MLND Capston Report

S&P 500 Predictor

Yichao Zhang

Oct, 2018

Definition

Project Overview

Inverstment companies use financial model to predict stocks. Some useful feature informations are hidden behind the historical data and other relative performance. However, stock price is not explicitly depend on these features, the prediction problem has never been well solved by any current math model.

Machine learning has the potential to create a complex statistical model for stock price prediction.

S&P 500 index (https://en.wikipedia.org/wiki/S%26P_500_Index)(Standard & Poor's 500) is a very important index in stock trading market. It reflects the bahavior of U.S. stock market and has been widely used for financial analysis.

Problem Statement

In this project, we will create and train a Linear Regression model and two LSTM (https://en.wikipedia.org/wiki/Long_short-term_memory) (Long Short-Term Memory) models. We use these models to predict the closing price of S&P 500 index (https://en.wikipedia.org/wiki/S%26P_500_Index).

Our data comes from Yahoo Finance (https://finance.yahoo.com/quote/%5EGSPC/), we download the data of recent 20 years (from 1998/08 to 2018/07) for training and testing.

Each data instance of one day includes: Date, Open, High, Low, Close, Adj Close and Volume. Adj Close (https://www.investopedia.com/terms/a/adjusted_closing_price.asp) is Adjusted Closing Price, and will be eliminated if it is all the same with Closing Price. For each instance:

- Input:
  - Open, High, Low, Close, Adj Close and Volume
- Output:
  - The Closing price in the next day, that means the data['Close'] shifted by -1 day.

Metrics

The error of each model will be represented by Root-mean-square deviation (RMSE (https://en.wikipedia.org/wiki/Root-mean-square_deviation)), which is the square root of the mean of squared errors.

We use RMSE to compare the performance of different models.

Analysis

Data Exploration, Visualization

The data has been downloaded from Yahoo Finance (https://finance.yahoo.com/quote/%5EGSPC/), and saved in the path: 'data/SP500_199808_201808.csv'.

Each data instance of one day includes: Date, Open, High, Low, Close, Adj Close and Volume. As shown below:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1998-08-03	1120.670044	1121.790039	1110.390015	1112.439941	1112.439941	620400000
1	1998-08-04	1112.439941	1119.729980	1071.819946	1072.119995	1072.119995	852600000
2	1998-08-05	1072.119995	1084.800049	1057.349976	1081.430054	1081.430054	851600000
3	1998-08-06	1081.430054	1090.949951	1074.939941	1089.630005	1089.630005	768400000
4	1998-08-07	1089.630005	1102.540039	1084.719971	1089.449951	1089.449951	759100000

	Date	Open	High	Low	Close	Adj Close	Volume
0	1998-08-03	1120.670044	1121.790039	1110.390015	1112.439941	1112.439941	620400000
1	1998-08-04	1112.439941	1119.729980	1071.819946	1072.119995	1072.119995	852600000
2	1998-08-05	1072.119995	1084.800049	1057.349976	1081.430054	1081.430054	851600000
3	1998-08-06	1081.430054	1090.949951	1074.939941	1089.630005	1089.630005	768400000
4	1998-08-07	1089.630005	1102.540039	1084.719971	1089.449951	1089.449951	759100000

Now data['Date'] is in string format, so we use pandas.to_datetime() function convert it into datetime format.

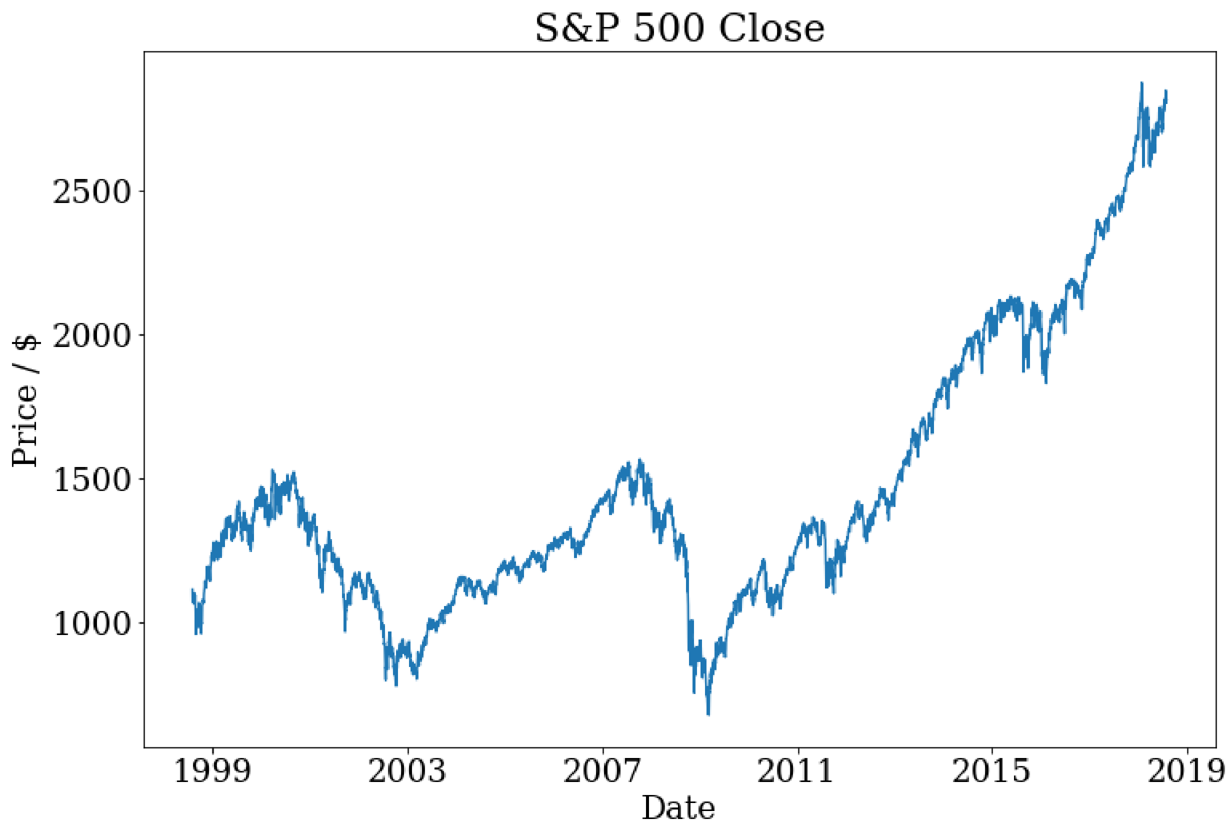
We find that data['Adj Close'] and data['Close'] are exactly the same. Below is the summation of the abs difference between data['Adj Close'] and data['Close']:

```
Out[2]: 0.0
```

So we drop the data['Adj Close'] column.

	Date	Open	High	Low	Close	Volume
0	1998-08-03	1120.670044	1121.790039	1110.390015	1112.439941	620400000
1	1998-08-04	1112.439941	1119.729980	1071.819946	1072.119995	852600000
2	1998-08-05	1072.119995	1084.800049	1057.349976	1081.430054	851600000
3	1998-08-06	1081.430054	1090.949951	1074.939941	1089.630005	768400000
4	1998-08-07	1089.630005	1102.540039	1084.719971	1089.449951	759100000

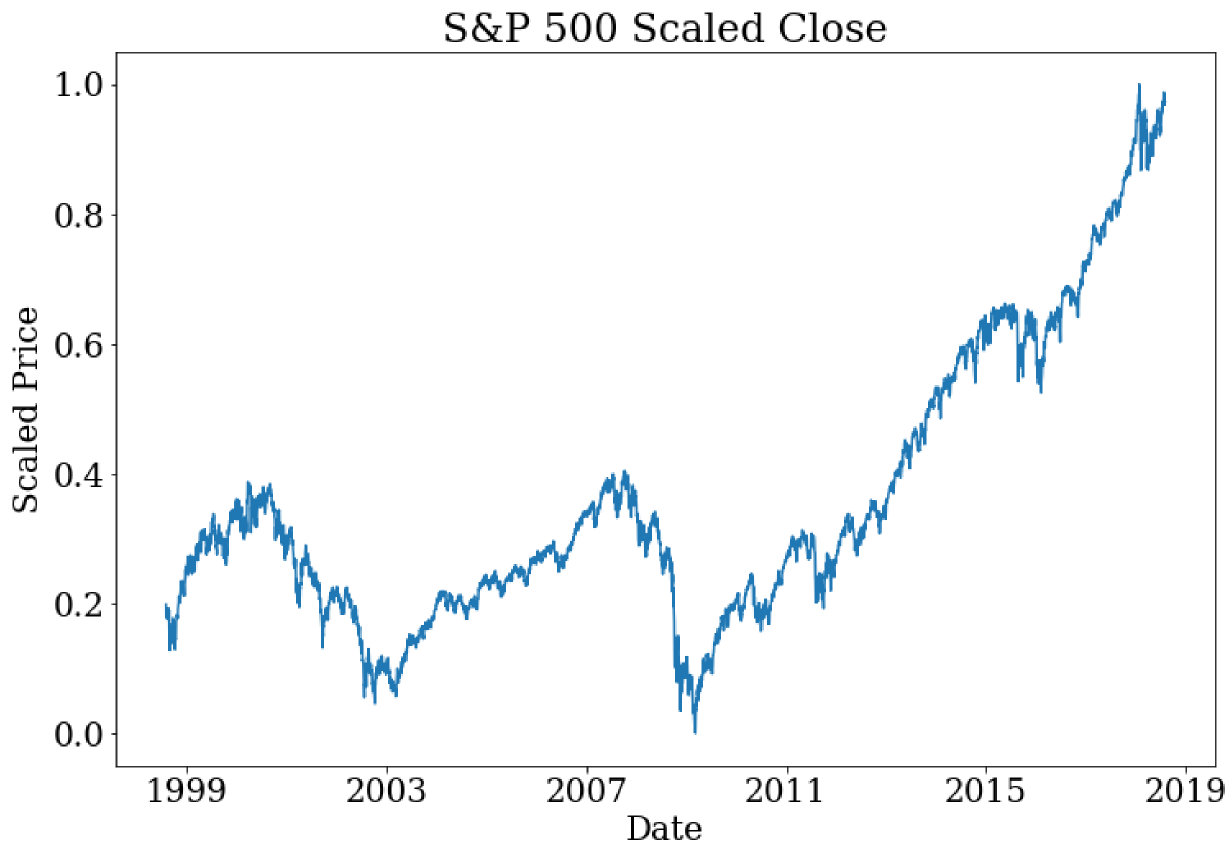
The original closing price of S&P 500 in the recent 20 years is shown in graph below:



The data is in a wide range. In order to get a better performance in machine learning, we scale the data in these columns: ['Open', 'High', 'Low', 'Close', 'Volume'], as shown below:

	Date	Open	High	Low	Close	Volume
0	1998-08-03	0.201737	0.195867	0.203049	0.198471	0.033314
1	1998-08-04	0.197975	0.194921	0.185395	0.180113	0.054029
2	1998-08-05	0.179547	0.178880	0.178771	0.184352	0.053939
3	1998-08-06	0.183802	0.181705	0.186823	0.188086	0.046517
4	1998-08-07	0.187550	0.187027	0.191299	0.188004	0.045687

Plot the scaled data, we can find that all the Closing Prices are located in the range of [0,1]:



We save the scaled data in 'data/SP500_199808_201808_scaled.csv' for further using.

Algorithms and Techniques

In this projest, we will use two regression models in supervised learning algorithms: Linear Regression and LSTM (https://en.wikipedia.org/wiki/Long_short-term_memory). We will compare the performance of these two models and choose the better one.

Benchmark:

Linear Regression Model

We will create the Linear Regression model as a benchmark. Let's import the scaled data from 'data/SP500_199808_201808_scaled.csv'.

Out[12]:

	Date	Open	High	Low	Close	Volume
0	1998-08-03	0.201737	0.195867	0.203049	0.198471	0.033314
1	1998-08-04	0.197975	0.194921	0.185395	0.180113	0.054029
2	1998-08-05	0.179547	0.178880	0.178771	0.184352	0.053939
3	1998-08-06	0.183802	0.181705	0.186823	0.188086	0.046517
4	1998-08-07	0.187550	0.187027	0.191299	0.188004	0.045687

To make a simple Linear Regression model, we select two columns from data:

```
data['Date'], data['Close']
```

Since data['Date'] is in datetime format, so we create a linear array, from 0 to len(data['Date']), to represent the Date.

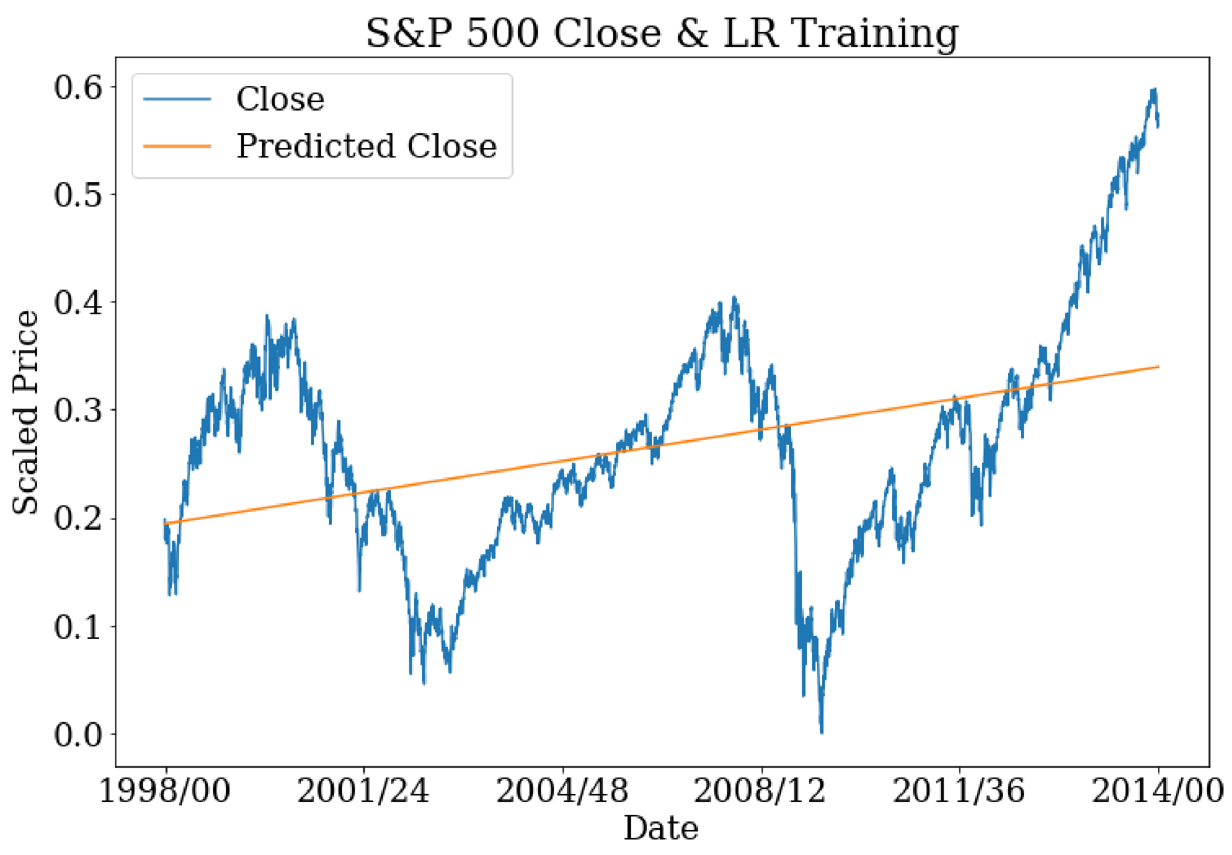
We split the datesets into training dataset and testing dataset. The last 1000 instances are for testing, and others are for training.

And we copy the data['Date'] for ploting later.

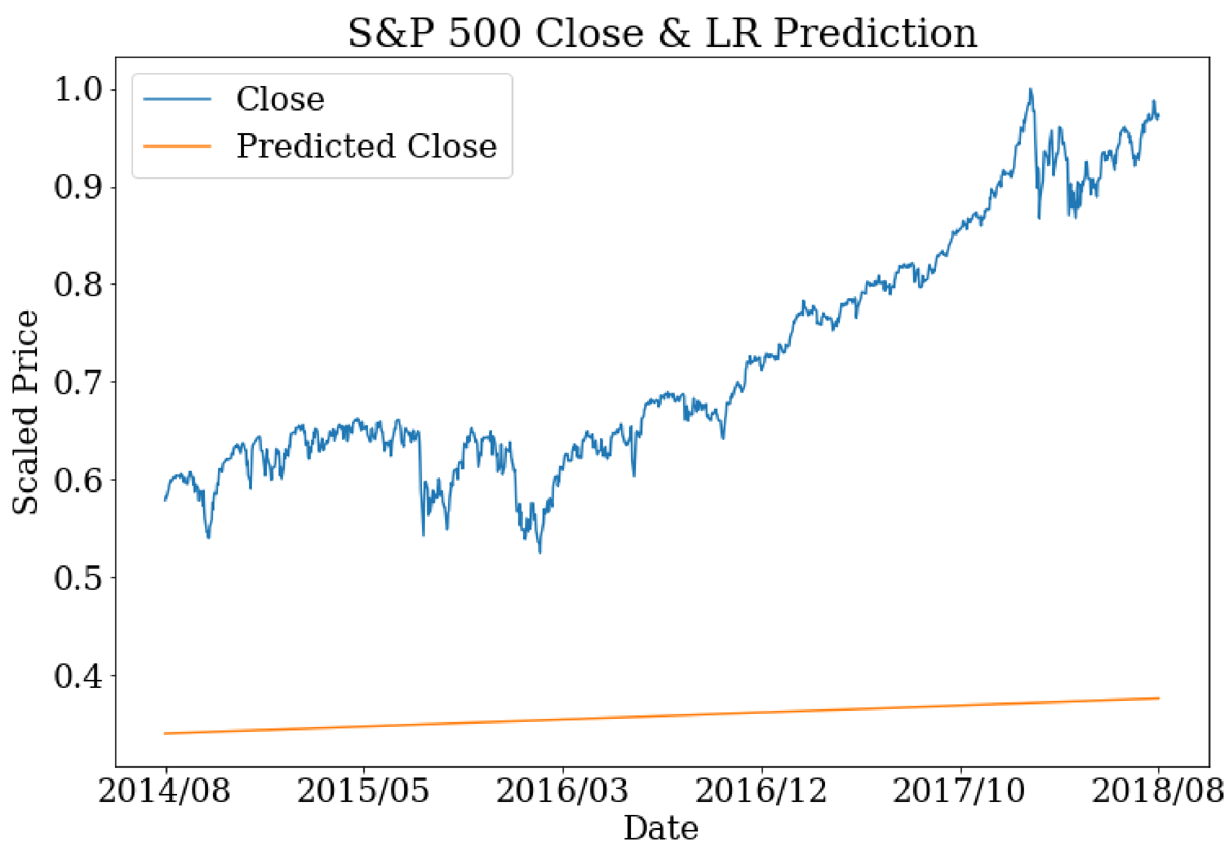
```
Size of X_train (4033, 1)
Size of y_train (4033,)
Size of X_test (1000, 1)
Size of y_test (1000,)
```

From sklearn, we import sklearn.linear_model as our model, and fit the model on our training dataset. Then use this model to predict the targets value in the testing dataset.

Plot the fitting result on the training dataset:



Plot the prediction result on the testing dataset:



The prediction result is not good in the graph. We can see the RMSE performance is vary bad.

```
RMSE on training dataset:  0.116004
RMSE on testing dataset:  0.383239

Original RMSE on testing dataset:  841.723179
```

The RMSE on the un-scaled original data is unbelievablly large, about 1/3 of the highest price value.

In conclusion, a simple linear regression is not valid in this problem.

Metholodogy

LSTM Model

In these Chapter, we will build a LSTM (https://en.wikipedia.org/wiki/Long_short-term_memory) (Long Short-Term Memory) model. And then try to improve it to get a better prediction.

Data Preprocessing

In LSTM model, the input is no longer a single instance in one day.

For each instance here, we will include the date of recent 50 days as a input, that means each input is a data matrix.

The Closing Price data['Close'] in the next day is the target.

We also copy the data['Date'] for ploting later.

```
Shape of the input dataset:
(4983, 50, 5)
Shape of the output dataset:
(4983, 1)
```

We split the datesets into training dataset and testing dataset. The last 1000 instances are for testing, and others are for training.

```
Size of X_train (3983, 50, 5)
Size of y_train (3983, 1)
Size of X_test (1000, 50, 5)
Size of y_test (1000, 1)
```

Implementation

Now we use keras to build a LSTM model.

Parameters:

- input_shape is based on the size of input instance
- unites is the how many days included in each input instance, here is 50
- batch_size = 100
- epochs = 5

Layers:

- Dropout(0.2)
- LSTM(128)
- Dropout(0.2)
- Dense(units = 1)
- Activation('linear')

Compile the model based on the size of input instance.

Then we train the LSTM model on our training dataset for 5 epoches.

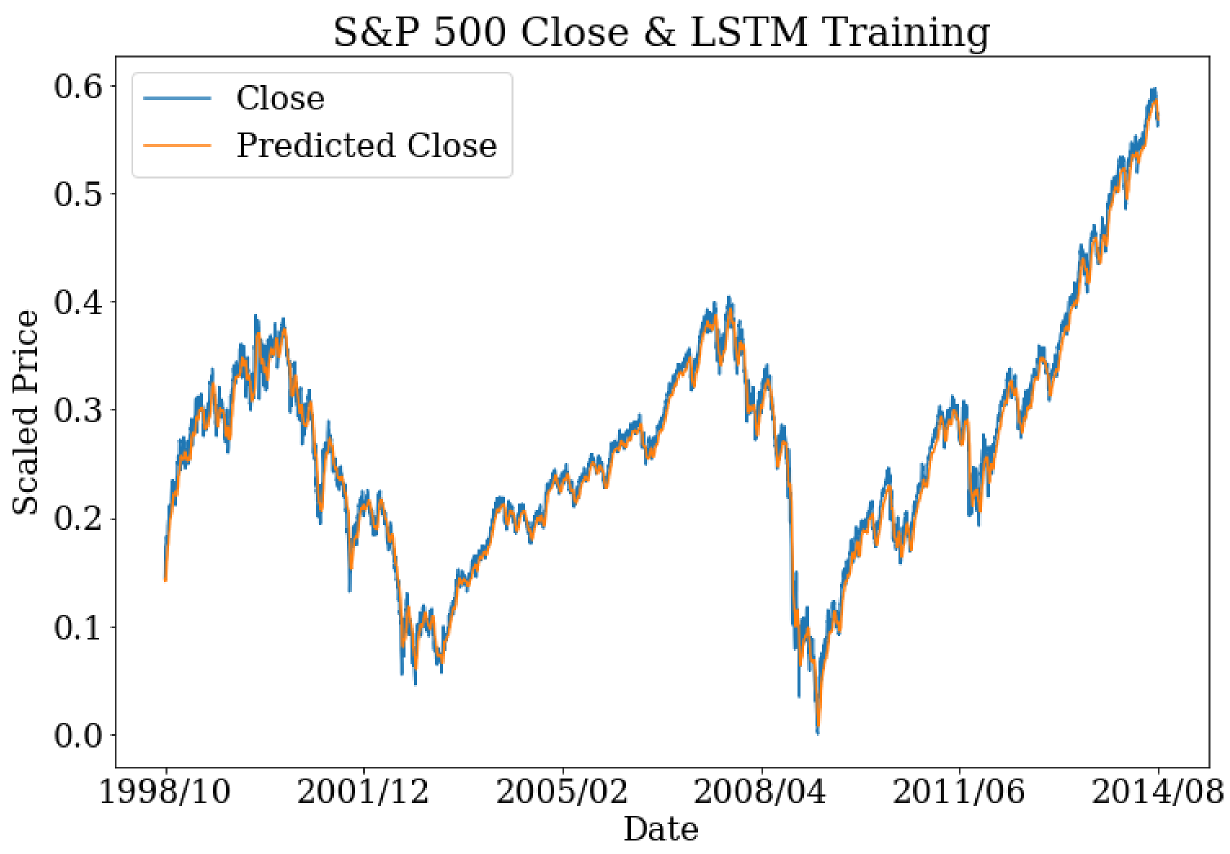
```
Train on 3783 samples, validate on 200 samples
Epoch 1/5
 - 43s - loss: 0.0042 - val_loss: 5.8551e-04
Epoch 2/5
 - 45s - loss: 6.7066e-04 - val_loss: 7.8203e-04
Epoch 3/5
 - 48s - loss: 5.3625e-04 - val_loss: 2.9757e-04
Epoch 4/5
 - 52s - loss: 5.0779e-04 - val_loss: 1.0400e-04
Epoch 5/5
 - 49s - loss: 4.7341e-04 - val_loss: 1.8065e-04
```

```
Out[52]: <keras.callbacks.History at 0x1a381ee198>
```

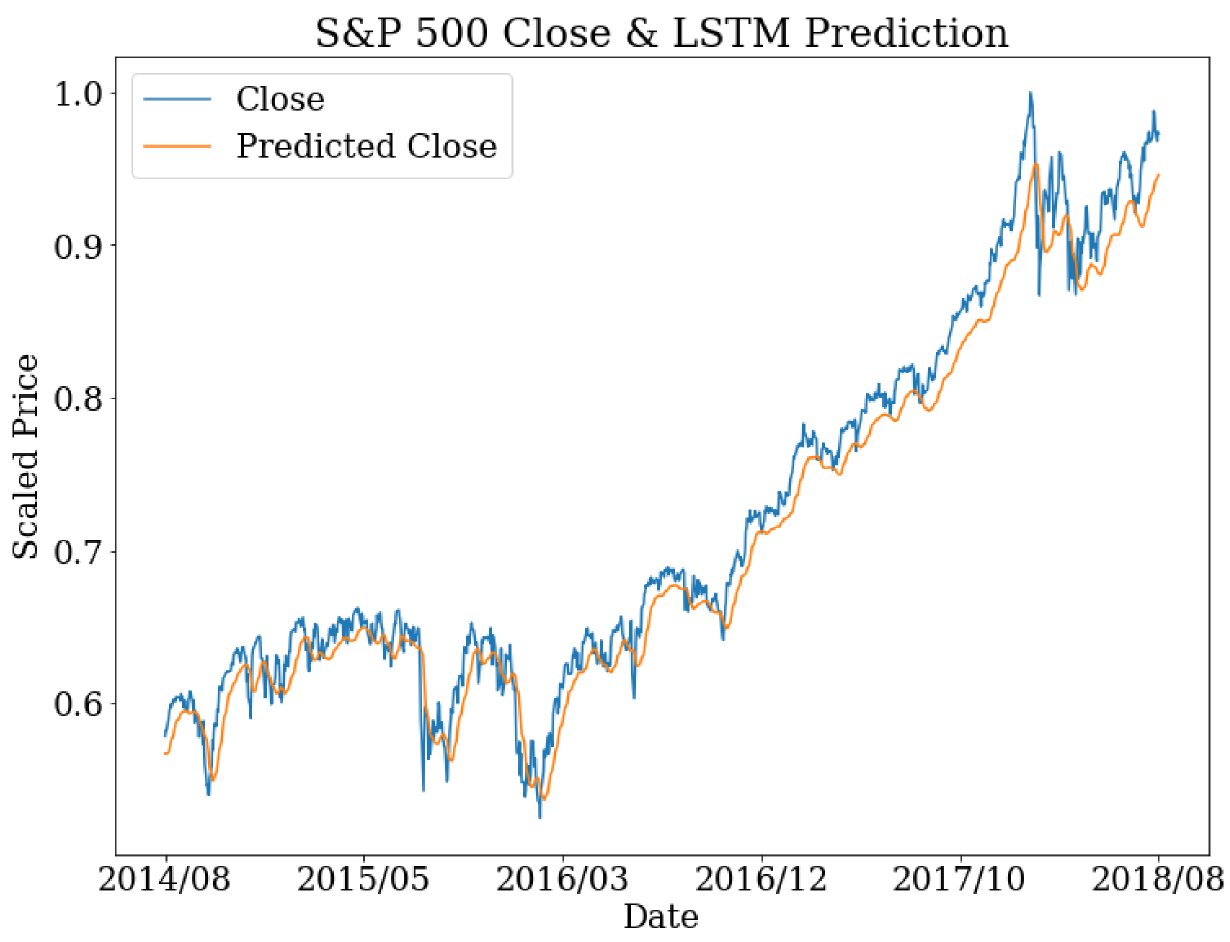
The trained model is been saved in 'saved_model/lstm.h5'.

Then we check the output performance on training dataset and make a prediction on testing dataset.

Plot the fitting result on the training dataset:



Plot the prediction result on the testing dataset:



The prediction result is much better than the Linear Regression model.

We can see the RMSE is very small on the training dataset.

However, on the testing dataset, the RMSE is not small enough, there is a bias that most of the prediction value is less than the real value. And obviously, the model is overfitted since RMSE on testing is much larger than training.

```
RMSE on training dataset:  0.013485
RMSE on testing dataset:  0.021833
```

All the values above is based on the scaled data.

For the original data, the RMSE should be magnified by a factor: max(data['Close']) - min(data['Close'])

```
Original RMSE on testing dataset:  47.953439
```

This is still not a good performance. It needs to be improved.

Refinement

We supposed to refine the parameters for a better performance, but did not get a better result. There are several reasons:

- The data in the recent 1000 days is the higher than anytime in history. That's why we got a negative bias on prediction.
- The Price everyday is changed from the Open Price today or Closing Price yesterday. That means we don't need to know what the price is, we just need to know the price change everyday. And the closing price can be derived from the price change amount. Then the prediction of the Closing Price tomorrow will not be far away from that today.

Use LSTM on Price Change

We generate another target: increasing rate everyday, and prepare the training and testing datasets as we shown in previous.

We still applied the LSTM model in previous. The only difference is we use increasing rate as the target.

Compile the model and training the model on the training dataset.

Result

Model Evaluation and Validation

Based on the training result , the validation performance is much better than the previous LSTM.

```
Train on 3783 samples, validate on 200 samples
Epoch 1/5
 - 44s - loss: 3.7539e-04 - val_loss: 1.0599e-04
Epoch 2/5
 - 45s - loss: 8.6071e-05 - val_loss: 2.9931e-05
Epoch 3/5
 - 47s - loss: 7.4710e-05 - val_loss: 4.7925e-05
Epoch 4/5
 - 51s - loss: 6.3652e-05 - val_loss: 2.9682e-05
Epoch 5/5
 - 47s - loss: 5.8748e-05 - val_loss: 3.1733e-05
```

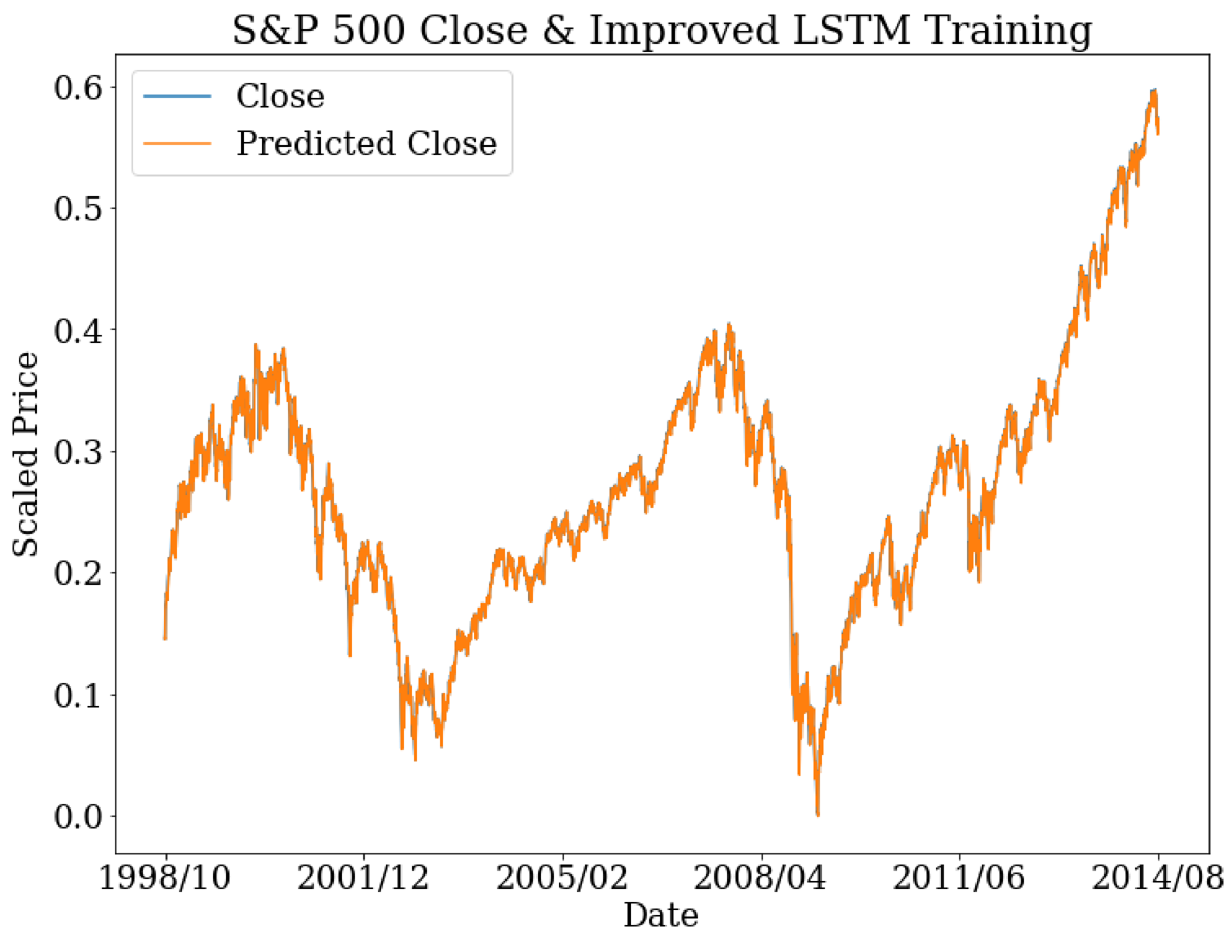
```
Out[91]: <keras.callbacks.History at 0x1a3f7a5b38>
```

Save the new LSTM model into 'saved_model/lstm2.h5'

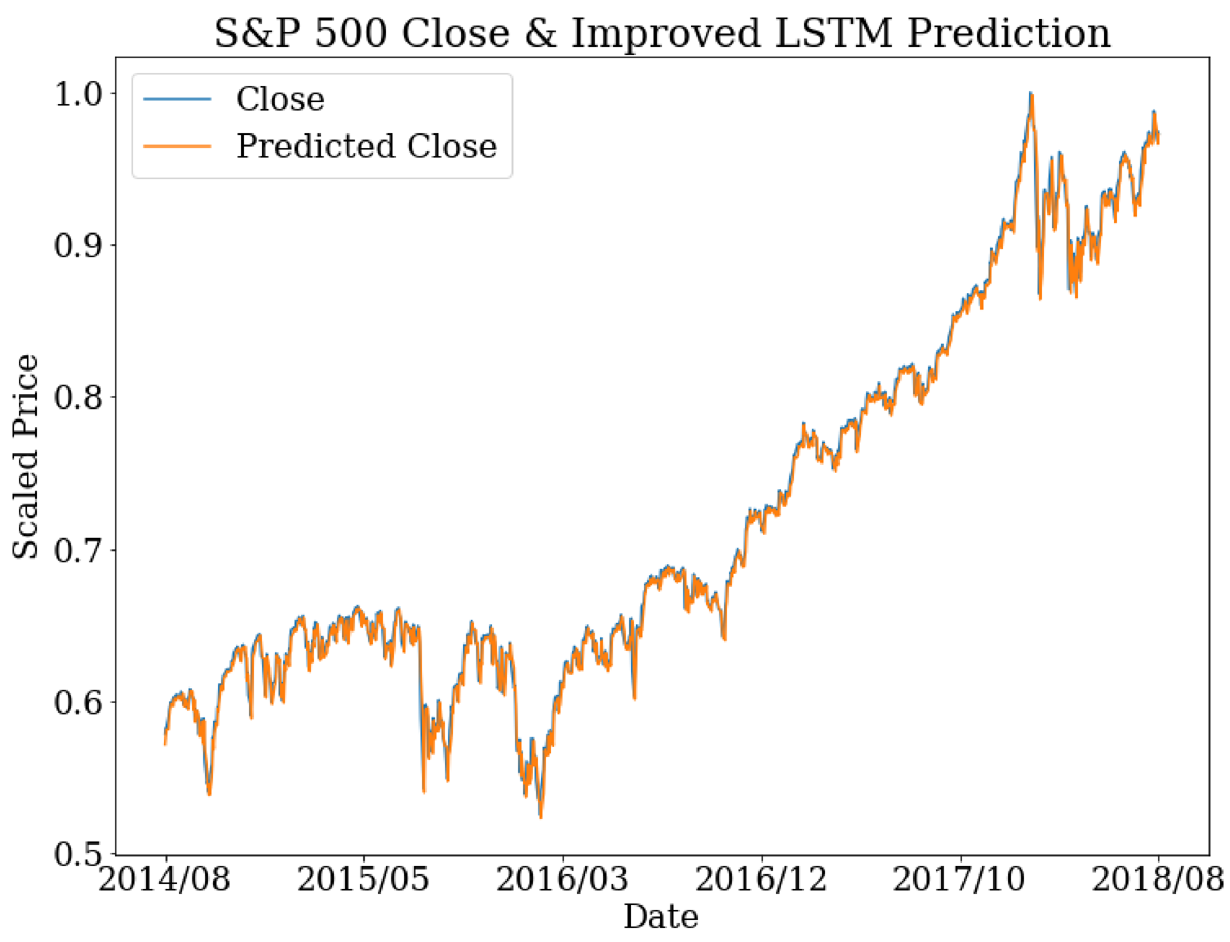
For the price change, we calculate the prediction on training dataset, and make a prediction on testing dataset.

Then calculate the scaled price value prediction, on both training set and testing set.

Plot the fitting result on the training dataset:



Plot the prediction result on the testing dataset:



```
RMSE on training dataset:  0.006712
RMSE on testing dataset:  0.008394
```

Now the prediction result has been improved much better.

We can find that RMSE becomes very small on both training and testing datasets.

For the un-scaled original data, the RMSE is also been decreased more than half.

```
Original RMSE on testing dataset:  18.435910
```

Justification

Compare with the benchmark result in Linear Regression model, our improved LSTM model improve the un-scaled RMSE from 841.723179 to 18.435910, decrease 97.8% RMSE.

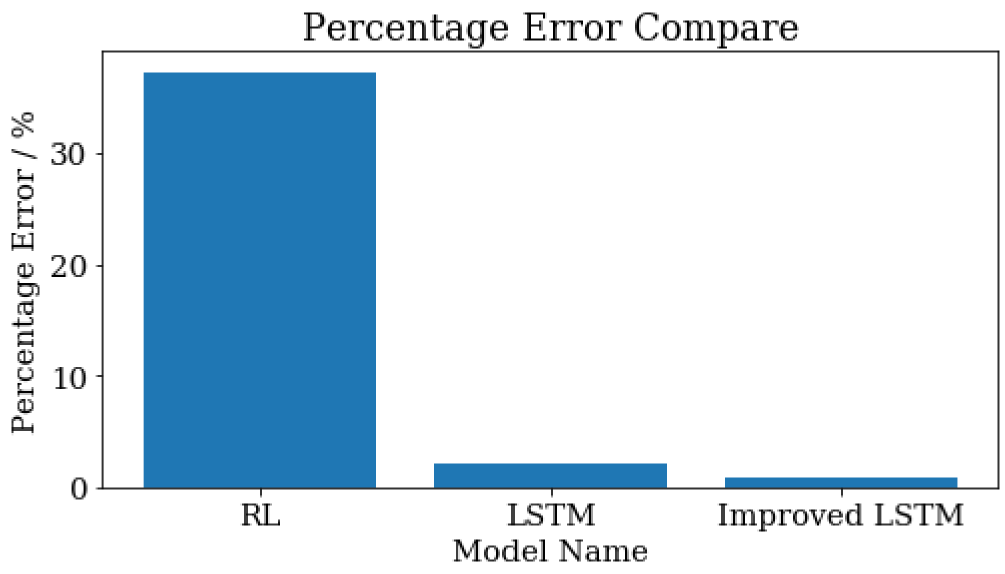
```
The average closing price in the recent 1000 days is: 2259.413251

So the average percentage error, is: 18.435910 / 2259.413251 = 0.815960 %
```

Conclusion

Free-Form Visualization

We compare the percentage error for the three models, the improved LSTM model has the best performance.



Reflection

- Our percentage error seems very small, but for a single day prediction, it is still unacceptable.
- If you just simply predict that the price tomorrow is the same with today, you can get a very good performance in RMSE. So this is just a toy model for machine learning practice, it cannot be used in real trading.

Improvement

Some improvement has not been used here, but might be useful:

- More features can be find from more datasets.
  - S&P 500 index comprises about 500 stocks, pick up some stocks and include them into input.
  - Different market interact with each other. We may add some other datasets like: Index Futures, CL Price.
- Stock price is determined by human behavoir, and human behavior is based the information they access. Internet informations, such as hot news and google search trends, indicats the potential behavior of people. An ideal model should include Natural Language Processing(NLP) on internet information.
- We may not need to create the wheel from the beginning, transfer learning from another well trained model could be another choice.

```
Out[1]:
```

Click here to toggle on/off the raw code.