Report on final project of Artificial intelligence

Liu Sichen

516030910528

Abstract

This report is to introduce my recent work on the final project of Artificial intelligence, which is about stock prediction. The basic aim of the project is to predict the average of midprice in future 20 days. And this article contains the brief introduction and the probable solutions I tried with their background knowledge as well as the parameter processing strategies.

1 Introduction

1.1 Background

Nowadays, stock trading has become an important means of financing for major companies. Many people also gain huge profits through stock trading. However, the stock market situation can be said to be changing rapidly, and the stock price fluctuations are also elusive, so the stock price forecast is a very worthwhile question. Economists have summarized many factors affecting stock prices, such as policies and economic situation, through observations of data. In this project, We only consider the factor of buying and selling games. Also, there are many factors in buying and selling games, such as the buying and selling action being implemented and holding money and waiting. But what we focus on is the open trading demand, which is reflected in the order book.

1.2 Data & Information

Since the amount of data of complete order book is very large, the data given by TA is part of it(just about one stock for simplification), and the meaning of all columns of data are as follows:

- —Date— and `Time` represents the time line of the snapshots.
- —MidPrice— is the the aver.
- —LastPrice— is the newest final price.
- —Volume— is the volume of total transaction occurred from some time before til now.

- —BidPrice1— is the highest price that the people bid for the stock.
- —BidVolume1— is the volume that the person who bid the highest price ask for.
- —AskPrice1— is the lowest price that the people who sell at for the stock.
- —AskVolume1— is the volume that the person who sell at the lowest price sell.

And the rule obeyed when updating the values of all variables is that, if the bid price is larger than the ask price, the transaction holds; if the bid volume is more than the ask volume, the last price increases.

1.3 Aim

The test data has the same column variables as the train data, but the difference is that the 10 rows are in a group, because our aim is to predict the average of midprice in future 20 days according to every ten-day group data.

Due to the basic requirement, a very nature idea is that we should build up a model, and use the known data to train the model. Then input the test data, and get the output of the model. The output is final result.

2 Problem Analysis

2.1 Model Selection

Facing this problem, the first idea is to use regression. This can be viewed as a typical multiple linear regression problem, the average of midprice in future 20 days is the y value, and all input columns correspond to $x_1, x_2, ..., x_n$, and train the model by doing regression to get $\beta_1, \beta_2, ..., \beta_n$. Moreover, the model can be expanded to multiple polynomial regression problem to increase the precision. I thought about using other regression model, such as logistic regression. But they are not suitable for this situation. Furthermore, linear regression can be optimized by XGBoost, a model using CART tree.

All the models mentioned above have a problem in common, because time sequential features are not considered. So I was wondering if there is a model that takes the time line of the ten rows of data into consideration. There is a model developed by facebook, whose name is prophet. This model can predict with checking trend changepoints and outliers, but the problem is predict the input and get the output directly without training. So this model can't be adopted by us.

And obviously, this problem can be solved using neural network. Naturally, every ten rows of train data can from a group, and every group is the input of the neural network, and the output is calculated average midprice in future 20 days of the last day in this group. Since the test data also forms ten-day groups, the output of the neural network is our prediction. But most models of neural

networks are not sensitive to time sequence, which is a factor that can't be ignored. RNN(Recurrent Neural Network) and LSTM(Long Short-Term Memory) are two models that are widely used for time sequence prediction. The RNN only considers recent information, but LSTM holds a long term memory. Considering the accuracy, I'd like to adopt LSTM.

And actually, we are expected to complete this work using reinforcement learning or Markov chain. If we are willing to build up a model using reinforcement learning, we should get clear of the four main elements in reinforcement learning, which are agent, environment, action and reward. But I find it difficult to connect the four concepts with the stock prediction work. Since in class, we learned how to predict the weather using hidden Markov model, and we also finished a word segmentation system using this model. Obviously, the prices and volumes are observation, but it's difficult to define the hidden state, as well as connect the hidden state with the value to predict. Also, ordinary problems using this model need discrete states while we have continuous states, and I can't figure out whether convert it to a discrete state problem or just map it to a probability distribution function. So for both of the methods, I came across a bottleneck, and I had to give up them.

2.2 Data Processing

For most of the possible models, if we just input the original data, the results are not satisfactory. So data processing is a very important part of our work. The data can be generally divided into four groups, which are time data, price data, volume data and label data. At first, I just ignored the time data since it has nothing to do with the result. But based on reality, the stock market opens in the morning and the afternoon, and during the rest time, there may be some changes. So all the data groups whose data stretch across the morning and afternoon would better be deleted. The price data is strongly connected with the result, because they are of the same type, and the result is calculated by averaging the midprice column. In different models, they are handled differently, and I will talk about it in every model. By observing the volume data, we can easily get that the Volume increases monotonically while the BidVolume1 and AskVolume1 fluctuate over time, so it's obvious that we can apply first order difference to the Volume data. Since they have a large scale, and varies without a certain law, z-score standardization is a good choice. It's possible to do noting to the label data, but in order to obtain a better result, we take the value of old label data minus last midprice of the group. Finally, all the test data should be processed the same as the train data as long as possible.

2.3 Parameter Tuning

After model selection and data processing, we need to adjust the model to fit the data better, and as a result get a better result. And adjusting models involves parameter tuning, which takes a large amount of time. Since different

models have different parameters, the detailed parameter tuning process will be discussed in every model.

3 Models

3.1 Regression

3.1.1 Model Introduction

The regression analysis mainly solves the following problems, and they are also the steps to follow:

- Determining the mathematical relationship between variables from a set of sample data.
- Estimate or predict the value of another specific variable based on the value of one or several variables using the relationship sought.

When performing regression analysis, it is necessary to determine the independent variables and dependent variables. The dependent variables are predicted or interpreted. One or more variables of the dependent variable used to predict or interpret are called independent variables. For multiple variables with a linear relationship, a linear equation can be used to represent the relationship between them. An equation describing how the dependent variable y depends on the independent variable x and the error term ϵ is called a regression model.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon \tag{1}$$

And we assume that the error term ϵ obey normal distribution, and it's mean value is 0, and then the model can be transformed to a regression equation.

$$E(y) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{2}$$

To get the coefficients of the regression equation, we can adopt the least square method. To get the best coefficients, we are willing to minimize the square sum of residual, which is the subtraction between the true value and the estimated value.

$$Q = \sum (y_i - \check{y}_i)^2 = \sum (y_i - \check{\beta}_0 - \check{\beta}_1 x_1 - \check{\beta}_2 x_2 - ... - \check{\beta}_n x_n)^2 \tag{3}$$

And we can get the standard equation set.

$$\frac{\partial Q}{\partial \beta_i}\Big|_{\beta_i = \check{\beta}_i} = 0 (i = 0, 1, 2..., n) \tag{4}$$

3.1.2 Implementation

First about data processing, I tried to use standardization(all columns used or used, I didn't got the experience mentioned before, and then found a better

model and abandoned it), but I found that the result is better than no standardization, but still far worse than the benchmark. The reason is that actually only prices are linearly related to the average price, while the volume terms not. So I deleted all the volume terms and performed linear regression using all price data, but the result still worse than the benchmark. The I tried to use difference and standardization, the results are not improved.

Then it comes to the code implementation. The file ending with ".csv" is read using pandas, calculate the average of midprice in future 20 days for every day except for the last 20 days. And 10 rows of data forms a group, and put all price values of a group into a vector. The vector represents a set values of x, and the average value calculated for the last day in this group represents the value of y. And the test data perform the same operation as the train data. Since sklearn provides an API for linear regression, just invoke the function LinearRegression() to create a model, then invoke fit() function to train the model, which is calculating the coefficients of the regression equation. At last use predict() function to get the prediction results and output them to a file ending with ".csv".

Since there is no parameter in this model, so there is no parameter tuning part in this model.

3.1.3 Extension

In regression area, there's not only linear regression, but also polynomial regression. And I used a better method combining multiple regression with polynomial regression. The difference is that we perform power operation from power 1 to the max power to all price values in the vector of the group and add the results to the vector. And the final accuracy improves as the max power increase from 1 to 3, while it decreases as the max power increases from 3 to higher value. This implies that higher order polynomials can improve the accuracy, but too high order leads to over-fitting.

3.1.4 Summary

This model is the first model I came across when getting this project, and it's the most natural idea. There are mainly two problems. Firstly, the volume information is eliminated by me, so the result is not sound and the error is larger than other methods. Secondly, as I mentioned before, the time sequence information is not involved in this model. And next I will talk about another model based on linear regression and improves the accuracy a lot.

3.2 XGBoost

3.2.1 Model Introduction

XGBoost is the abbreviation for "Extreme Gradient Boosting", and it's based on the gradient boosted trees. For every model in supervised learning, there is objective function to measure the performance of a model. A very

important fact about the objective functions is that they must always contain two parts: training loss and regularization.

$$Obj(\theta) = L(\theta) + \omega(\theta) \tag{5}$$

L is the loss function in training process, and ω is the regularization term. The common training loss is mean squared error. Regularization term controls the complexity of the model, which helps to avoid over-fitting.

The basic model of XGBosst is tree ensembles, which is a group of classification and regression trees(CART). CART is different with decision trees whose leaves only contains decision value. In CART, every leaf has a real score, which gives us richer interpretations that go beyond classification. Add all the prediction scores of every tree together to get a final score.

$$\check{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{6}$$

K is the number of trees, f is the function of the function space F, and F is the set of all possible CAFTs. And the object function can be written as:

$$Obj(\theta) = \sum_{i}^{n} l(y_i, \check{y}_i) + \sum_{k=1}^{K} \omega(f_k) \tag{7}$$

In this model, we should learn the functions f_i, and every function contains the structure of tree and real score of every leaf node. The strategy used is to optimize all trees that have been studied, and add a new tree every tree. And to solve the problem of our project, only MSE need to be considered as the loss function, and if we expand it using Taylor Formula. ($\check{y}^(t)$ is the prediction value at time t)

$$Obj^(t) = \sum_{i}^{n} [l(y_i, \check{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t) + constant \tag{8}$$

$$g_i = \partial_{\check{y}^{(t-1)}{}_i} l(y_i, \check{y}_i^{(t-1)}) \tag{9}$$

$$h_i = \partial^2_{\check{y}_i^{(t-1)}} l(y_i, \check{y}_i^{(t-1)}) \tag{10}$$

And after deleting all the constant, the target in step t becomes:

$$\sum_{i}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t) \tag{11}$$

The definition of a tree is:

$$f_t(x) = z_{q(x)}, z \in R^T, q : R^d \to \{1, 2, ..., T\} \tag{12}$$

z is the real score vector of the leaf, q is the function assigning each data point to the corresponding leaf, T is the number of leaves. And the complexity is defined as

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{13}$$

The object value can be further simplified as($I_j = \{i|q(x_i) = j\}$):

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i z_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} z_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_j) z_j + \frac{1}{2}(\sum_{i \in I_j} h_j + \lambda) z_j^2] + \gamma T \\ &= \sum_{j=1}^{T} [G_j z_j + \frac{1}{2}(H_j + \lambda) z_j^2] + \gamma T \end{aligned}$$

Since z_j are independent with respect to each other, the term is the sum symbol is quadratic and the best z_j for a given structure q(x) ans the best objective reduction is(the last equation shows how good a tree structure is):

$$\begin{aligned} z_j^* &= -\frac{G_j}{H_j + \lambda} \\ obj^* &= -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \end{aligned} \tag{14}$$

And when splitting a leaf into two leaves the score it gains is:

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma \tag{15}$$

3.2.2 Implementation

First about the data processing, its process is nearly the same as what I mentioned before. The price values are strongly connected with the final prediction results, and the key data of XGBoost doesn't need standardization. Since the volume data are large and varies a lot, using z-score standardization not only narrows the range of the statistics mapping them to [0, 1] interval, but also reduce the impact of the less important data on key data. Considering these two factors, this way to process the data is the best. There is another idea to divide the groups. In previous statement, the step size of group selection is 1, but in this way, the step size becomes 10. It means, groups whose index is not divisible by 10 are eliminated, and only one tenth of the groups are conserved. Since the second way also uses all the data given, it makes sense. And the result proved the method.

Then it comes to the code implementation. Since Reading and processing data process was mentioned in previous contents, I don't Narrative it here. In python, there is a package named XGBoost, and we can just invoke the API to use the model. There is a Scikit-Learn Wrapper interface for XGBoost, and the class name is XGBRegresssor. We just invoke the construction function xgboost.XGBRegressor() of the class to create a model, and also invoke fit() and predict() function to fit and predict the model as what we did in last model. Since the output of the model is not the final prediction value, we need to save the midprice of the last day in every group when processing the test data, and add them to the output of the model. At lsat, output the data into the file ending with ".csv".

The complete code of model creating is: model = xgb.XGBRegressor(max_depth=8, learning_rate=0.1, n_estimators=150, silent=False, objective='reg:linear'). The max_depth parameter represents the max depth of the boosted trees, which characterizes the complexity of the model. Within a certain range of max depth, the accuracy of the model increases as the complication of the model, but too large depth results in over-fitting, so I take 8 as the max_depth(from 5 to 10 are similar, so I took the average). Parameter learning_rate(xgb's "eta") is the boosting learning rate and n_estimators is the number of boosted trees to fit. By decreasing the step size learning rate, the randomness of the model is improved to resist the noise. And if the leaning rate is low, the time when the model converges will come later, so the number of boosted trees should increase. But if the learning rate is too low, the training process is very time consuming, so I chose learning_rate = 0.1 and n_estimators=150. Silent parameter means whether the training details is printed out, and I set it false, which is easy for me to debug. And the objective parameter is the objective function mentioned previously. Since we use XGBoost to optmize the result of linear regression, there's no doubt that I chose "reg:linear" as my objective parameter.

3.2.3 Summary

XGBoost is a model focusing on supervised learning, and in our project, it's used to optimize the linear regression model. Since in linear regression model, using volume data reduces the accuracy of the results, XGBoost model solves this problem and gets a better result than linear regresion model. Besides, using increment prediction considers the time sequence information to some extent but not complete, since only one midprice is considered. But the one considered is the one directly connected with the average, so the result won't be so bad. And actually, the results of this model can easily exceed the benchmark and reach a relatively high accuracy. So this is one of the models that I submitted.

3.3 LSTM

3.3.1 Model Introduction

LSTM is a model based on RNN, so I'd like to briefly talk about the principle of the RNN and expand it to the LSTM model. In the Figure 1, we can see a

module of the neural network, A, is reading an input x_i and output a value h_i. Different from traditional neural networks, there is connection between modules in the same layer, and the connection reflects the time dependency relationship between inputs. More specifically, the information flows from the past inputs to the future inputs.

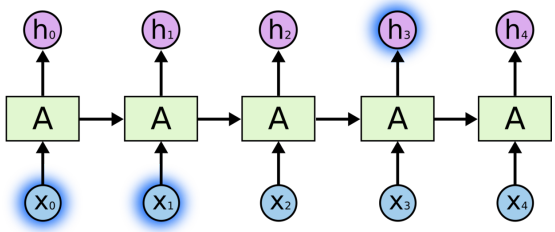

Figure 1: The elationship between modules in RNN

Let's focus on the Figure 2. On the left, it's the structure of RNN. All of the RNN networks have the form of chain of repeating modules of neural network, and in standard RNN, the repeating module A shown in the graph has a very simple structure, which is a tanh layer. The problem is that RNN can only learn the relevant information close to current time, and as the gap grows, RNN becomes unable to connect the past information.

On the contrary, LSTM model, which can be viewed as a special RNN model, is able to learn long-term dependencies, since remembering information for long periods is its default behavior. The structure is also shown in the Figure 3, and compared with the single layer of RNN, LSTM has four layers interacting in a special way.

The key to LSTMs is the cell state, which is the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through", while a value of one means "let everything

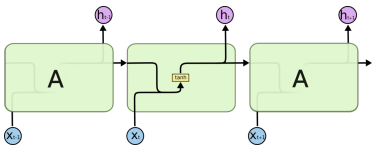

Figure 2: The strcuture of RNN

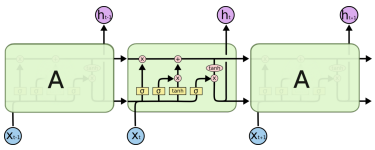

Figure 3: The structure of LSTM

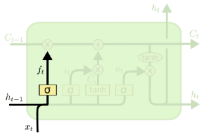

Figure 4: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

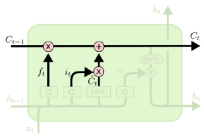

Figure 5: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

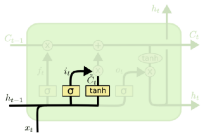

Figure 6: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

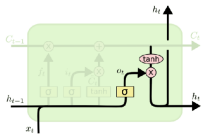

Figure 7: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
$h_t = o_t * tanh(C_t)$

through". An LSTM has three of these gates, to protect and control the cell state.

3.3.2 Implementation

First about the data processing, the processing idea is similar to that of XGBoost, but somwhere different. Since XGBoost is a tree model, main statistics don't need standardization. But using neural networks, standardization is the basic requirements for convergence, so it's indispensable. And as mentioned previously, the column of Volume is monotonically increasing, so one order difference is mandatory, or the accuracy will be very bad. Since neural network is data sensitive, setting a reference value is important. This means we can take the last day of the group as a reference day, and all the values in a column minus the value of the last day, then the scale of data decreases a lot, especially for the price values, the scale changes from 10^0 to $10^{-2}/10^{-3}$. The step size also have two choices as mentioned, one of which is 1 and move group selection batch 1 index per loop, and the other is 10 and no intersection between two groups. And the label of every group also minus the midprice of the last day in each group, because they have the strongest connection.

In code, use pandas to load the data from the csv file as usual and calculate the average midprice to be the label of the train data. Use a loop to process all the 10-day group. In every loop, perform data operation as said before, and get a matrix then add it to the train data set. Also, apply data processing to the test data which is similar to that of train data. Since Keras provides an API for LSTM model implemented basing on tensorflow, invoke function Sequential()

to create a sequential model. Then use function add() four times to add LSTM layers and two fully connected layers. Actually, taking one LSTM layer and one fully connected layer is also right, and the accuracy is similar. Invoke compile() function to set the optimizer, loss and metrics. And use fit() and predict() functions to fit the model and predict the results basing on the model. Since the output of the model is the increment value, we should save the last midprcie of every group, and add it to the result. At last, also display the results to the csv file.

Then it comes to the parameter tuning. In creating the first LSTM layer, my code is model.add(tf.keras.layers.
LSTM(128, input_shape=(train_x.shape[1], train_x.shape[2]), use_bias=True, recurrent_initializer = 'orthogonal', bias_initializer='zeros', unit_forget_bias=True, return_sequences=True)).128 is the number of hidden neurons in every LSTM module. And in the sample code posted in the official document and others' blogs, this parameter is usually 64 or 128, and usually 128 is better than 64, so I chose 128. The other options in this function are all options for the initialization, and I just take the value that most people use. And for the other thress layers, the parameter I used is only the size. Then invoke the compile function to configure the training model, I used model.compile(optimizer='adam', loss='mae', metrics=['accuracy']), which means I chose adam optimizer since its robust to the selection of hyperparameter, mean_absolute_error as the loss function, which is similar to what I applied in XGBoost, accuracy as the standard of evaluation. And in the fit function, the code is model.fit(train_x, train_y, epochs=5, batch_size=128, shuffle=True). the epochs is the time that we train the neural network, and after only 5 epochs, the loss coverges, so I chose 5 as the parameter. The batch size parameter I tried 32, 64, 128, and the change is not obvious though 128 can get the best result. I used shuffle parameter to shuffle the data to resist the effect caused by noise.

3.3.3 Summary

Neural network is a very hot topic recently, it can solve many problems in artificial intelligence area though we don't actually know the mechanism. The problems that linear regression have are both solved using this model, since LSTM is specially applied to time sequence problems. Also, all the data is used in the the model, so the final result is better than linear regression. Compared with XGBoost, the results are similar, both are 148-151 on the public board. But from my point of view, since the relationship is not exactly linear, I think the upper bound of neural networks is higher than regression or improved regression, but I still didn't figure a way to improve the neural network model.