The Hong Kong Polytechnic University

Stock Price Prediction

Course Project for COMP5541

Machine Learning and Data Analytics

Delievered by Group 5

Group Members:

19099089G JI Peng

19090807G SONG Jingru

19093448G HE Xun

December 24,2019

List of contents

1. Abstract

This report is all about exploring the data of stocks and trying to predict the Adjusted Closing Price of the stocks by analyzing the previous Opening price of stock and news related to that stock.

2. Background

2.1. Problem Statement

In this project we are going to build a predictive model using Recurrent Neural network(RNN) made using Long Sort-Term Memory(LSTM) nodes, to predict the closing price of the stock with just the news data and the Opening price of the stock on that particular date, and also try to beat the prediction of a Polynomial Regression model(Benchmark model) which also takes the same data as the input.

The strategy for the solving this problem can be outlined as follows:

1) Process the news data according to the need of the model, as to make the model to understand the news easily we will convert it into sentiment score using Natural Language Toolkit (NLTK).

2) Refine and normalize the complete data set to bring all the data set into a single range.

3) Make and analyze the performance of the benchmark model.

4) Make the final model using RNN and tune the model till the satisfactory results are obtained in all of the 2 models.

To make the benchmark model we used a single perceptron with linear activation and polymerized data set to perform polynomial regression. The RNN model contain LSTM layer to remember the sequence in the data and a set of Dense layer to make effective prediction of the data.

2.2. Metrics

To measure the performance of the model, both benchmark and the RNN model, we have used the final score as the sum of relative error of all the test data points .i.e

score = (| predicted price - actual price | ÷ actual price) . sum()

The smaller the score, the better the model. We have also plotted the data to see that how much they actually differ from the actual price of the stocks and see if the model is able to recognize the pattern or not, these graphs can be seen in the final result representation.

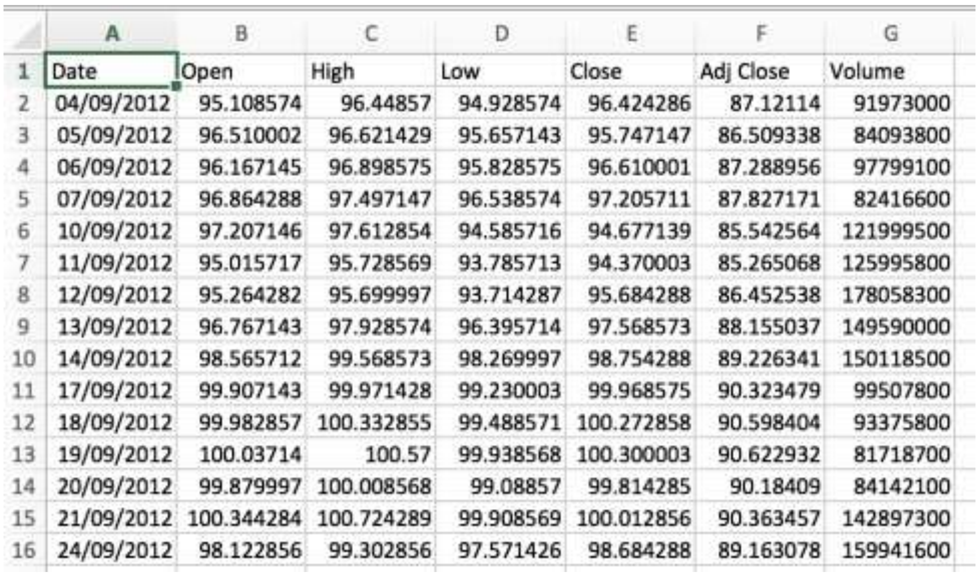
Performance of all the model on the training data can also be observed using the loss and the validation loss graph made by the 'tensorboard', these are shown in the Method section. These graphs were helpful in deciding the final set of parameters like learning rate for the final models and batch size for the training.

3. Analysis

3.1. Data Exploration

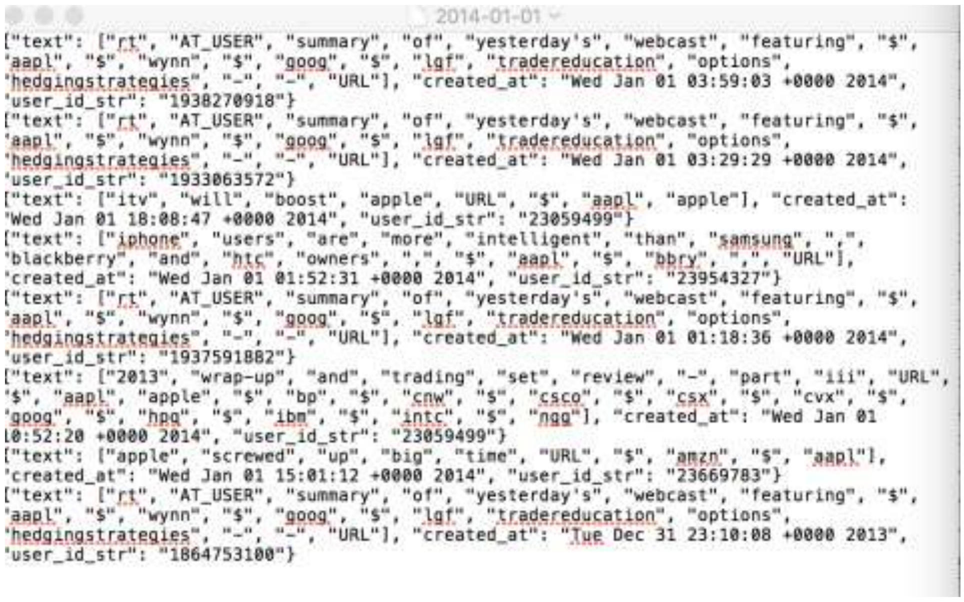
This section and the Exploratory Visualization section will present the finding from the 'exploratory_visualisation.py' file

This project is done using two data sets, stock data and news of the company. The main data of the company stocks is downloaded from the **blackboard of Polyu**. The data is stored in the form of tables inside the respective csv. files of each company. Each table contain 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close' and 'Volume' column of the given stock.Here are the first 16 lines in a row data file for a company's stock data and news data:



	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	04/09/2012	95.108574	96.44857	94.928574	96.424286	87.12114	91973000
3	05/09/2012	96.510002	96.621429	95.657143	95.747147	86.509338	84093800
4	06/09/2012	96.167145	96.898575	95.828575	96.610001	87.288956	97799100
5	07/09/2012	96.864288	97.497147	96.538574	97.205711	87.827171	82416600
6	10/09/2012	97.207146	97.612854	94.585716	94.677139	85.542564	121999500
7	11/09/2012	95.015717	95.728569	93.785713	94.370003	85.265068	125995800
8	12/09/2012	95.264282	95.699997	93.714287	95.684288	86.452538	178058300
9	13/09/2012	96.767143	97.928574	96.395714	97.568573	88.155037	149590000
10	14/09/2012	98.565712	99.568573	98.269997	98.754288	89.226341	150118500
11	17/09/2012	99.907143	99.971428	99.230003	99.968575	90.323479	99507800
12	18/09/2012	99.982857	100.332855	99.488571	100.272858	90.598404	93375800
13	19/09/2012	100.03714	100.57	99.938568	100.300003	90.622932	81718700
14	20/09/2012	99.879997	100.008568	99.08857	99.814285	90.18409	84142100
15	21/09/2012	100.344284	100.724289	99.908569	100.012856	90.363457	142897300
16	24/09/2012	98.122856	99.302856	97.571426	98.684288	89.163078	159941600

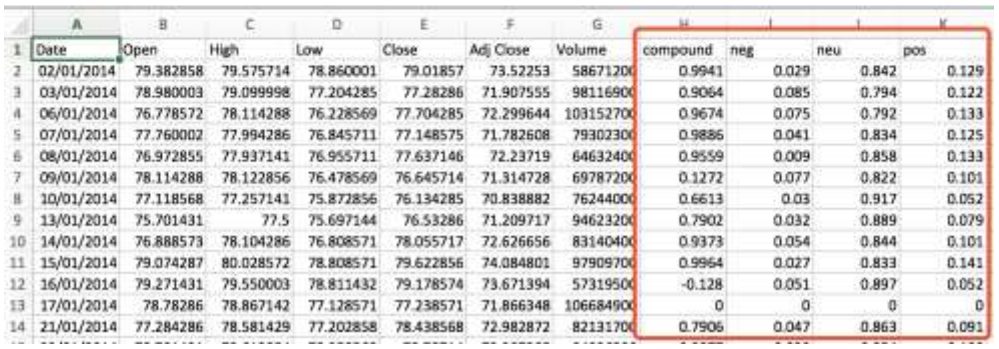
Figure 1stock data



```
{"text": ["rt", "AT_USER", "summary", "of", "yesterday's", "webcast", "featuring", "$",
"aapl", "$", "wynn", "$", "goog", "$", "lgf", "tradereducation", "options",
"hedgingstrategies", "-", "-", "URL"], "created_at": "Wed Jan 01 03:59:03 +0000 2014",
"user_id_str": "1938270918"}
{"text": ["rt", "AT_USER", "summary", "of", "yesterday's", "webcast", "featuring", "$",
"aapl", "$", "wynn", "$", "goog", "$", "lgf", "tradereducation", "options",
"hedgingstrategies", "-", "-", "URL"], "created_at": "Wed Jan 01 03:29:29 +0000 2014",
"user_id_str": "1933063572"}
{"text": ["itv", "will", "boost", "apple", "URL", "$", "aapl", "apple"], "created_at":
"Wed Jan 01 18:08:47 +0000 2014", "user_id_str": "23059499"}
{"text": ["iphone", "users", "are", "more", "intelligent", "than", "samsung", ",",
"blackberry", "and", "htc", "owners", ",", "$", "aapl", "$", "bbry", ",", "URL"],
"created_at": "Wed Jan 01 01:52:31 +0000 2014", "user_id_str": "23954327"}
{"text": ["rt", "AT_USER", "summary", "of", "yesterday's", "webcast", "featuring", "$",
"aapl", "$", "wynn", "$", "goog", "$", "lgf", "tradereducation", "options",
"hedgingstrategies", "-", "-", "URL"], "created_at": "Wed Jan 01 01:18:36 +0000 2014",
"user_id_str": "1937591882"}
{"text": ["2013", "wrap-up", "and", "trading", "set", "review", "-", "part", "iii", "URL",
"$", "aapl", "apple", "$", "bp", "$", "cnw", "$", "csco", "$", "csx", "$", "cvx", "$",
"goog", "$", "hpq", "$", "ibm", "$", "intc", "$", "ngg"], "created_at": "Wed Jan 01
10:52:20 +0000 2014", "user_id_str": "23059499"}
{"text": ["apple", "screwed", "up", "big", "time", "URL", "$", "amzn", "$", "aapl"],
"created_at": "Wed Jan 01 15:01:12 +0000 2014", "user_id_str": "23669783"}
{"text": ["rt", "AT_USER", "summary", "of", "yesterday's", "webcast", "featuring", "$",
"aapl", "$", "wynn", "$", "goog", "$", "lgf", "tradereducation", "options",
"hedgingstrategies", "-", "-", "URL"], "created_at": "Tue Dec 31 23:10:08 +0000 2013",
"user_id_str": "1864753100"}
```

Figure 2 news data

we calculate the sentiment data of the news by using the nltk library and gather the sentiment results of every stock for the respective data from the stock data field, which makes the following final data sets:



	A	B	C	D	E	F	G	H	I	J	K
1	Date	Open	High	Low	Close	Adj Close	Volume	compound	neg	neu	pos
2	02/01/2014	79.382858	79.575714	78.860001	79.01857	73.52253	5867120(	0.9941	0.029	0.842	0.129
3	03/01/2014	78.980003	79.099998	77.204285	77.28286	71.907555	9811690(	0.9064	0.085	0.794	0.122
4	06/01/2014	76.778572	78.114288	76.228569	77.704285	72.299644	10315270(	0.9674	0.075	0.792	0.133
5	07/01/2014	77.760002	77.994286	76.845711	77.148575	71.782608	7930230(	0.9886	0.041	0.834	0.125
6	08/01/2014	76.972855	77.937141	76.955711	77.637146	72.23719	6463240(	0.9559	0.009	0.858	0.133
7	09/01/2014	78.114288	78.122856	76.478569	76.645714	71.314728	6978720(	0.1272	0.077	0.822	0.101
8	10/01/2014	77.118568	77.257141	75.872856	76.134285	70.838882	7624400(	0.6613	0.03	0.917	0.052
9	13/01/2014	75.701431	77.5	75.697144	76.53286	71.209717	9462320(	0.7902	0.032	0.889	0.079
10	14/01/2014	76.888573	78.104286	76.808571	78.055717	72.626656	8314040(	0.9373	0.054	0.844	0.101
11	15/01/2014	79.074287	80.028572	78.808571	79.622856	74.084801	9790970(	0.9964	0.027	0.833	0.141
12	16/01/2014	79.271431	79.550003	78.811432	79.178574	73.671394	5731950(	-0.128	0.051	0.897	0.052
13	17/01/2014	78.78286	78.867142	77.128571	77.238571	71.866348	10668490(	0	0	0	0
14	21/01/2014	77.284286	78.581429	77.202858	78.438568	72.982872	8213170(	0.7906	0.047	0.863	0.091

Figure 3 final combined data (top 14 rows)

The strategy for operating this data can be outlined as follows (the operate details can be find in processData.ipynb)：

1) Iterate all of the ‘r_price_train.csv’ in the ‘raw_price_train’ file and find the data field, through the data value we can find the news in that day
2) Calculates the sentiment data of the news using the nltk library
3) Put each day’s sentiment result into stock data filed

For the final price we used the ‘Adj Price’ for both of the stocks. One important point to note is that, the news data is in range of -1 to 1 but the stock data is not, hence before plotting graphs, normalized the stock data one column at a time, using following formula :

Normalized Data = (Raw data - Min of Range) ÷(Max of Range - Min of Range)

It’s a known fact that these companies have fixed yearly events and they affect their stock values, to make the model easier to understand we also added the month in the final data frame before using it for the training of the model.

3.2. Exploratory Visualization

All the visualization of the data in this project is done by using the “matplotlib” of python. Following are all the graphs in the sequence that we plot to understand the data and the conclusion we made on the basis of them at that time.

Stock’s “Adj Close” for all the companies: From this graph we can see that all of the companies have almost same pattern in the price rise and fall although they have their separate stock price range, this is the reason that we choose to add a new column containing only the month number so that the model can also understand the years pattern in these data.

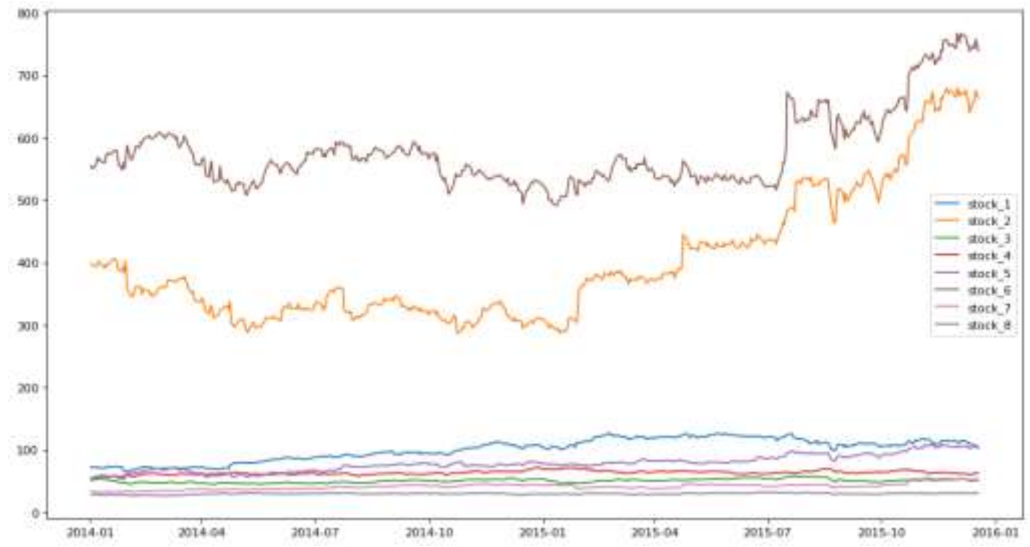


Figure 4 All stock Raw Adj Closing Prices

a. For the further analysis, we used the stock data of number one. Hence the 2nd plot is the combined plot of Open, Close, High and Low, adj

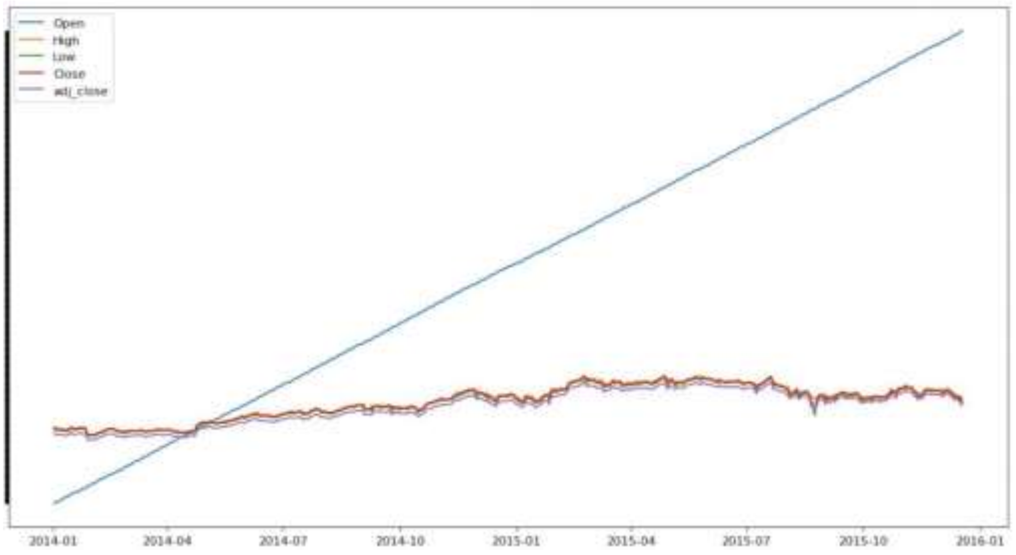


Figure 5 Stock1 data include open price

The Open price may be wrong, so we can see the all the data except Open price.

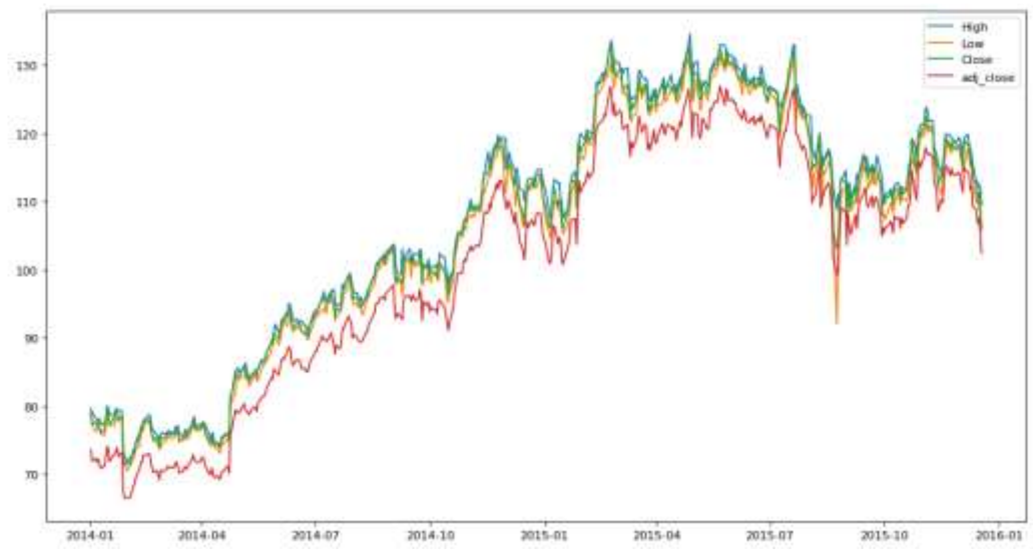


Figure 6 Stock1 data except open price

Visualization of the normalized data to check that the final data still contain all the patterns after the normalization.

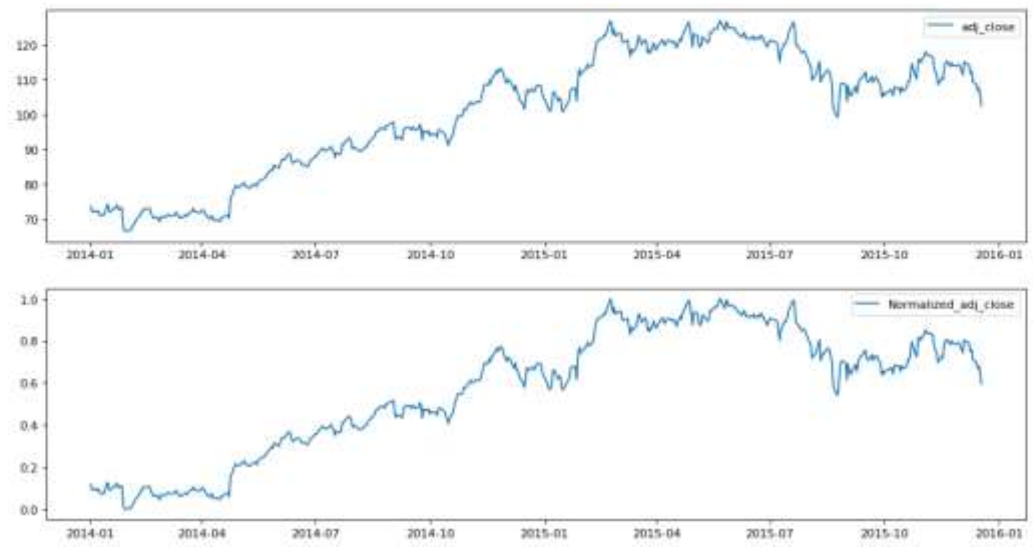


Figure 7 stock1 adj_close and Normalized adj_close Stock data

b. The final data set also contain the sentiment analysis of the news related to all of the companies. In this graph we can see the relation between the rise and fall of the price based on the news in that region. In the following graph blue line is the final normalized stock prices of stock1 and the area fill is made on that graph.

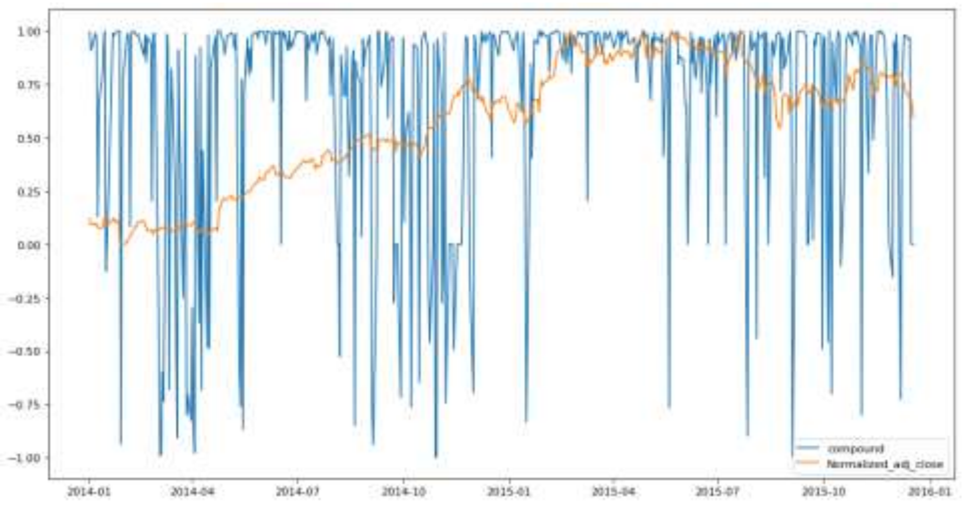


Figure 8 stock1 adj close data (normalized) overlapped by the Compound sentiment value of the news related to stock1

3.3. Algorithms and Techniques

Following are the algorithms for benchmark and final RNN model, that were used:

A. Benchmark

Benchmark model uses the Polynomial Regression to predict the closing price of these stocks. To do so we use a single node single layer neural network, i.e. a perceptron. This perceptron is trained on the same normalized data set but with few more extra features generated through Polymerization.

The perceptron is made using Dense layer of Keras with single output and linear activation function. To optimize this during the training,we use the ‘adam’ optimizer with default settings.

B. Final model

For the final prediction we made 2 different models, all of them are trained on the companies data separately with the news data.

All of the these models contain 3 LSTM layers and 3 Dense layers. The separate prediction models have single node in these output node The LSTM layers are a kind of Recurrent Neural network which work great with the sequential data, and these LSTM layers do not suffer from the gradient vanishing problem like the RNN does with long sequential data sets.

C. Training

This is an interesting data set due to the time sequence, hence we use following 2 different training process:

a. Simple single training:

It a conventional way of training, in which the model if provided with complete data and it trains on the complete data during all the epochs.

b. Framed training

The problem with the above approach is that it sends the complete data to the model and we are trying to predict the closing price of the next seven days. Hence in this framed way we made a data frame of 40 % and a shift size of 10% of the training dataset. Then at the time of training the model 1st get the starting 40% of data and next 10% as validation, after training the frame is shifted forward by a shift length of 10% and the training starts again. This process repeats itself till the frame reaches its end . This framed training gives preference to the latest data more than the older data and by overwriting the weights of the model.

These benchmark models are able to identify the pattern in the abstract manner.i.e they can predict the either the cost will rise or fall, but these are lacking in the accuracy of the data.

4. Method

4.1. Data Processing

Data processing is done on every notebook that uses the stock data. Here are the common steps taken :

1) Arranging Columns

After accessing the “csv” file, we renamed the Adj Close column as Price and removed the Close column. This is helpful in plotting the graphs. After that we created a new field of Month and extracted the month number from the Date column and store it into the Month as an integer, so that they could be used in the training. And finally we arranged the column in way that all the Feature set columns were before the month column and the target column were after the Date.

2) Data Normalization

Values in the column related to the stocks have different range than that of the data in the sentiment columns, so I split the main data into two different data sets, “stock” and “news”. And now we have the stock data into separate set we can implement the following Normalization formula on all of the columns:

stock=(stock-min(stock)) ÷(max(stock)-min(stock))

3) Data Polymerization

After Normalization of the data we polymerized the stock data using the “sklearn” lib . This is done to increase the final feature set that will be used to train the model. For the benchmark the degree of polymerization is 3 and for the main model it is 2.

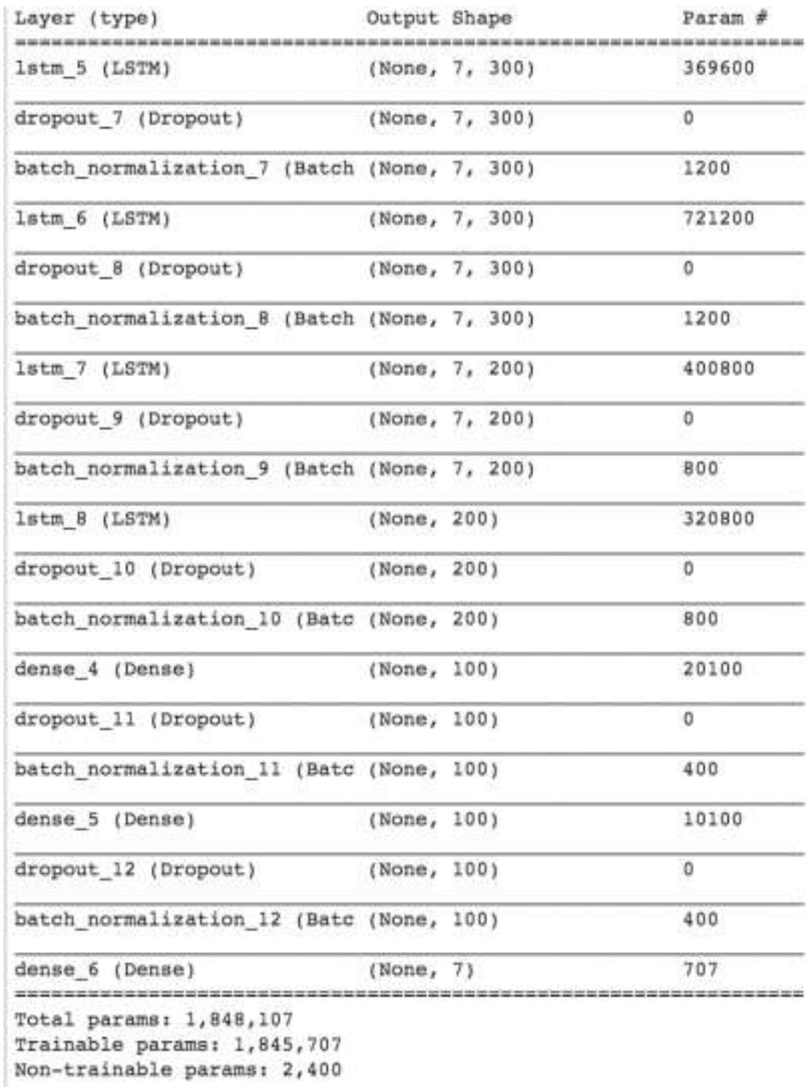
Make the Training and Testing Data: Once the data is completely processed I combined the news and stock data to make the final training and testing sets. The training data set is of 400 rows in length for every model and the testing data have 96 rows in it.

For the training of the main model I have only used the Opening Price of the stock from the stock price and its squared value as the polynomial feature, making the final feature set of 7, two from the stocks and five from news. We develop a function to transform a time series data set for our modal data set.

LSTM in the main model requires one more step after these processing as it accepts a 3D array [samples, time steps, features] and the current data is in 2D array shape [samples, features]. Hence the data was converted before using it into the training of the LSTM layers. In this we do not have to change the dimension of the target data is really important to perform as it causes a lot of errors and due to just one reason.

4.2. Implementation

The RNN tarin by simple model that is implemented is for the #1 stock, and it is presented in the “rnn_simple_train.ipynb” notebook. This notebook 1st reads and preprocess the stock#1 data as described in the above data processing section. This model contain 3 LSTM layer where 1st two layers has return_sequence = True, and the 3rd layer with return_sequence = False as after that there are 3 Dense layer and one final output layer of one node. All the layers have ‘relu’ as their activation function and are followed by a pair of Dropout layer and a Batch Normalization layer to have a generalized output . Hence the final summary looks like this:



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 7, 300)            369600
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 300)            0
_________________________________________________________________
batch_normalization_7 (Batch (None, 7, 300)            1200
_________________________________________________________________
lstm_6 (LSTM)                (None, 7, 300)            721200
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 300)            0
_________________________________________________________________
batch_normalization_8 (Batch (None, 7, 300)            1200
_________________________________________________________________
lstm_7 (LSTM)                (None, 7, 200)            400800
_________________________________________________________________
dropout_9 (Dropout)          (None, 7, 200)            0
_________________________________________________________________
batch_normalization_9 (Batch (None, 7, 200)            800
_________________________________________________________________
lstm_8 (LSTM)                (None, 200)               320800
_________________________________________________________________
dropout_10 (Dropout)         (None, 200)               0
_________________________________________________________________
batch_normalization_10 (Batc (None, 200)               800
_________________________________________________________________
dense_4 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_11 (Dropout)         (None, 100)               0
_________________________________________________________________
batch_normalization_11 (Batc (None, 100)               400
_________________________________________________________________
dense_5 (Dense)              (None, 100)               10100
_________________________________________________________________
dropout_12 (Dropout)         (None, 100)               0
_________________________________________________________________
batch_normalization_12 (Batc (None, 100)               400
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 707
=================================================================
Total params: 1,848,107
Trainable params: 1,845,707
Non-trainable params: 2,400
```

Figure 9 Model summary

For the debugging I am also using the tensorboard callback to generate the log file of the model during the training and ModelCheckpoint to store the weights of the model when it has minimum validation loss, just to skip the overfitting, if occurs. Initially RMSprop optimizer is user with 0.001 learning rate is used. Then the model is trained using model.fit(). Initially I trained it for 30 to 40 epochs and I started with the batch size of 10. We kept the verbose = 1 so that I can see where it got the minimum loss in validation. This setup produced the score of 5.98 with #1 stock data and prediction looks like:

predict_seven_days : [100.6 101.6 109.7 110.4 110.4 110.7 110.8]

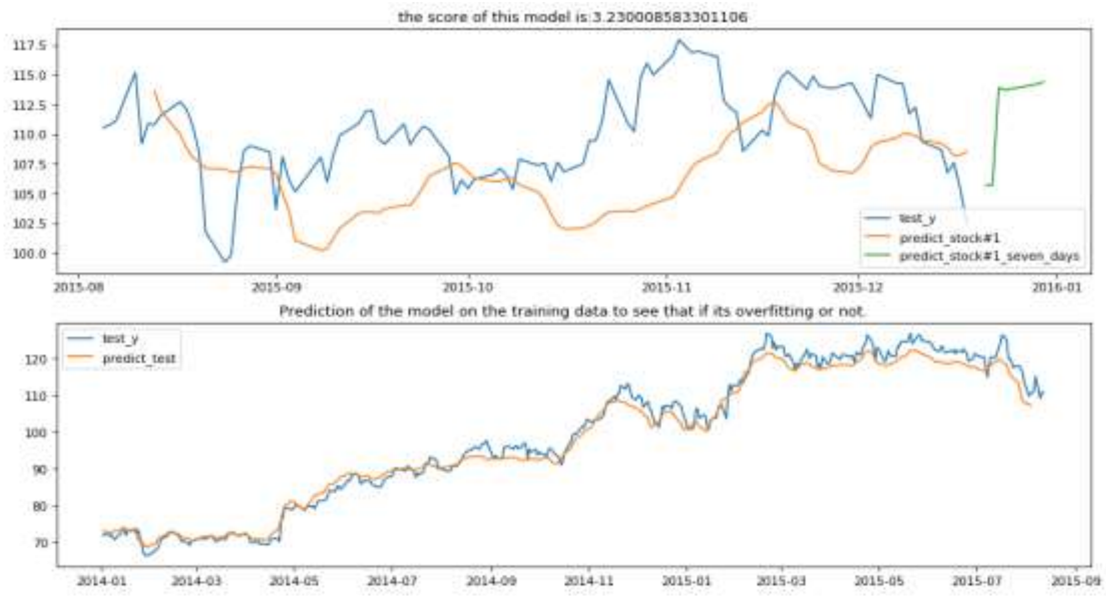
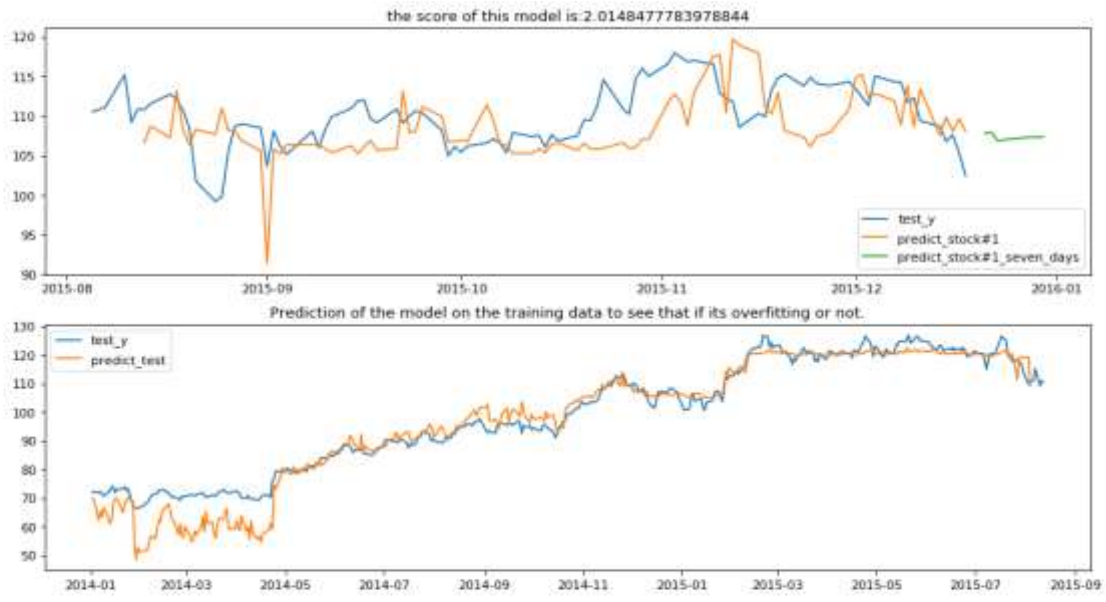


Figure 10 Prediction by Baseline Structure implementation on stock#1 data

The RNN tarin by framed model for stock#1 as follow:

#1[107.8 107.9 106.8 107.   107.3 107.3 107.4]



The model is able to predict the sequential pattern in the stock data, it just has to improve the final price values, hence it need modification in the Dense layers. This model was also implemented with the rest 7 stocks, These results are also similar to the prediction of the stock#1 as in this they are able to predicted yearly pattern but not able to exactly calculate the price.

4.3. Refinement

First thing that can be seen from the prediction graph is that the model is not able to calculate the price close to the actual Price, hence I have to improve the Dense layers. To test I started with the higher values like 500/200/100 nodes in three consecutive layers. Results from this structure was better than the baseline but it was still suffering from the overfitting as if the data was out of the range of the training data it was not performing well.

We also tested the model with different training setups, that is with different size of batch and epochs. It was taking not able to identify the general pattern with smaller batch size hence to improve the performance I increased the batch size and tested the model with batch size near 50. And to decide the number of epoch we trained the model with 10 epochs repeatedly till it started overfitting and noted down the final sets. For most of the parameter sets, it took nearly 60 to 80 epoch with batch size near 50 to get the model to overfit.

For the sake of exploration I also tested the model with different optimizers like Adam and Adadelta, but the RMSprop had the best results. For the error function I also tried “mean_absolute_error” and “mean_square_error”. For the training I also tested the model with the following ways:

1) Simple training :

in this all the data is provided to the model at the same time.

2) Framed training :

in this a data frame is moved from one end of the data set to the other end and the model is trained on each frame with the weights of the previous frame. This process also created a new window for more exploration as what should be the frame size and the shift size to shift the frame. I started with 60% frame size and 10% shift size.

At the time of testing different parameters we first trained the model with simple training and if the parameter set showed improvement, then we used the framed training with the same parameter set. At the end of tuning the RNN model got 5.66 on stock#1, on simple training and 200/100/50 nodes in Dense layer.

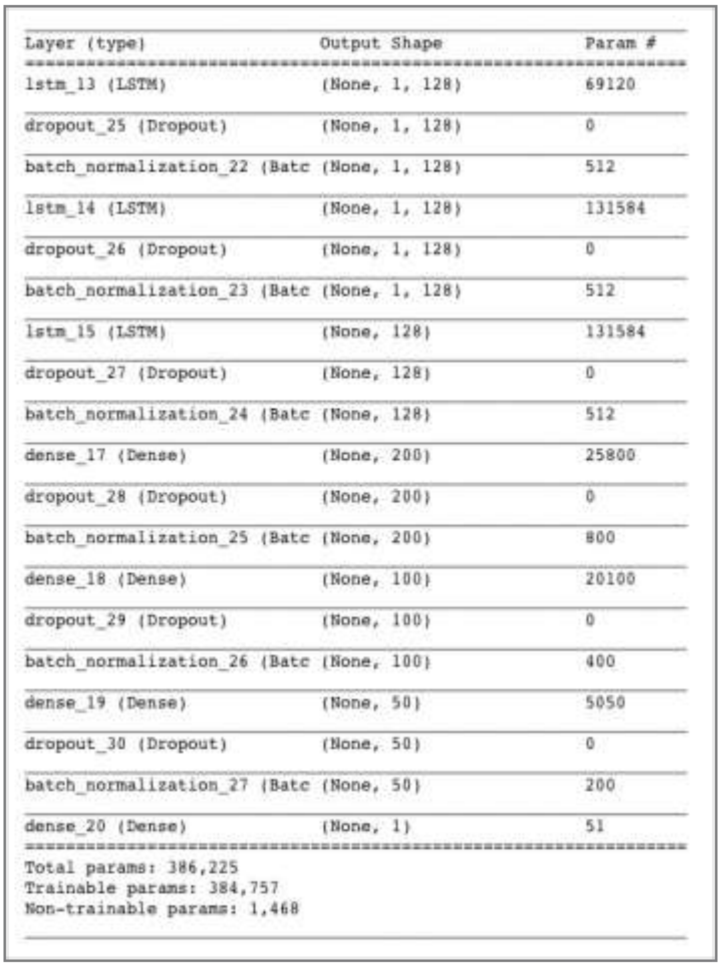
5. Results

5.1. Model Evaluation and refinement

Following are the final set of parameters and model structure that we used:

5.1.1. Model Parameter:

This model have 3 LSTM layers, 3 Dense layer and 1 final output node. Each layer is followed by a combination of Dropout layer and BathNormalization layer. I used ‘relu’ activation function for every layer, except the last output layer, it docent have one. Final structure of the model is as follows:



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 1, 128)            69120
_________________________________________________________________
dropout_25 (Dropout)         (None, 1, 128)            0
_________________________________________________________________
batch_normalization_22 (Batc (None, 1, 128)            512
_________________________________________________________________
lstm_14 (LSTM)               (None, 1, 128)            131584
_________________________________________________________________
dropout_26 (Dropout)         (None, 1, 128)            0
_________________________________________________________________
batch_normalization_23 (Batc (None, 1, 128)            512
_________________________________________________________________
lstm_15 (LSTM)               (None, 128)               131584
_________________________________________________________________
dropout_27 (Dropout)         (None, 128)               0
_________________________________________________________________
batch_normalization_24 (Batc (None, 128)               512
_________________________________________________________________
dense_17 (Dense)             (None, 200)               25800
_________________________________________________________________
dropout_28 (Dropout)         (None, 200)               0
_________________________________________________________________
batch_normalization_25 (Batc (None, 200)               800
_________________________________________________________________
dense_18 (Dense)             (None, 100)               20100
_________________________________________________________________
dropout_29 (Dropout)         (None, 100)               0
_________________________________________________________________
batch_normalization_26 (Batc (None, 100)               400
_________________________________________________________________
dense_19 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_30 (Dropout)         (None, 50)                0
_________________________________________________________________
batch_normalization_27 (Batc (None, 50)                200
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 51
=================================================================
Total params: 386,225
Trainable params: 384,757
Non-trainable params: 1,468
_________________________________________________________________
```

Figure 11 Structure of the final Model

5.1.2. Model Compilation:

Final mode is using RMSprop as its optimizer with learning rate of 0.0005, I left the remaining parameter to default value .i.e rho=0.9 and decay=0. It uses " mean_squared_error " as its final loss function, with 'accuracy' as metrics.

5.1.3. Training:

To save the data log of the training I have used the TensorBoard and ModelCheckpoint. And the remaining parameters are as follows:

For stock#1

Simple training :

Training X= (400,7), .i.e complete data set

Training Y = (400,1) epoch=80 Batch

Size=50

Validation split= 0.1 .i.e 10%

For stock#1

Framed training:

Training X = (60% of 400,7), this frame id shifted with 20% shift size after every training of 30 epochs.

Training Y = Same as the X .i.e (40% of 400,1)

epoch=30 for all frames and 20 for the extra end frame

Batchsize=80

Validation split=0.1, .i.e 10% of the current frame

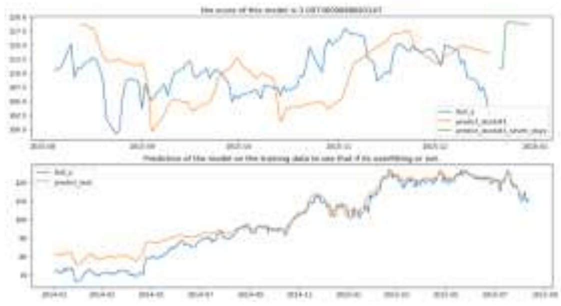
6. Conclusion

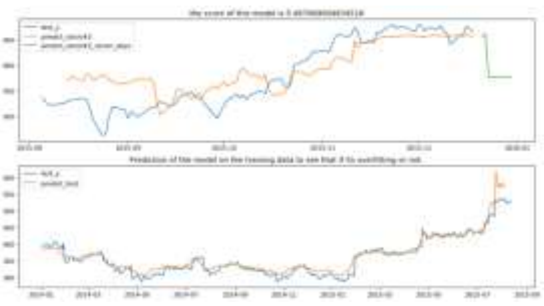
6.1. RNN Model result

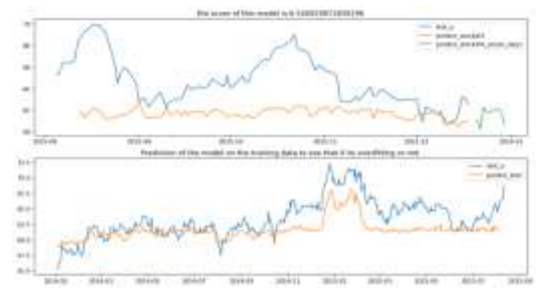
Following are the final result with RNN models from #1 to #8 Stock:

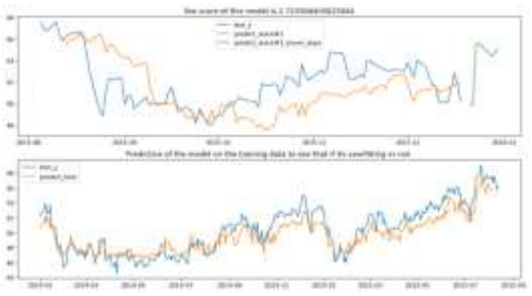
6.1.1. Simple training:

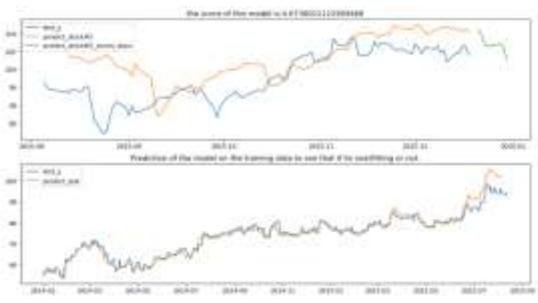
```
stock#1[110.8 110.9 118.7 119.1 118.8 118.5 118.7]
stock#2[655.7 660.9 577.2 577.8 577.8 575.2 578.7]
stock#3[49.9 49.9 55.5 55.6 54.4 54.7 55.1]
stock#4[61.1 60.2 62.3 61.8 62.1 61.8
stock#5[111.2 109.6 106.9 106.4 107.2 106.1 102.9]
stock#6[876.1 872.8 705.3 709.1 702.   715.5 704.3]
stock#7[55.9 57.5 43.1 43.   43.1 43.   43.1]
stock#8[29.8 29.8 30.1 30.5 30.4 30.2 30.4]
```

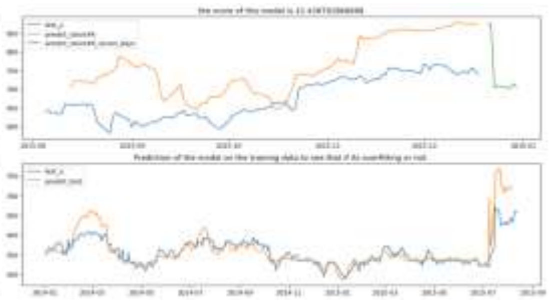

stock1

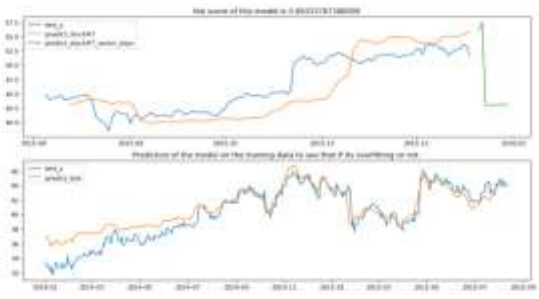

stock2

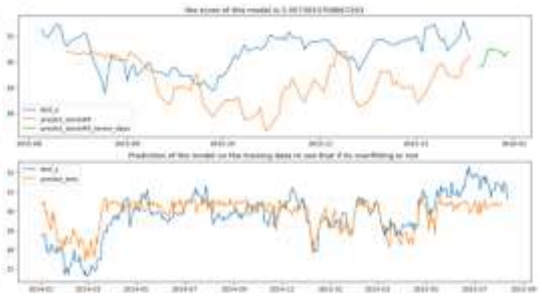

stock3


stock4


stock5


stock6

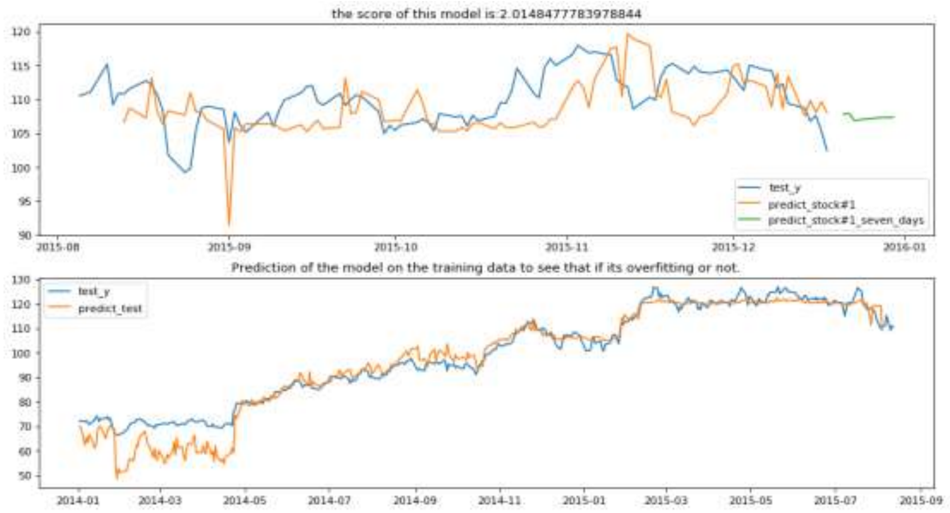

stock7


stock8

we can clearly see the effect of using LSTM layers as they have a significant improvement. And the model didn’t performed as expected, but they also have better score than the benchmark and have good prediction graph also in comparison to benchmark.

6.1.2. Framed Training:

stock#1[107.8 107.9 106.8 107.   107.3 107.3 107.4



We can clearly see the improvement in the model by using the framed training, it is now able to predict the price even more accurately. (It is great pity for us we do not have much time to train the model from stock#2 to stock#8)

6.2. Improvement

This is really a very wast field and even after this much of work I really believe that we have just scratched the surface of the field. There are limitless amount of data to explore and hence a lot of places to implement the machine learning.

For this specific project there are many improvements that can be made specifically in the model, as I expected framed training is work better than the simple training model. In the terms Machine learning we have just used the supervised learning, but we can make complex models using Reinforcement learning to make the model predict even more better and in near future.

7. References

[1] Kohara, K., Ishikawa, T., Fukuhara, Y., & Nakamura, Y. (1997). Stock price prediction using prior knowledge and neural networks. Intelligent systems in accounting, finance and management, 6(1), 11-22.

[2] Padhiary, P. K., & Mishra, A. P. (2011). Development of improved artificial neural network model for stock market prediction.International Journal of Engineering Science.

[3] Jeon, S., Hong, B., & Chang, V. (2018). Pattern graph tracking-based stock price prediction using big data. Future Generation Computer Systems, 80, 171-187.

[4] Lee, M. S., Ahn, C. H., Kwahk, K. Y., & Ahn, H. (2018). Stock Market Prediction Using Convolutional Neural Network That Learns from a Graph. World Academy of Science, Engineering and Technology, International Journal of Business and Economics Engineering, 5(1).

[5] TORRES, D. G., & QIU, H. (2018). Applying Recurrent Neural Networks for Multivariate Time Series Forecasting of Volatile Financial Data.

8. Job Division

JI Peng: Data processing code, RNN_simple_train code, related report
SONG Jingru: RNN_framed_train code, related report
HE Xun: benchmark_model code, related report