---

title: "Machine Learning Engineer Nanodegree Capstone Project"

subtitle: "Machine Learning Approach to Stock Price Prediction"

author: "Moataz Eldesoky"

date: "January 29,2019"

output: pdf_document

fontsize: 12pt

I.Definition

Project Overview

Stock price prediction is a popular topic throughout last century. Using statistical methods and stochastic analysis to make stock price prediction is the mainstream in last 30 years, while using machine learning techniques to predict stock price is becoming more and more prevalence in recent years.

In their research *Support Vector Machines for Prediction of Futures Prices in Indian Stock Market*, Shom Prasad Das and Sudarsan Padhy discuss Back Propagation Technique and Support Vector Machine Technique to predict futures prices traded in Indian stock market. The reported NSME for all the futures stock index taken into consideration fall in the range of 0.9299 to 1.1521, which is a decent result.

In their study *A deep learning framework for financial time series using stacked auto encoders and long-short term memory*, Wei Boa, Jun Yue and Yulei Rao discuss combining wavelet transforms (WT), stacked auto encoders (SAEs) and long-short term memory (LSTM) to forecast stock index prices. The performance is reported across a year worth of data across various types of market and report Mean absolute percentage error (MAPE), correlation coefficient (R) and Theil's inequality coefficient (Theil U). In developed market they predict CSI 300 index, with average value of MAPE and Theil U of WSAEs-LSTM as 0.019 and 0.013.

It seems like machine learning and even deep learning methods can yield good result in stock price prediction. That is why I would like to try my own methods in this project.

In this project, I will use all the stocks in the S&P 500 as inputs and target. For each stock, the data will contain `Open`, `High`, `Low`, `Close`, `Adj Close` and `Volume`(already explained in the problem statement part) six variables. The `Adj Close` of each stock can be the target, and rest variables of this stock and all the variables of other stocks can be inputs. In order to pick a relatively stable time period, I will use data from 2001 to 2017.

We will get the symbol list of all S&P 500 stocks from Wikipedia, and then query the historical prices and volumes from Alpha Vantage through its provided API.

Problem Statement

The project aims to create a machine learning model that can predict stock price by using historical information as a time-series data. The task is to build a stock price predictor that takes daily trading data over a certain date range as input, and outputs projected estimates for given query dates. The inputs will contain multiple variables, such as opening price (Open), highest price the stock traded at (High), how many stocks were traded (Volume) and closing price adjusted for stock splits and dividends (Adjusted Close); my system will need to predict the Adjusted Close price.

Metrics

This is a regression problem, so the metrics of choice would be R-square and root-mean-squared-error. R-square can provide how much variation in the dependent variable can be explained by the variation in the independent variables. Root-mean-squared-error can provide what is the average deviation of the prediction from the true value, and it can be compared with the mean of the true value to see whether the deviation is large or small.

II. Analysis

Data Exploration

```{r echo=FALSE, message=FALSE, warning=FALSE}
library(tidyverse)
library(lubridate)
library(ggthemes)
library(knitr)
library(broom)
data = read_csv('data.csv',progress =F)
```

There are 505 companies in the S&P 500. The list of companies and symbols can be find [here](https://en.wikipedia.org/wiki/List_of_S%26P_500_companies). My data will include all these stocks' `Open`, `High`, `Low`, `Close`, `Adj Close` price and trading `Volume`. The data will start from 2001-07-31 and end at 2017-12-22. A glimpse of a typical stock's data is shown in table 1.

```{r echo=FALSE}
data %>%
  select(timestamp,open_A,high_A,low_A,close_A,adjusted_close_A,volume_PVH) %>%
  head %>% kable(caption = 'Head of Historical Prices and Volume for PVH')
```

There are 3030 columns in the whole data set, while 564 of them has missing values. This may be because some of the companies have not enter into the stock market before 2001, so they do not have the records. Because neural network can not take missing value as input, and it is impossible to impute these missing values because they do not exist by nature, I drop these columns with missing values.

In this project, I will randomly pick a stock's adjusted close price as target for illustrative purpose. The model can be applied on any of the stocks. The stock I pick is the PVH. Table 2 shows the summary statistics for adjusted close price of PVH over the whole time period. We can see that the mean and median is close to each other, indicate the target value is not skewed very much. The minimal and maximum values are far from the mean value, indicate this stock is pretty volatile.

```{r echo=FALSE}
data$adjusted_close_PVH %>% summary %>% tidy %>% kable(caption = 'Summary Statistics for PVH')
```

Data Visualization

A volatile price series may have many outliers, which may do cause problems when modelling. We can use boxplot to check that, which is shown in Figure 1. It turns out there is no outlier in the target series.

```{r echo=FALSE, fig.cap='Boxplot of Adjusted Close Price of PVH', fig.height=2}
data %>%
```

```
 ggplot(aes('adjusted_close_PVH',adjusted_close_PVH)) +
 geom_boxplot()+
 ylab('price')+
 xlab('')+
 ggtitle('Boxplot of Adjusted Close Price of PVH')+
 coord_flip() +
 theme_few()
```

Next we can visualize the target series over time. In Figure 2, we can see that there is an upward trend in the series, which is the trend for most of the stock prices in S&P 500. There is also a huge drop in 2008 because of the financial crisis. If we are using statistical analysis, then their will be a lot of statistic test and data preprocessing going on because we have to make sure the dependent and independent variable are stationary, otherwise they have to be co-integrated. But for neural network, there is no need to do these and do feature engineering(generate rolling window aggregates-like features), we can just use the raw time series.

```{r echo=FALSE, fig.cap='Adjusted Close Price of PVH'}
data %>%
 mutate(timestamp = date(timestamp)) %>%
 ggplot(aes(x=timestamp,y=adjusted_close_PVH)) +
 geom_line(col='blue')+
 ggtitle('Adjusted Close Price of PVH')+
 theme_few()
```

Finally, let us visualize the absolute correlation coefficient of target variable with all the other variables. Higher absolute correlation coefficient means the variable can provide more information about how the target variable moves. In figure 3 we can see the top 200 variables all have higher than 0.9 correlation coefficient with the target, which is very high. In feature selection part, we will choose these variables as inputs.

```{r echo=FALSE, message=FALSE, warning=FALSE, fig.cap='Histogram of Absolute Correlation Coefficient with Adjusted Close Price of PVH'}

data = data[sapply(data, function(x) !any(is.na(x)))]

r = apply(data %>% select(-timestamp), 2, function(x) cor(x,data$adjusted_close_PVH))

r %>%

 abs %>%

 as.tibble %>%

 ggplot(aes(value)) +

 geom_histogram(fill='blue') +

 theme_few()+

 xlab('Correlation Coefficient')+

 ggtitle('Histogram of Absolute Correlation Coefficient with PVH')
```

Algorithms and Techniques

In this project, I would like to use recurrent neural network to solve the problem. I will use Long-short term memory network as the model, the PVH stock as the target, and top 200 variables that are most relevant to the target as input. I will tune the sequence length which can yield the best result, and set this as the length of the LSTM network. The prediction will be one-step-ahead stock price of the target stock. Once the model is trained, the user can choose what is the period they want to take as input, and the model can predict the next date's adjusted close price.

Benchmark

The benchmark model for this project would be a linear regression. Including lagged features of the dependent variable and other exogenous features in a linear regression is called auto-regressive model with exogenous inputs, which is proved to be very successful in statistical time series modeling. This benchmark model will use the lag 1 term of 200 input variables as input, to predict the current period target variable. The model is trained on 2001-2016, the result on test period(2016-2017) is shown in figure 4. We can see that without doing required statistic test and data preprocessing like stationary check and co-integration check, linear regression on raw time series data can yield horrible results. In 2016 most of the prediction is in right scale, while in 2017 the prediction are all very negative values.

The R-squared is -26177.5 and RMSE is 2583.3. We will see in the following sections that our LSTM model will perform way better than this benchmark model.

```{r echo=FALSE, message=FALSE, warning=FALSE, fig.height=4 , fig.cap='Bencmark Model Prediction against Truth for PVH'}
read_csv('LR_prediction.csv') %>%
 gather(key='type',value='price',-timestamp) %>%
 ggplot(aes(x=timestamp,y=price,col=type)) +
 geom_line()+
 theme_few()+
 ggtitle('Bencmark Model Prediction against Truth for PVH')
```

III. Methodology

Data Preprocessing

The first thing in data preprocessing is feature selection. We have already seen in the data visualization part that the top 200 variables have higher than correlation coefficient with the target variables, we will choose these variables as input. Notice the target variable itself is also included, because it has correlation coefficient of 1.

The next step is shift feature set 1 day ahead. For example, the price on 2009-01-01 will become feature for the target on 2009-01-02. By doing this we turn the corresponding period features into lagged features, including the target variable itself.

Then we will split the data into training and testing period. The training data contain data from 2001 to 2015, and the test data contain data from 2016 to 2017.

We have prices of different stocks, they are in different scales. And also prices and volumes are in totally different scales. In order to make neural network converge faster, we should scale our inputs. We will use Min-Max scaler to scale the input in the range of 0 to 1 for all the 200 variables in the training period. Then we will use the scaler which has the min max information of the training period to scale the data in testing period. We split the scaling process because we do not want to have information leakage, which leaks testing period information into training period.

Finally it is the sequence generation. I have tried multiple sequence length in the modelling process, and find out that sequence length of 5 can yield the best result, so I will take 5 as the sequence length in the

report. I set a rolling window of length 5 days, move this window along the time series, and record the data in each step as one row of input sequence. Then the 2 dimension data frame became a 3 dimensional array, the first dimension is the number of rows(samples) in the data, the second dimension is the sequence length(5) and the third dimension is the number of features(200). We will also set this rolling window on the target sequence, and it is equivalent to cutting the first 5 days from the 1 dimensional array. This sequence generation procedure is both implemented on the training and testing data set.

Implementation

The neural network models will be implemented using Keras module. The network has one input layer, which takes the 3 dimensional array in the data preprocessing part as input. Then there are two LSTM layers. The have 100 and 50 dimensions of hidden units respectively. I use Relu activation function because it can solve vanishing gradient problem. I also use drop out layers after each LSTM layers, in order to regularize the network in order to get more generalization ability. The keep probability is set as 0.8. The final layer is the output dense layer. It has 1 neuron, with a linear activation function.

The loss function is chosen to be mean absolute error, and the optimizer is chosen to be Nesterov Adam, which is Adam RMSprop with Nesterov momentum.

The batch size used in the training process is 1000, and the whole training process will take 500 epochs. A validation set will be split out from the training set randomly with proportion of 0.1, which is 363 samples. By keeping track of the validation loss, we are able to prevent the model from over-fitting.

Refinement

There are a few hyper-parameters in the models I have tried to tune. The first is the length of the data set. At first I only use the data from 2009 to 2017, in order to avoid the 2008 financial crisis. But with this amount of data, the model converge to a point that all predictions in the testing period is a straight line. When I extend the data to 2001(in order to avoid the 2001 stock market bubble), the prediction performs well.

The second is the sequence length. I have tried sequence length of 20(it is approximately the number of trading days in a month) and 5 days, and it turns out the result of 5 days is better than the result of 20 days.

The third hyper-parameter is the number of layers. I tried network structure of 3 LSTM layers with hidden units of 150-100-50 and 2 LSTM layers described above, and it turns out the result are pretty similar. In the light of the Occam's Razor, a more parsimonious model is adopted.

The forth hyper-parameter is the loss function. I tried using mean absolute error and mean squared error as loss function, and it turns out that the model with mean squared error loss function gives prediction of a straight line, while the model with mean absolute error gives a better result.

The fifth hyper-parameter is the optimizer. I tried using RMSprop, Adam and Nesterov Adam, and the result are pretty similar. Since in the literature it is reported that Nestrov Adam can yield better convergence, I go with the Nesterov Adam.

The final hyper-parameter is the number of epochs. According to the learning curve I will show in the next section, the loss function converges for both training and testing set converge after 500 epochs, so I tried using epochs of 500 and 1000, and it turns out 1000 gives better result.

IV. Results

Model Evaluation and Validation

Using the model described above to train on training data set, we can use the trained model to predict on testing data set. The result is shown in the figure 5. We can see that the prediction is quite good. It performs well in 2016, especially in second half of 2016, it almost perfectly matched. In 2017, the prediction is smaller than the actual value, which is the same direction as the benchmark model, while the error is much more smaller than the benchmark model. The R-square is 0.54 and RMSE is 10.81. Although R-square of 0.54 is not a very high number in common sense, but consider there are so many confounding factors in stock price determination, this result is quite good. The RMSE is way smaller than the mean value of the testing data set, which also indicate this model has a good performance.

```{r echo=FALSE, message=FALSE, warning=FALSE, fig.height=4, fig.cap='LSTM Model Prediction against Truth for PVH'}
read_csv('LSTM_prediction.csv') %>%
 gather(key='type',value='price',-timestamp) %>%
 ggplot(aes(x=timestamp,y=price,col=type)) +
 geom_line()+
 theme_few()+
 ggtitle('LSTM Model Prediction against Truth for PVH')
```

Now that we have a valid model on one stock, we can try to do robustness check and see whether this model works on other stocks. Here I randomly select another stock, ADS, and apply the same data preprocessing and LSTM model on the adjusted close price of ADS. The result is shown in figure 6. We can see that our prediction has the similar 'shape' with the actual value, while the predictions are like

shifted downward a little bit. Thus the result is not as good as PVH, it has R-square of -0.33 and RMSE of 21.16. The RMSE is still way smaller than the mean value in the testing data, and considering the model is correctly predicting the upward and downward trend in the actual value, we can say that the model is robust over different stocks.

```{r echo=FALSE, message=FALSE, warning=FALSE, fig.height=4, fig.cap='LSTM Model Prediction against Truth for ADS'}
read_csv('ADS_prediction.csv') %>%
 gather(key='type',value='price',-timestamp) %>%
 ggplot(aes(x=timestamp,y=price,col=type)) +
 geom_line()+
 theme_few()+
 ggtitle('LSTM Model Prediction against Truth for ADS')
```

Justification

Now we can compare the model performance of benchmark model and LSTM model. In table 3 we can clearly see that the LSTM model perform much better than the benchmark model. As I discussed in the benchmark model section, linear regression on time series data should be taken good care of in statistic tests and data preprocessing, otherwise it might fail. However, LSTM model will generate its own feature representation in the hidden units, and it's long-short term memory cell is good at keep useful information and throw away useless information through its gate units, that is why LSTM is simple but useful in dealing with time series data. Although the metrics shows the LSTM model is not perfect, but it is still a very good model for stock price prediction.

```{r echo=FALSE, message=FALSE, warning=FALSE}
tribble(~Stock,~Model,~`R-Squared`,~RMSE,
      'PVH','Benchmark',-26177.50, 2583.33,
      'PVH','LSTM',0.54,10.81,
      'ADS','Benchmark',-40646.19, 3701.37,
      'ADS','LSTM',-0.33, 21.16) %>%
 kable(caption = 'Performance Metrics of Benchmark and LSTM Models')
```

```

# V. Conclusion

## Free-Form Visualization

A very important topic for neural networks is its convergence of loss function. Not only the convergence will show whether the model is successfully trained, it will also tells you whether the model is under-fitting or over-fitting. Figure 7 shows the learning curve of the training process of the model.

We can see from the curve that both the loss for training and validation set is decreasing as the number of epoch increases, and converges to almost zero. This indicates the model is successfully training, at least is converging into a local minimum. The validation loss is decreasing and is less volatile as the number of epoch increase, which indicates there is no over-fitting or under-fitting problem in the training process.

```{r echo=FALSE, message=FALSE, warning=FALSE, fig.height=4, fig.cap='Learning Curve for LSTM Model on PVH'}

read_csv('history.csv') %>%

 gather(key='type',value='Mean Absolute Error',-X1) %>%

 ggplot(aes(x=X1,y=`Mean Absolute Error`,col=type)) +

 geom_line()+

 theme_few()+

 xlab('Epochs')+

 ggtitle('Learning Curve for LSTM Model on PVH')
```

Reflection

This project discuss a machine learning approach to predict the stock price, more specifically, the adjusted close price, using lagged terms of itself and other stocks' prices and trading volumes.
```

First we will query historical stock price data from Alpha Vantage. Actually getting data is the most difficult problem in this project. Previously people will use pandas built in method to query historical stock price, and it works well with Yahoo-finance and Google Stock. However, after May 2017, Yahoo-finance closed its developer API, and Google Stock stopped permission for automatic query. Although pandas still works fine with Yahoo-finance, but it will shut down request once you made around 10 consecutive queries. A very well-known finance data provider Quandl provide API to their database, however the data quality is bad, there are a lot of missing values in the database. It took me one day to find a database that has high quality data and free to use API.

After querying the data, we did some data exploration and visualization to prove the data is valid to do modelling, and have a visual check on the feature selection. I also delete columns that has missing values.

The next step is programmatic feature selection. I use Pearson correlation to calculate the correlation of target price with all the other 2999 features, and select 200 features that has the highest correlation with the target price.

After feature selection, the remaining 100 time series are turned into sequences. The sequence generation is follows: set a rolling window of length 5 days, move this window along the time series, and record the data in each step as the feature sequence for the next day.

After we have generated our feature sets, we can input these features into the benchmark linear regression and LSTM networks. I will use scikit-learn's `LinearRegression` class to do the linear regression, and use Keras's `LSTM` function to build the neural networks. The networks have two layers, with hidden states dimension of 100-50. Each layer will use Relu as activation function, and each layer has a dropout layer with keep probability of 80% in order to regularize. The output layer has dimension of 1, and linear activation function.

I split the data into 2001 to 2016 as training, and 2017 as validation data. I use R-square and RMSE to check the model performance, on both benchmark model and LSTM model. The final result turns out that the LSTM model is much better than the benchmark model, and can successfully and robustly predict the adjusted close price of multiple stocks. I believe this model can be implemented on all the stocks and can yield good prediction on most of them.

One interesting thing about the hyper-parameter turning process is it seems to converge at 500 epochs, both the training and validation loss hardly change after 500 epochs(in Figure 7). However, training the model with 1000 epochs yield much better results than 500 epochs on testing data set. I may think this is because the validation loss is less volatile after 1000 epochs. So not only the loss is a good indicator of the training process, but also the volatility of the loss.

Improvement

One thing that can be improved in the future for the same type of problem is feature selection. In this project I only use Pearson correlation to select relevant features and choose the top 200, which leave tons of information out. One thing can try is use dimensional reduction techniques like PCA, and choose

top 200 principle components as the features. The reason why I did not use this in this project is PCA generate new dimensions based on the direction of maximal variance, but this variance may not be relevant to the target variable. Doing PCA might totally mess up the information in the feature set, that is why I choose a more conservative method in the project, but it is definitely worthy to try different dimensional reduction techniques and check their performance.

Reference

Das, Shom Prasad, and Sudarsan Padhy. "Support vector machines for prediction of futures prices in Indian stock market." International Journal of Computer Applications 41.3 (2012).

Bao, Wei, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." PloS one 12.7 (2017): e0180944.