s class indices associated with symbol i [9]. To ensure analytic gradient functions under the cross-entropy error measure, each of the nodes associated with the i^{th} symbol in the output layer are activated with a *softmax* function of the form:

$$\hat{y}_k := \phi_{\text{softmax}}\left(\mathbf{s}^{(L)}\right) = \frac{\exp\left(s_k^{(L)}\right)}{\sum_{j\in\mathcal{K}_i}\exp\left(s_j^{(L)}\right)}, \forall k \in \mathcal{K}_i \tag{3.2}$$

The gradient of the likelihood function $w.r.t.s$ then takes simple form:

$$\frac{\partial e(\mathbf{w})}{\partial s_k^{(L)}} = \hat{y}_k - y_k \tag{3.3}$$

and in a fully connected feed-forward network $s_k^{(l)}$ is the weighted sum of outputs from the previous layer $l-1$ that connect to node j in layer l:

$$s_j^{(l)} = \sum_i^{n_{l-1}} w_{ij}^{(l)} x_i^{(l-1)} + \text{bias}_j^{(l)} \tag{3.4}$$

Here n_l is the number of nodes in layer l. For each node i in the $(l-1)_{th}$ layer, the recursion relation for the back propagation using conjugate gradients is:

$$\delta_i^{(l-1)} = \sum_{j=1}^{n_{(l)}} \delta_j^{(l)} w_{ij}^{(l)} \sigma\left(s_i^{(l-1)}\right)\left(1-\sigma\left(s_i^{(l-1)}\right)\right) \tag{3.5}$$

where we have used the analytic form of the derivative of the *sigmoid* function:

$$\sigma'(v) = \sigma(v)(1-\sigma(v)) \tag{3.6}$$

which is used to activate all hidden layer nodes.

A trained feed-forward network can be used to predict the outputs states of all symbols, given any observation as an input. In this essay, the number of nodes in every layer is 31-31-16-8-2; the dropout ratio is 0.2 in input layer and 0.4 in hidden layers; the activation is 'softmax'; the loss function is 'categorical crossentropy'; the optimizer is 'adam'; the shrinkage parameter is 0.00001; the epoch is 400 and use 'early stopping' to decide the ending point.

3.2 Random Forest

The random forest classifier consists of a combination of tree classifiers where each classifier is generated using a random vector sampled independently from the input vector, and each tree casts a unit vote for the most popular class to classify an input vector Breiman (1999)[22]. The random forest classifier used for this study consists of using randomly selected features or a combination of features at each node to grow a tree.

Following the logic in Krauss(2017)[16], we draw a random subset from the original training data. Then, we grow a modified decision tree to this sample. There are some important parameters needed to be selected. We select m_{RAF} features at random from the p features upon every split. We grow the tree to the maximum depth of J_{RAF} . The final output is an ensemble of B_{RAF} random forest trees, so that classification can be performed via a majority vote. Sub-sampling substantially reduces variance of (low bias) trees and the random feature selection decorrelates them. We have three tuning parameters, i.e., the number of trees B_{RAF} , their maximum depth J_{RAF} , and the number of features to randomly select m_{RAF} . Random forests are not prone to overfit, so we can choose a high B_{RAF} of 800 trees. We fix the maximum depth J_{RAF} at 20 and the number of features in sub-sampling m_{RAF} at $\lfloor\sqrt{p}\rfloor$.

3.3 XGBoost

In boosting, the trees are built sequentially such that each subsequent tree aims at reducing the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals. The base learners in boosting are weak learners in which the bias is high, and the predictive power is just slightly better than random guessing. Each of these weak learners contributes some vital information to prediction, enabling the boosting technique to produce a strong learner in the end by effectively combining these weak learners. The final strong learner is expected to bring down both the bias and the variance.

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. Such small trees, which are not very deep, are highly interpretative. Parameters like the number of trees or iterations, the rate at which the gradient boosting learns, and the depth of the tree, could be optimally selected through cross-validation technique, such as k-fold cross validation.

Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria in the boosting procedure. Boosting consists of three simple steps:

- An initial model F_0 is defined to predict the target variable y. This model will be associated with the residual (y–F_0).
- A new model h_1 is fit to the residuals from the previous step
- Now, F_0 and h_1 are combined to give F_1, the boosted version of F_0. The mean squared error from F_1 will be lower than that from F_0:

$$F_1(x) < -F_0(x) + h_1(x) \tag{3.7}$$

To improve the performance of F_1, we could model after the residuals of F_1 and create a new model F_2:

$$F_2(x) < -F_1(x) + h_2(x) \tag{3.8}$$

It can be done for m iterations, until the residuals have been minimized as much as possible:

$$F_m(x) < -F_{m-1}(x) + h_m(x) \tag{3.9}$$

Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors.

For MSE, the change observed would be roughly exponential. Instead of fitting $h_m(x)$ on the residuals, fitting it on the gradient of the loss function, or the step along which loss occurs, would make this process generic and applicable across all loss functions.

Gradient descent helps us minimize any differentiable function. Earlier, the regression tree for $h_m(x)$ predicted the mean residual at each terminal node of the tree. In gradient boosting, the average gradient component would be computed.

For each node, there is a factor γ with which $h_m(x)$ is multiplied. This accounts for the difference in impact of each branch of the split. Gradient boosting helps in predicting the optimal gradient for the additive model, unlike the classical gradient descent techniques which reduce error in the output at each iteration.

The following steps are involved in the gradient boosting procedure:

- $F_0(x)$ is to be defined as (initialize the boosting algorithm):

$$F_o(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L\left(y_i, \gamma\right) \tag{3.10}$$

- The gradient of the loss function is computed iteratively:

$$\gamma_{im} = -\alpha \left[\frac{\partial \left(L\left(y_0\right), F(x) \right)}{\partial F\left(x_j\right)} \right]_{F(x)=F_{m-1}(x)}, \text{ where } \alpha \text{ is the learning rate} \tag{3.11}$$

- Each $h_m(x)$ is fit on the gradient obtained at each step. The multiplicative factor γ_m for each terminal node is derived and the boosted model $F_m(x)$ is defined:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{3.12}$$

XGBoost stands for eXtreme Gradient Boosting, which is a popular implementation of gradient boosting. It has an option to penalize complex models through both L_1 and L_2 regularization. Regularization helps in preventing overfitting. XGBoost has a strong ability to deal with missing data and it has a distributed weighted quantile sketch algorithm to effectively handle weighted data. Moreover, for faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. Data is sorted and stored in in-memory units called blocks. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again. This feature is also useful for steps like split finding and column sub-sampling. In this eassy, we use the following set of parameters: the number of trees or boosting iterations $M_{XGB} = 100$, the depth of the tree $J_{XGB} = 3$, the learning rate $\lambda_{XGB} = 0.1$, and the subset of features to use at each split $m_{XGB} = 16$.

3.4 Support Vector Machines

SVMs are based on statistical learning theory and have the aim of determining the location of decision boundaries that produce the optimal separation of classes. In a two-class pattern recognition problem where classes are linearly separable, the SVMs select the one linear decision boundary that leaves the greatest margin between the two classes. The margin is defined as the sum of the distances to the hyperplane from the closest points of the two classes (Vapnik 1995). This problem of maximizing the margin can be solved using

standard Quadratic Programming (QP) optimization techniques. The data points that are closest to the hyperplane are used to measure the margin. Therefore, these data points are termed 'support vectors' and are always small in number [21].

If the two classes are not linearly separable, the SVMs try to find the hyperplane that maximizes the margin, while at the same time, minimizing a quantity proportional to the number of misclassification errors. The trade-off between margin and misclassification error is controlled by a positive user-defined parameter C. SVMs can also be extended to handle nonlinear decision surfaces. Boser et al. (1992)[3] proposed a method for projecting the input data into a high-dimensional feature space through some nonlinear mapping, and formulating a linear classification problem in that feature space. Kernel functions are used to reduce the computational cost of dealing with a high-dimensional feature space. SVMs were initially designed for binary (two-class) problems[21]. When dealing with multiple classes, an appropriate multi-class method is needed. Techniques such as 'one against one' and the 'one against the rest' are in frequent use for the multi-class problems. In this essay, we select a kernel width $\gamma = 0.8$ and a regularization parameter $C = 3000$.

3.5 Recurrent Neural Network

RNN is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

LSTM networks, which are used in this essay are a deep and recurrent model of neural networks. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network (Figure 3.1). Recurrent networks differ from the traditional feed-forward networks in the sense that they do not only have neural connections on a single direction, in other words, neurons can pass data to a previous or the same layer. In which case, data doesn't flow on a single way, and the practical effects for that is the existence of short term memory, in addition to long term memory that neural networks already have in consequence of training[19]. LSTM were introduced by Sepp Hochreiter and Jürgen Schmidhuberand [24], it aimed at a better performance by tackling the vanishing gradient issue that recurrent networks would suffer when dealing with long data sequences. It does so by keeping the error flow constant through special units called "gates" which allows for weights adjustments as well as truncation of the gra-

dient when its information is not necessary.

In this essay, we add four LSTM layers, unit number in each layer is set as 136, Dropout is 0.2. In the output layer, the type of optimizer used can greatly affect how fast the algorithm converges to the minimum value. Also, it is important that there is some notion of randomness to avoid getting stuck in a local minimum and not reach the global minimum. We choose to use Adam optimizer, which combines the perks of two other optimizers: ADAgrad and RMSprop.

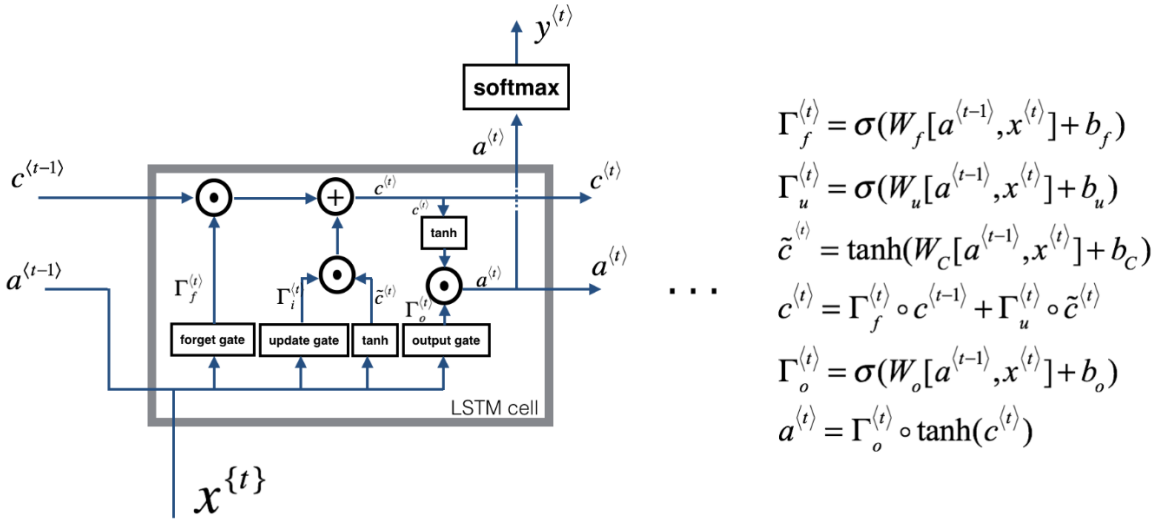

Figure 3.1: LSTM Architecture

Chapter 4

Strategy and Results

4.1 Basic Investment Strategy

In classification models, we select four representative models: Deep Neural Network(DNN), Random Forest(RAF), XGBoost and Support Vector Machine(SVM). As mentioned before, we set Jan.1st, 2015 - Dec.31st, 2017 as the training set and Jan.1st, 2018 to Dec.31st, 2018 as our test set.

The basic investment strategy is :

Step 1: For each period t+1 in the test set, we forecast the probability $\left(\mathcal{P}^S_{t+1|t}\right)_a$ for each stock S to outperform its cross-sectional median, where $a \in (DNN, RAF, XGBoost, SVM)$.

Step 2: Sorting all stocks in descending order results in 4 rankings for 4 models so that we could go long the top n stocks and short the bottom n ones according to every model, where $n \in (10, 20, 50, 100)$. By the results in DeMiguel (2007) [7], we choose a Naïve portfolio strategy (1/N Portfolio Strategy) to decide the weight for each stock.

In the basic investment strategy, we rebalance and reconstruct the portfolio every trading day. As a result, we get different daily returns each day. Following Avellaneda and Lee (2010)[1], we define the transaction costs of 0.05 percent per share per half-turn. To compare the efficiency and profitability, there are some measurements that need to be explained:

- *Daily return for each stock S:*

$$R^S_t = \frac{P^S_t}{P^S_{t-1}} - 1, \tag{4.1}$$

where P is the price, S is the stock name and t is the date.

- *Average Daily return:*

$$R_{t,a} = \frac{\sum_{S \in constituents\ in\ day\ t} R^S_{t,a}}{number\ of\ constituents\ in\ day\ t}. \tag{4.2}$$

- *Annualized return for model a:*

$$R_a = \prod_{t \in all\ trading\ days\ in\ 2018} (1 + R_{t,a}) - 1, \tag{4.3}$$

where a ∈ (DNN, RAF, XGBoost, SVM).

- *Standard Deviation for model a:*

$$\sigma_a = \sqrt{\frac{\sum_{t \in all\ trading\ days\ in\ 2018} (R_{t,a} - \bar{R_{t,a}})^2}{N-1}}, \tag{4.4}$$

After using parameters mentioned above in each model, the results are shown below:

	Before Fees				*After Fees*			
	DNN	RAF	XGBoost	SVM	DNN	RAF	XGBoost	SVM
Mean Return	1.0708	1.4774	1.6988	1.6734	0.2943	0.6981	0.9132	0.8925
Excess Return	1.0117	1.4127	1.6327	1.6027	0.2267	0.6277	0.8312	0.8034
St.D	0.3981	0.3786	0.3771	0.3846	0.3981	0.3786	0.3771	0.3846
Sharpe Ratio	2.5413	3.7324	4.3208	4.1672	0.5694	1.6580	2.2042	2.0889

Table 4.1: Annualized Return for n=10 (long 10, short 10)

	Before Fees				*After Fees*			
	DNN	RAF	XGBoost	SVM	DNN	RAF	XGBoost	SVM
Mean Return	0.9664	1.2447	1.4563	1.4412	0.2012	0.4641	0.6812	0.5887
Excess Return	0.9012	1.1774	1.3887	1.3779	0.1312	0.3884	0.5318	0.5124
St.D	0.3887	0.3612	0.3677	0.3654	0.3887	0.3612	0.3677	0.3654
Sharpe Ratio	2.3185	3.2596	3.7767	3.7709	0.3375	1.0753	1.4463	1.4023

Table 4.2: Annualized Return for n=20 (long 20, short 20)

	Before Fees				*After Fees*			
	DNN	RAF	XGBoost	SVM	DNN	RAF	XGBoost	SVM
Mean Return	0.8142	1.1128	1.3327	1.2901	0.1832	0.3017	0.5642	0.5331
Excess Return	0.7312	1.0324	1.2701	1.2375	0.1271	0.2315	0.5001	0.4565
St.D	0.3633	0.3488	0.3500	0.3497	0.3633	0.3488	0.3500	0.3497
Sharpe Ratio	2.0126	2.9599	3.6288	3.5387	0.3498	0.6637	1.4289	1.3054

Table 4.3: Annualized Return for n=50 (long 50, short 50)

	Before Fees				*After Fees*			
	DNN	RAF	XGBoost	SVM	DNN	RAF	XGBoost	SVM
Mean Return	0.7001	0.9981	1.2775	1.2335	0.1125	0.3388	0.4988	0.4622
Excess Return	0.6135	0.9273	1.2012	1.1781	0.0432	0.2701	0.4117	0.3882
St.D	0.3565	0.3461	0.3517	0.3442	0.3565	0.3461	0.3517	0.3442
Sharpe Ratio	1.7209	2.6793	3.4154	3.4197	0.1212	0.7804	1.1686	1.1278

Table 4.4: Annualized Return for n=100 (long 100, short 100)

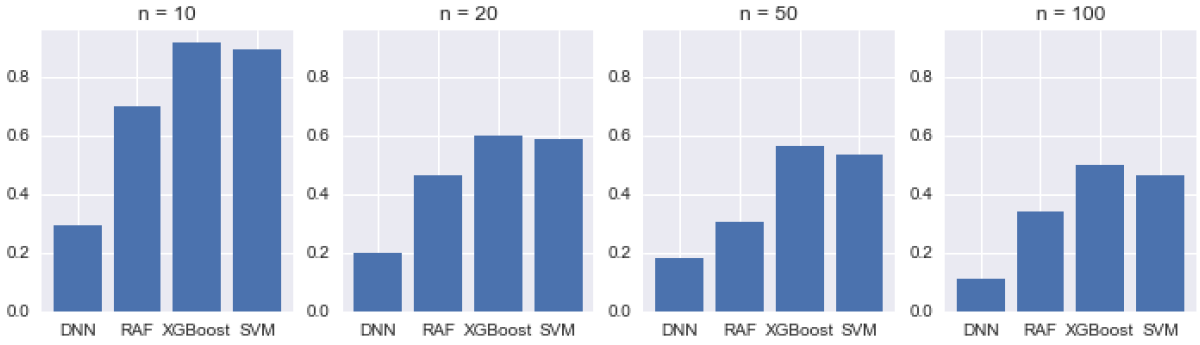

Figure 4.1: Annualized Mean Return after fees for different models

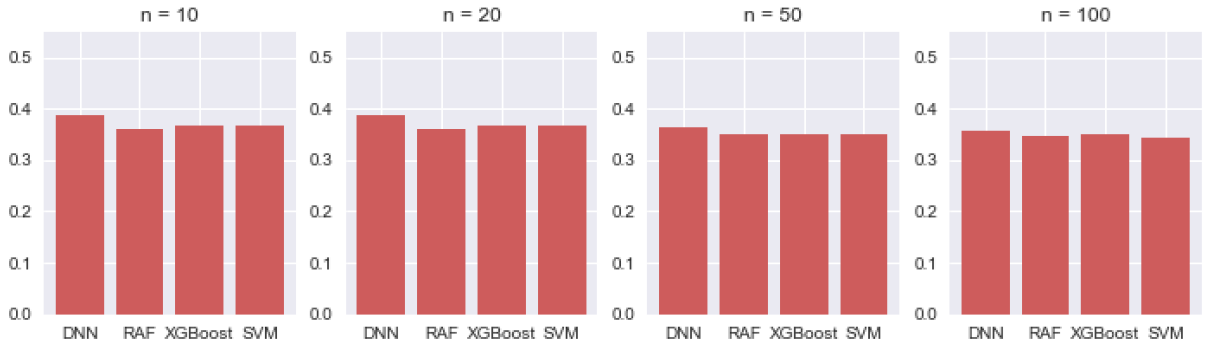

Figure 4.2: Standard Deviation for different models

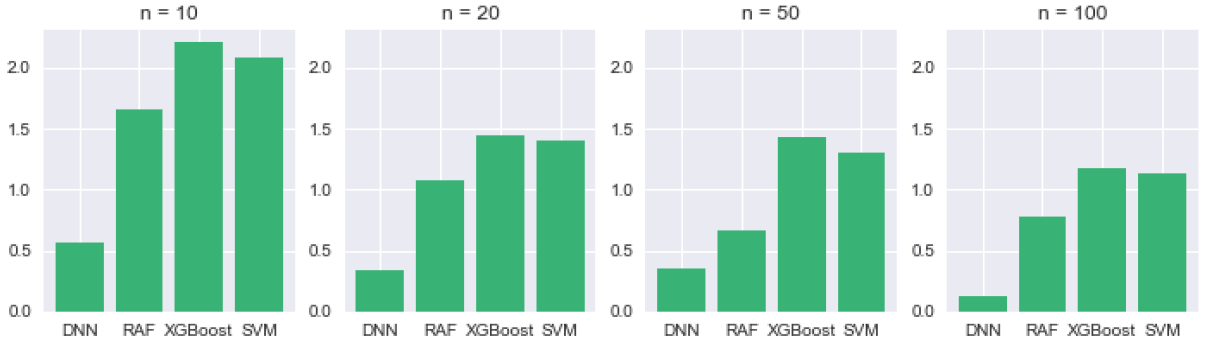

Figure 4.3: Sharpe Ratio after fees for different models

From the results above, we can draw the conclusions that:

- XGBoost's performance is the best, which has the highest return rate and the highest Sharpe Ratio, which means XGBoost helps us to make the best profit with the relative minimum risk. SVM is also a good model for this dataset and its perfomance is just a little bit worse than that of XGBoost. However, DNN does not show an acceptable result, since the returns are only 60% of the best return. From Chen (2016) [5], we can know that TREE-based models(such as GBDT and XGBoost) are more robust when we apply them to different datasets;

- Since we need to rebalance and reconstruct our portfolio every day, the transaction fee dominates the net returns and has a relative big influence on the final performance. As the results shown above, the returns decreased dramatically after we include transaction fees even though we can still still gain profits, albeit lower;

- From the figures, the mean returns decrease when the portfolio includes more stocks while the standard deviations of the returns remain almost the same with more stocks. After ranking all stocks according to their forecasting probabilities, the portfolio includes more stocks means it has to contain more stocks with lower outperformed probabilities, which might lead to a lower return. The results are reasonable.

- When we only long top 10 and short the last 10 stocks in our portfolio, the best return rate after considering transaction fees is 0.9132 with the highest Sharpe ratio of 2.2042.

4.2 Day Trading and Normal Trading

Day trading is speculation in securities, specifically buying and selling financial instruments within the same trading day, such that all positions are closed before the market closes for the trading day. The methods of quick trading contrast with the long-term trades underlying buy and hold and value investing strategies. Day traders exit positions before the market closes to avoid unmanageable risks and negative price gaps between one day's close and the next day's price at the open. By Bitvai (2014)[2], we get the information that day trading can make a better profit in some cases. Motivated by this idea, we compare the results between 'day trading' and the previous normal trading. The main difference is whether holding the portfolio overnight, so the buying price for 'day trading' is today's open price while the normal trading is yesterday's adjusted close price.

According to the comparisons between different models, we select XGBoost with $n = 10$ as our main model. By keeping all other features unchanged, we only change the return formula as $R^S_{t,dt} = \frac{P^S_{t,close}}{P^S_{t,open}} - 1$. Following the same logic, we long the top 10 and short the bottom 10 according to the forecasting probabilities. The comparisons for these two trading strategies are shown below:

	Before Fees		*After Fees*	
	Day Trading	Normal Trading	Day Trading	Normal Trading
Mean Return	1.2862	1.6988	0.4437	0.9132
Excess Return	1.2101	1.6321	0.3753	0.8312
St.D	0.3962	0.3771	0.3962	0.3771
Sharpe Ratio	3.0543	4.3208	0.9451	2.2042

Table 4.5: Day Trading v.s. Normal Trading by XGBoost (n=10)

From the results above, day trading's return is less than that of normal trading, and the difference between them is quite large. Including transaction costs, the day trading can only achieve an excess return of 0.3753, while the excess return of normal trading is 0.8312. So we may not regard the day trading as a good strategy unless the investor do not want their money to be in the market overnight.

4.3 Label Improvements using Different Methods

In this part, we try some innovations on label generation. According to the generation directions, vertical ensemble model and horizontal ensemble model are defined differently.

4.3.1 Vertical Ensemble Model

Insider trading is the trading of a public company's stock or other securities based on material nonpublic information about the company[25]. Under Weak efficient market or Semi-strong efficient market, the nonpublic or private information can significantly help insider make huge profits. Normally, the private information owners are likely to trade before the signals show up in the market, leading to some early fluctuations which cannot be discerned by conventional investors.

In order to capture the early fluctuations in the market induced by insider trading, we use the last three days' forecasting probabilities instead of only using the probability of t*(only the previous day)* to predict the performance of $t+1$*(next period)*. Since this kind of model considers a vertical time line, we regard it as 'Vertical Ensemble Model'. Moreover, only using one day probability for prediction may hide the 'trend' for the stock and can also be disturbed by the presence of an 'extreme event'. As a result, we use a Grid Search

with the step-length of 0.1 for giving different weights for $\left(P^S_{t-1|t-2}, P^S_{t|t-1}, P^S_{t+1|t}\right)$. It is worth mentioning that we assume $weight\left(P^S_{t+1|t}\right) > 0.5$ and distribute the other different weights for $\left(P^S_{t-1|t-2}, P^S_{t|t-1}\right)$. After distributing different weights, we tried 20 weights combinations. The best weight combination is $\left(P^S_{t-1|t-2}, P^S_{t|t-1}, P^S_{t+1|t}\right) = (0.1, 0.3, 0.6)$. The excess returns and Sharpe Ratios for different combinations are shown below:

Figure 4.4: Excess Return and Sharpe Ratio after fees for different Vertical Ensemble Models

Similarly, we only show the detailed results from using XGBoost and $n = 10$:

	Before Fees		*After Fees*	
	Best Weights	Only t	Best Weights	Only t
Mean Return	1.8379	1.6988	1.0121	0.9132
Excess Return	1.7718	1.6321	0.9327	0.8312
St.D	0.3802	0.3771	0.3802	0.3771
Sharpe Ratio	4.6602	4.3208	2.4532	2.2042

Table 4.6: Best Weights v.s. Only t by XGBoost (n=10)

Comparing results under XGBoost model with $n = 10$, we find that the vertical ensemble model shows us a much better performance on excess return and risk measurement. After removing the influence of transaction costs, the ensemble model with the best weight combination increases the excess return by 10% than the model that only considers the probability of t. At the same time, the best weights combination also achieves a higher Sharpe Ratio at 2.4532. Thus, the vertical ensemble model is able to help capture the early fluctuations in the market.

4.3.2 Horizontal Ensemble Model

According to Dietterich(2000)[8], there are several reasons for the success of ensemble models. From Krauss(2007)[16], we receive useful information that Simple Ensemble Model is the best in three types of Ensemble models. Motivated by their researches, we consider to use Simple Ensemble model which defines the forecasting probability as $P^{S,ENS}_{t+1|t} = \frac{1}{4}\left(P^{S,DNN}_{t+1|t} + P^{S,RAF}_{t+1|t} + P^{S,XGB}_{t+1|t} + P^{S,SVM}_{t+1|t}\right)$. Since this step is a combination of four probabilities from different models in the same day, we define the Simple Ensemble model as 'Horizontal Ensemble Model'. However, this kind of the horizontal ensemble model does not have a better performance compared to XGBoost with 'Best Weights' in the last part. So we do not consider this Horizontal Ensemble Model any further. The comparisons are shown below:

	Before Fees		*After Fees*	
	Best Weights(XGB)	H-Ensemble	Best Weights(XGB)	H-Ensemble
Mean Return	1.8379	1.5142	1.0121	0.8761
Excess Return	1.7718	1.4681	0.9327	0.8071
St.D	0.3802	0.3790	0.3802	0.3790
Sharpe Ratio	4.6602	3.8736	2.4532	2.1296

Table 4.7: XGBoost (Best Weights) v.s. Horizontal Ensemble Model

4.4 Combinations of Classification and Further Regression

From the classification machine learning models, such as DNN, Random Forest, XGBoost and SVM, we can only get the output of 0 or 1 and the probability for output 1 as well.

From the regression machine learning model, such as RNN(Recurrent Neural Network), we have the ability to know the specific prediction prices for the next term. From Kamijo and Tanigawa(2012)[13], RNN can work well when it is used to predict time series data, especially stock prices. In this essay, we consider combining a classification model and a regression model in order to improve the return rates. Our new strategy follows these steps:

Step 1: Apply XGBoost with best weights (classification model) to select the top 10 and the last 10 stocks, totally 20 stocks, for each trading day, which can be regarded as a 'stock pool'.

Step 2: After getting a 'candidate pool', use RNN and LSTM approach to predict the prices for these 20 stocks in order to forecast tomorrow's return rates.

Step 3: When we obtain 'tomorrow's returns for these 20 stocks', select top 5 and last 5 according to their return ranks to create a new investment portfolio.

After switching to the new investment portfolio, we improve our return significantly and reduce capital use at the same time. Considering the relative slow running speed, we only compare the results of the last 30 trading days and convert the results to an annualized level. Sample size of 30 is already a valid statistical big sample which can show the main features in some extent. The comparison is shown as following:

	After Fees	
	Only Classification	After Regression
Mean Return	1.0121	1.3328
Excess Return	0.9327	1.1713
St.D	0.3802	0.4107
Sharpe Ratio	2.4532	2.8520

Table 4.8: Only Classification v.s. After Regression

As the results shown above, the combination with a regression model (RNN-LSTM) moves the excess return from 0.9327 to 1.1713, increasing almost by 25.8%. Meanwhile, the Sharpe Ratio also performs better than that from only using classification models.

4.5 Reduce dimension by PCA

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. Although we get a better return rate by using RNN-LSTM in the last section, the running time increases by almost 20 times. There is a general trade-off between the number of features (model accuracy) and computational speed. So we use a build-in function to apply PCA for the purpose of selecting the most important features from the original 31 features. This step is able to increase the running speed without losing much accuracy. In this essay, we try to keep the first 5, 10, 12, 15 and 20 features according to their importance. After re-running the model, we get the results as follows:

Num. of Features	5	10	12	15	20	31
Running Time/min(s)	5.6	8.3	9.2	11.7	13.2	17.6
Accuracy	0.5643	0.6042	0.6103	0.6237	0.6574	0.6632
Excess Return	0.3238	0.5997	0.8002	0.9190	1.1328	1.1713
Sharpe Ratio	1.4245	2.0134	2.4398	2.5344	2.8832	2.8520

Table 4.9: Different numbers of Features

According to the results above, selecting 20 features instead of the original 31 features can lead to similar excess return and Sharpe Ratio but reduces the running time by about 4.4 minutes. So it is a good idea to keep the main features by using a PCA. The final 20 features are:

$$m \in (1, 2, 3, 4, 5, 7, 9, 10, 12, 15, 18, 20, 40, 60, 80, 100, 140, 160, 200, 220)$$

Chapter 5

Strategy in Chinese Stock Market

5.1 Comparisons among Classification Models

In the U.S. Equity market, investor may choose long or short position according to their own judgment. However, there is only one position in Chinese Stock Market, which means investors can only long stocks. Given this main difference, we slightly change our strategy with only going long the top $2n$ stocks instead of going long the top n stocks and short the bottom n ones, where $n \in (10, 20, 50, 100)$ to adapt to the Chinese Market. Similarly, we set the data from Jan.1st, 2015 to Dec.31st, 2017 as the training set and Jan.1st, 2018 to Dec.31st, 2018 as the test set.

Following the same steps used for the U.S. market, the annualized returns and risk measurement of $2n = 20$ portfolio, before and after transaction costs are shown below:

	Before Fees				*After Fees*			
	DNN	RAF	XGBoost	SVM	DNN	RAF	XGBoost	SVM
Mean Return	0.9637	1.0208	1.3132	1.2124	0.2141	0.3013	0.5912	0.5027
Excess Return	0.8822	0.9487	1.2682	1.1567	0.1567	0.2012	0.5082	0.4234
St.D	0.3331	0.3892	0.3894	0.3846	0.3331	0.3892	0.3894	0.3846
Sharpe Ratio	2.6485	2.2585	3.2585	3.0075	0.4704	0.5169	1.3051	1.1009

Table 5.1: Annualized Return for n=10 (long 20, CSI300)

Same as the results in the S&P500, XGBoost's performance is the best and SVM also shows a good result. Moreover, the results show that we can still generate positive profit

in the Chinese Stock Market with only a long position, but the profit is less than that in the U.S. market.

5.2 Vertical Ensemble Model

Similarly, we use Grid Search with the step-length of 0.1 for giving different weights for $\left(P^S_{t-1|t-2}, P^S_{t|t-1}, P^S_{t+1|t}\right)$. It is worth mentioning that we assume $weight\left(P^S_{t+1|t}\right) > 0.5$ and distribute the other different weights for $\left(P^S_{t-1|t-2}, P^S_{t|t-1}\right)$. After distributing different weights, we tried 20 weights combinations. The best weight combination is $\left(P^S_{t-1|t-2}, P^S_{t|t-1}, P^S_{t+1|t}\right) = (0.2, 0.2, 0.6)$. The excess returns and Sharpe Ratios for different combinations are shown below:

Figure 5.1: Results after fees for different Vertical Ensemble Models(CSI300)

Similarly, we only show the detailed results from using XGBoost and $2n = 20$ below:

	Before Fees		*After Fees*	
	Best Weights	Only t	Best Weights	Only t
Mean Return	1.4623	1.3132	0.9012	0.7882
Excess Return	1.3727	1.2682	0.8227	0.7001
St.D	0.3802	0.3894	0.3802	0.3894
Sharpe Ratio	3.6105	3.2585	2.1637	1.7979

Table 5.2: Best Weights v.s. Only t by XGBoost (2n=20, CSI300)

In order to capture some early insider-trading signals, we apply the vertical ensemble model and are able to improve the results by almost 17.5% in return rates.

5.3 Reduce dimension by PCA

Following the strategy process for the U.S. market, we try to keep the first 5, 10, 12, 15, 20 features according to their importance. Despite that the whole return rates are lower than those in the U.S. market, PCA can increase the running speed substaintially. The results are:

Num. of Features	5	10	12	15	20	31
Running Time/min(s)	6.2	9.1	11.5	12.5	14.1	19.1
Accuracy	0.4997	0.5012	0.5299	0.5963	0.6607	0.6993
Excess Return	0.2278	0.4588	0.6612	0.7288	0.7982	0.8227
Sharpe Ratio	0.7813	1.1265	1.4211	1.6652	2.0049	2.1637

Table 5.3: Results for Different numbers of Features

From the result above, we still choose to keep the first 20 features instead of the original 31 features. In this step, we get a similar excess return and Sharpe Ratio while we are able to reduce the running time by about 5 minutes. The selected 20 features are:

$$m \in (1, 2, 3, 4, 5, 6, 10, 12, 15, 16, 18, 20, 40, 60, 80, 100, 140, 160, 180, 220)$$

5.4 Combination with Regression Model

According to the results from the PCA, we only use 20 features for the purpose of increasing running speed. To adapt this procedure to Chinese Market, we also apply classification

models to select the top 20 stocks, for each trading day, which can be regarded as a 'stock pool'. After getting a 'pool', we use RNN-LSTM model to predict the price for these 20 stocks for the purpose of forecasting the return. When we get 'tomorrow's returns for these 20 stocks', we select top10 according to their return ranks. By doing this, we are able to improve our return rates and reduce the use of capital as follows:

		After Fees
	Only Classification	After Regression
Mean Return	0.8721	1.0001
Excess Return	0.7982	0.9221
St.D	0.3894	0.3885
Sharpe Ratio	2.0049	2.3735

Table 5.4: Only Classification v.s. After Regression (CSI300)

Chapter 6

Conclusion

6.1 Summary

In this essay, we create and improve the investment strategy for the U.S. stock market following the steps shown in the Figure 6.1.

1. Apply four basic models of DNN, RAF, XGB and SVM to get the initial results, including mean return, excess return, standard deviation and Sharpe Ratio.

2. Motivated by the idea of a day trading, we compare the results between the day trading and the normal trading. Since the performance of the day trading is worst than that of the normal trading, we continue to focus on the normal trading in the following steps.

3. Vertical ensemble model and horizontal ensemble model are defined to make creation on labels. The vertical model improves the return rates while horizontal do not show any substantial increased in returns.

4. The combination of the classification models and the regression model helps us generate more accurate predictions and PCA is beneficial at reducing the running time without losing much accuracy.

5. As a result, in the U.S. stock market, we achieve the best annualized excess return of 1.1328 and the best Sharpe Ratio of 2.8832. In the Chinese stock market, the final best excess return is 0.9221 and the according Sharpe Ratio is 2.3735.

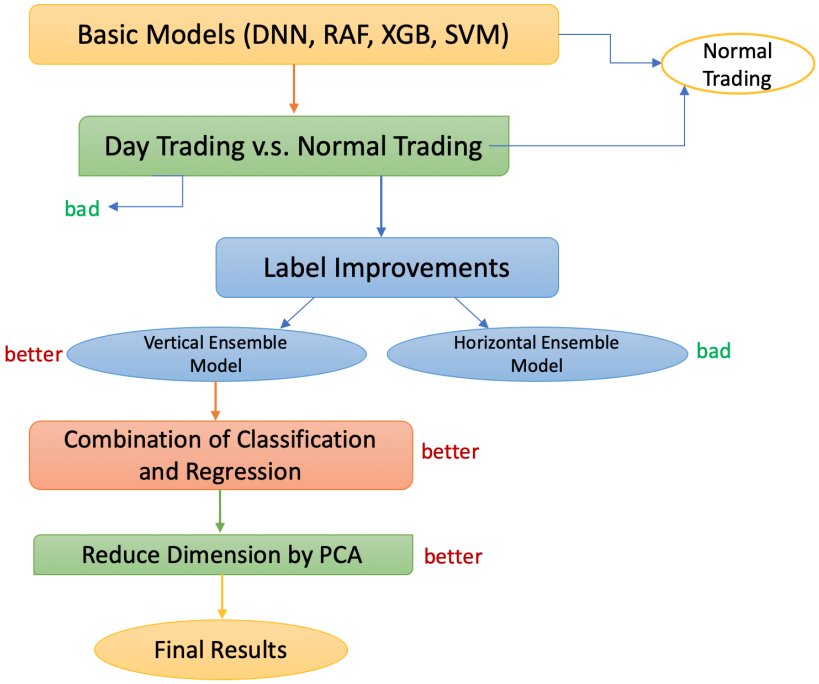

Figure 6.1: Flow Chart for Investment Strategy

6.2 Improvements and Future Work

Compared with the traditional trading strategy or investment strategy, the investment strategy in this essay shows the improvements and innovations in the following aspects:

1. Instead of only using the forecasting probability of the previous trading day, we consider the last three days' probabilities together. By distributing different weights to the last three days, we capture some early fluctuations in the market brought from Insider-Trading, which can help investors to make extra profits. In the U.S. market, the return increases by 12.21% after applying the best weights combination. In the Chinese market, the return increases by 17.51%. Here, the vertical ensemble model is more efficient in the Chinese market.

2. Most of researches about the investment strategy are either based on classification models or regression models. In this paper, we make a combination of classification and regression models, which help us improve the results a lot. After the classification model performs the first screening from the perspective of probability, we apply the regression model to predict the price and perform the second more detailed screening.

3. Due to the slow computing speed of regression models, PCA is used to reduce dimension so that we can save more running time without losing much accuracy.

Since the sample period is limited, we still have some other ideas which are not implemented yet. We will try to achieve the following ideas in the future:

1. This essay only considers the features about lagged returns, which means that we pay more attention to the profitability. It maybe helpful if we add more features which can express some considerations on risk, such as Value at Risk (VaR), shortfall risk and downside risk.

2. In the future, we may collect more data with a longer sample period to test whether this investment strategy is suitable for other financial assets. Otherwise, we will try to adjust our strategy to adapt to different stages of the financial cycle.

3. As for the regression models, we only apply RNN-LSTM in this essay. However, there are some other useful regression machine learning models and statistical functional estimation methods to fit and predict the stock prices. For example, Time-delay recurrent neural network can predict the data with temporal correlations[15]; spline methods can fit data better than some regression machine learning models.

4. The financial asset return often demonstrates volatility clustering. When markets respond to new information with large price movements (volatility), these high-volatility environments tend to endure for a while after that first shock. In other words, when a market suffers a volatile shock, more volatility should be expected. Thus, we can exploit volatility clustering of the returns in order to further improve the accuracy of predictions.

References

[1] Marco Avellaneda and Jeong-Hyun Lee. Statistical arbitrage in the us equities market. *Quantitative Finance*, 10(7):761–782, 2010.

[2] Zsolt Bitvai and Trevor Cohn. Day trading profit maximization with multi-task learning and technical analysis. *Machine Learning*, 101(1-3):187–209, 2015.

[3] 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.

[4] Xianggao Cai, Su Hu, and Xiaola Lin. Feature extraction using restricted boltzmann machine for stock price prediction. In *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, volume 3, pages 80–83. IEEE, 2012.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[6] Robert Culkin and Sanjiv R Das. Machine learning in finance: the case of deep learning for option pricing. *Journal of Investment Management*, 15(4):92–100, 2017.

[7] Victor DeMiguel, Lorenzo Garlappi, and Raman Uppal. Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The review of Financial studies*, 22(5):1915–1953, 2007.

[8] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[9] Matthew Dixon, Diego Klabjan, and Jin Hoon Bang. Implementing deep neural networks for financial market prediction on the intel xeon phi. In *Proceedings of the 8th Workshop on High Performance Computational Finance*, page 6. ACM, 2015.

[10] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.

[11] Eduardo A Gerlein, Martin McGinnity, Ammar Belatreche, and Sonya Coleman. Evaluating machine learning classification for financial trading: An empirical approach. *Expert Systems with Applications*, 54:193–207, 2016.

[12] JB Heaton, Nicholas G Polson, and Jan Hendrik Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

[13] K. Kamijo and T. Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 215–221 vol.1, June 2012.

[14] Ahmad Kazem, Ebrahim Sharifi, Farookh Khadeer Hussain, Morteza Saberi, and Omar Khadeer Hussain. Support vector regression with chaos-based firefly algorithm for stock market price forecasting. *Applied soft computing*, 13(2):947–958, 2013.

[15] Sung-Suk Kim. Time-delay recurrent neural network for temporal correlations and prediction. *Neurocomputing*, 20(1-3):253–263, 1998.

[16] Christopher Krauss, Xuan Anh Do, and Nicolas Huck. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

[17] Christopher Krauss and Johannes Stübinger. Non-linear dependence modelling with bivariate copulas: Statistical arbitrage pairs trading on the s&p 100. *Applied Economics*, 49(52):5352–5369, 2017.

[18] Nagaraj Naik and Biju R Mohan. Study of stock return predictions using recurrent neural networks with lstm. In *International Conference on Engineering Applications of Neural Networks*, pages 453–459. Springer, 2019.

[19] David MQ Nelson, Adriano CM Pereira, and Renato A de Oliveira. Stock market's price movement prediction with lstm neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1419–1426. IEEE, 2017.

[20] João Nobre and Rui Ferreira Neves. Combining principal component analysis, discrete wavelet transform and xgboost to trade in the financial markets. *Expert Systems with Applications*, 125:181–194, 2019.

[21] Mahesh Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[22] Greg Ridgeway. The state of boosting. *Computing Science and Statistics*, pages 172–181, 1999.

[23] Lawrence Takeuchi and Yu-Ying Albert Lee. Applying deep learning to enhance momentum trading strategies in stocks. In *Technical Report*. Stanford University, 2013.

[24] Jie Wang and Jun Wang. Forecasting energy market indices with recurrent neural networks: Case study of crude oil price fluctuations. *Energy*, 102:365–374, 2016.

[25] William K Wang. *Insider trading*. Oxford Press, 2010.