Machine Learning Engineer Nanodegree

Capstone Project

Tagne Nguifo Eric Hermann

14th August 2017

Definition

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behaviour and make profitable investments and trades. Stock market is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The successful prediction of a stock's future price could yield significant profit [1]. The top level aim of this project is to utilize machine learning to predict stock prices. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process. For this project, stock price indicator, the dataset to be used will be that of publicly traded companies from the S&P 500 stock market, Microsoft Corporation (MSFT), Amazon.com, Inc. (AMZN) and The Walt Disney Company (DIS) obtained for free from Yahoo! Finance.

Problem Statement

There are a lot of complicated financial indicators and also the fluctuation of the stock market is highly violent. However, as the technology is getting advanced, the opportunity to gain a steady fortune from the stock market is increased and it also helps experts to find out the most informative indicators to make a better prediction. The prediction of the market value is of great importance to help in maximizing the profit of stock option purchase while keeping the risk low.

Deep Recurrent neural networks (D-RNN) have proved one of the most powerful models for processing sequential data.

Long Short – Term memory is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. With these memory cells, networks are able to effectively associate memories and input remote in time, hence suit to grasp the structure of data dynamically over time with high prediction capacity.

In this project, I have presented modelled and predicted the stock returns of the S&P 500 using LSTM. Statistical analysis will be carried out on S&P 500 stock market, Microsoft Corporation (MSFT), Amazon.com, Inc. (AMZN) and The Walt Disney Company (DIS). I collected about 17 years of historical data of the S&P 500 and used it for the training and validation purposes for the model[2].

Metrics

The metrics that will be used to determine the model are the:

- R2 score for coefficient of determination for the test datasets

- The mean Squared Error for the test datasets

The R2 score will let us know the quality of the fit of the model against the underlying data. R2 scores range from 0 (the model seems to disregard the input data) to 1 (the model is a perfect representation of the data) and can go to negative (as in, the model is doing worse than ignoring the data). R2 is calculated as 1- (Sum of Squares of residuals/Total Sum of Squares).

The mean squared error function computes mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error loss or loss.

Analysis

Data Exploration

I downloaded historical data from 2000/01/01 to 2017/07/28. Adjusted Close is what we will be trying to predict and prediction will only be carried out on S&P 500.the others will be about statistical analysis.

All stock data files have the same columns:

- Date (date of the stock price)

- Open (opening price of the stock

- High (highest price of the stock for the day)

- Low (lowest price of the stock for the day)

- Close (price of the stock at close)

- Volume (the trading volume of the stock for the day)

- Adj Close (the adjusted close price)

note: the adjusted close price will be different from the "Close" price when a company chooses to split the stock, give dividends, etc…

sample for S&P 500:

	Date	Open	High	Low	Close	Volume	\
0	1/3/2000	148.250000	148.250000	143.875000	145.4375	8164300	
1	1/4/2000	143.531204	144.062500	139.640594	139.7500	8089800	
2	1/5/2000	139.937500	141.531204	137.250000	140.0000	12177900	
3	1/6/2000	139.625000	141.500000	137.750000	137.7500	6227200	

```
4 1/7/2000  140.312500  145.750000  140.062500  145.7500   8066500
```

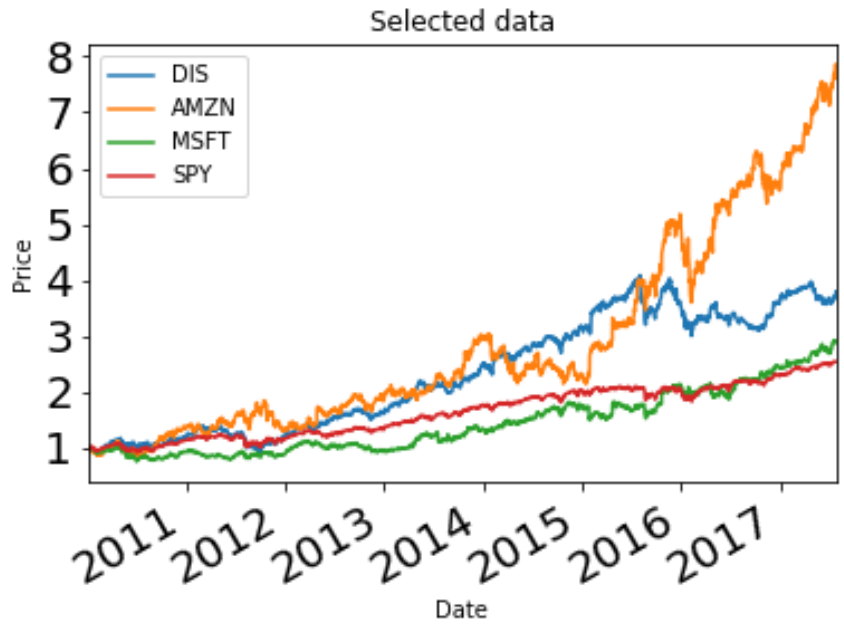
```
    Adj Close
0  104.855598
1  100.755112
2  100.935326
3   99.313148
4  105.080872
```

Statistical description of the Adjusted prices of the different stock markets are as follows

```
                DIS          AMZN          MSFT          SPY
count   4421.000000   4421.000000   4421.000000   4421.000000
mean      42.016462    190.310063     27.338878    121.304903
std       29.679853    228.561495     12.550046     45.723925
min       11.074709      5.970000     12.236407     57.161503
25%       21.503023     38.439999     19.353493     89.097977
50%       29.171387     79.190002     22.527864    105.394592
75%       53.108875    264.269989     29.354351    141.670441
max      118.245178   1052.800049     74.220001    247.429993
```

Exploratory Visualization

- Visuals in this file display data for the various stock markets
- To be able to easily visualize stocks, the stock data had to be normalized to view the differences in performance through time.

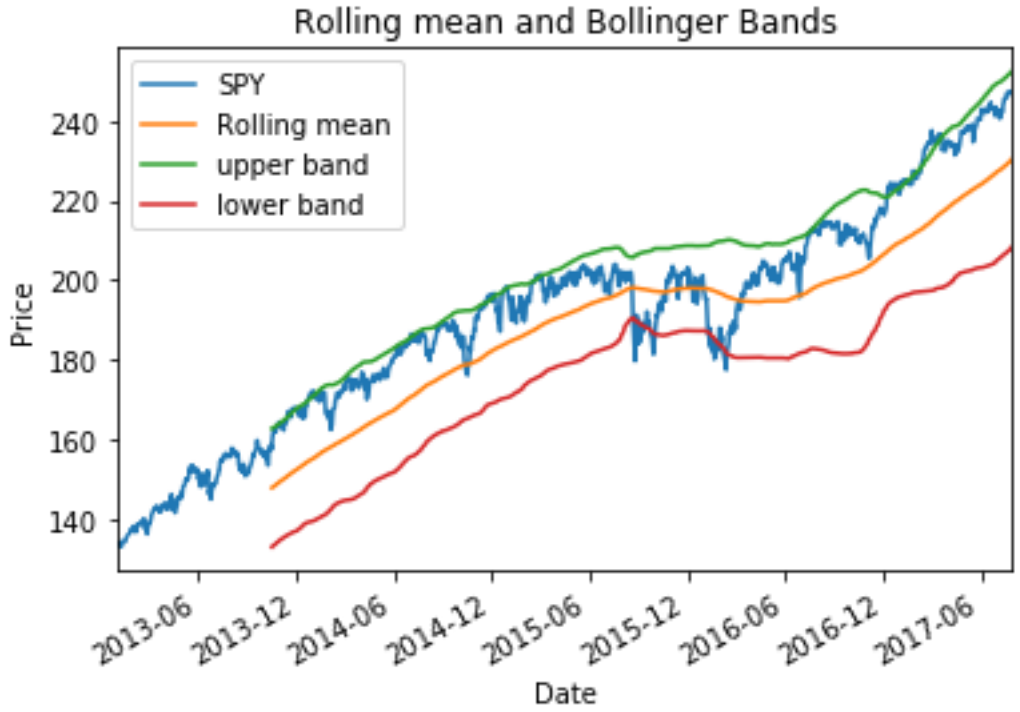


Bollinger bands and Rolling mean

- Visuals here displays the famous Bollinger bands and Rolling mean for S&P 500 .

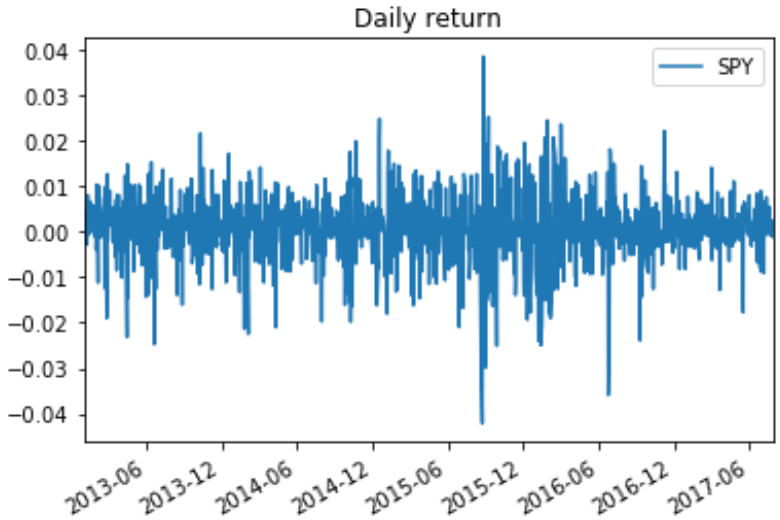
- The space between the upper and lower bands indicate the amount of risk (aka standard deviation) in a stocks movement

-In statistics, a moving average (rolling average or running average) is a calculation to analyze data points by creating series of averages of different subsets of the full data set. It is also called a moving mean (MM) or rolling mean and is a type of finite impulse response filter[3].



Daily returns

Visuals here show the daily returns of the S&P 500



Algorithms and Techniques

For this project, I will be using a recent Deep Recurrent Neural Network algorithm, Long Short-Term Memory (LSTM) to predict Adjusted Close prices for S&P 500.

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation through Time and overcomes the vanishing gradient problem.

As such, it can be used to create large recurrent networks that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results.

Instead of neurons, LSTM networks have memory blocks that are connected through layers.

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A block operates upon an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional.

There are three types of gates within a unit:

- **Forget Gate**: conditionally decides what information to throw away from the block.
- **Input Gate**: conditionally decides which values from the input to update the memory state.
- **Output Gate**: conditionally decides what to output based on input and the memory of the block.

Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.[4]

An LSTM is well-suited to learn from experience to predict time series given time lags of unknown size and bound between important events.

Benchmark

The benchmark model to be used in this project is linear regression.

Sklearn's linear regression fits the best line to the given data. The equation for this algorithm is y = mx + b. The algorithm processes a dataset to produce the m and b variables. Once complete, the algorithm now relies on the y = mx + b equation to produce a prediction. In this project, the target prediction is the price of a stock 7 days later. For the prediction, the x features are fed into the querying function and the result is a predictive y variable.

Methodology

Data Preprocessing

- By default, the csv files are indexed by a number starting from "0". The code needed to explicitly identify the "Date" column as the index column.
- Since we are interested in predicting Adjusted Close price using LSTM, we will drop out all the other columns

- For the benchmark model, all the data will be used and another column will be added to view Adj Close 7 days later before training is done on the data
- We normalised the data before training it with the LSTM model by taking each n-sized window of training/testing data and normalize each one to reflect percentage changes from the start of that window (so the data at point i=0 will always be 0). We used the following equation to normalise

Normalisation:

n = normalised list [window] of price changes
p = raw list [window] of adjusted daily return prices

$n_i = (p_i/p_0) - 1$

Implementation

Deep Recurrent Neural Network (LSTM)

Now that we have the data, we want the LSTM to learn the Adjusted Close price from a set window size of data that we will feed it and then hopefully we can ask the LSTM to predict the next N-steps in the series and it will keep spitting out the Adjusted close price.

We'll start by transforming and loading the data from the CSV file to the numpy array that will feed the LSTM. The way Keras LSTM layers work is by taking in a numpy array of 3 dimensions (N, W, F) where N is the number of training sequences, W is the sequence length and F is the number of features of each sequence. I chose to go with a sequence length (read window size) of 50 which allows for the network so get glimpses of the shape of the adjusted close price pattern at each sequence and hence will hopefully teach itself to build up a pattern of the sequences based on the prior window received. The sequences themselves are sliding windows and hence shift by 1 each time, causing a constant overlap with the prior windows.Also, we will take each n-sized window of training/testing data and normalize each one to reflect percentage changes from the start of that window (so the data at point i=0 will always be 0).

We then divide the data up into 50 day sequences,so the network will learn to predict patterns in volume based on the prior 50 days data. 90% of the sequences will go in our training dataset and 10% will go in our test dataset. Each sequence is a sliding window, shifting forward one day with each new sequence. This means that there is a constant overlap with prior windows and we will have plenty of training data for the demonstration.

Initially, I used the test set as it is, I ran each window full of the true data to predict the next time step. This was fine if we are only looking to predict one time step ahead, however if we're looking to predict more than one time step ahead, maybe looking to predict any emergent trends or functions (e.g. the Adjusted close price in this case) using the full test set would mean we would be predicting the next time step but then disregarding that prediction when it comes to subsequent time steps and using only the true data for each time step. I chose 50 by trial and error,also I wanted my model to make prediction for more than a month

Here's the code to load and normalise the training data CSV into the appropriately shaped numpy array:

```
def load_data(stock, seq_len, normalise_window):
```

```
    amount_of_features = len(stock.columns)
    data = stock.as_matrix()
    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
    if normalise_window:
        result = normalise_windows(result)

    result = np.array(result)
    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))

    return [x_train, y_train, x_test, y_test]
def normalise_windows(window_data):
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data
```

Next up we need to actually build the network itself. This is the simple part! At least if you're using Keras it's as simple as stacking Lego bricks. I used a network structure of [1, 50, 100, 1] where we have 1 input layer (consisting of a sequence of size 50) which feeds into an LSTM layer with 50 neurons, that in turn feeds into another LSTM layer with 100 neurons which then feeds into a fully connected normal layer of 1 neuron with a linear activation function which will be used to give the prediction of the next time step.

Here's the code for the model build functions:

```
def build_model(layers):
    model = Sequential()
    model.add(LSTM( input_dim=layers[0],output_dim=layers[1],return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM,  layers[2],return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense( output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print(">Compilation Time : ", time.time() - start)
    return model
```

Finally it's time to train the network on the data and see what we get. I used 500 training epoch with this LSTM.

Here is the code to run the model:

```
if __name__=='__main__':
    global_start_time = time.time()
     # Read data
    dates = pd.date_range('2000-01-03', '2017-07-28')
    symbols = ['SPY']
    df = get_data(symbols, dates)
    epochs  = 500
    seq_len = 50
     # Load Data
    print('>Loading data... ')
    X_train, y_train, X_test, y_test = load_data(df, seq_len,True)
    print('>Data Loaded. Compiling...')
    model = build_model([1, 50, 100,  1])
#Train the model
```

```
  model.fit(
      X_train,
      y_train,
      batch_size=512,
      nb_epoch=epochs,
      validation_split=0.1,
      verbose=1)
  # Predicting the model
  predictions = predict_sequences_multiple(model, X_test, seq_len, 50)
```

Linear Regression (the benchmark model)

The first algorithm I implemented was sklearn's Linear Regression to predict stock prices 7-days later.

The first processing step in the implementation was to read the data from a csv file in a separate folder. I used the python pandas library to read the csv files with a function called read_csv. Below is a snippet of code related to this topic:

```
import pandas as pd
...
df = pd.read_csv(symbol_to_path('SPY'), index_col='Date', parse_dates=True,
usecols=['Date','Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], na_values=['nan'])
```

The df variable captures all the data from the csv file. The data then needs to be sorted in an ascending order to display the historical data through time. Below is the code to sort the data in an ascending order:

```
df = df.sort_index(ascending=True, axis=0)
```

Once the data is properly sorted, I appended a second "adjusted price" column to compare the price of the stock 5-days later. After copying over the column, I shifted the data up 7 rows. With the addition of this new column, the dataset now has two adjusted price datapoints in the same row: current price and the stock price 5-days later. With this information, the dataset can be processed by a machine learning algorithm for price predictions, and calculate the variation of the prediction against the actual stock price. The code for appending the new column is below:

```
df['Adj_Close_7_Days_Later'] = df['Adj Close']

df['Adj_Close_7_Days_Later'] = df['Adj_Close_7_Days_Later'].shift(-7)
```

The last step before running the data through the machine learning algorithm is to split the data for training and testing. The training subset will be processed to create a function to best draw a straight line through the given data. The testing subset will be to predict the target value using unseen data. Snippet of code:

```
# split dataset between train and test subsets

X_train = df.iloc[0:2000,:-1]

  y_train = df.iloc[0:2000, -1]

  X_test = df.iloc[2000:3000,:-1]

  y_test = df.iloc[2000:3000, -1]
```

```
# Create linear regression object

 regr = linear_model.LinearRegression()
```

```
# Train the model using the training sets

regr.fit(X_train, y_train)
```

The evaluation will be done on the test sets and it will be the same procedure for both the LSTM and the linear regression model.

```
 # The mean square error

  print "Mean squared error: ", mean_squared_error(y_test, regr.predict(X_test)), "\n

   # The R_2 score

  print "R_2 score is: ", r2_score(y_test, regr.predict(X_test)), "\n"
```

Refinement

Linear Regression: Little work was done on the linear regression: it is pretty much what I started with.

Recurrent Neural Network (LSTM): There are quite a few other components that could be tuned and adjusted however such as added layers, added components like batch normalization, etc but I could not implement these due to lack of computational power.

Results

	LSTM	Linear Regression
Mean Squared Error	6.98992275718e-05	14.6820359285
R2 Squared	0.965766681239	0.935150907258

Overall, the recurrent neural network (LSTM) performed far more than the Linear Regression.The ratio of R2 squared of LSTM to that of linear regression is about 1.03 : 1 and the ratio of the mean squared error of the LSTM to that of the linear regression is 210045.753 :1.Due to the high R2 score and very low Mean squared error I think the LSTM model is significant enough to have adequately solved the problem.

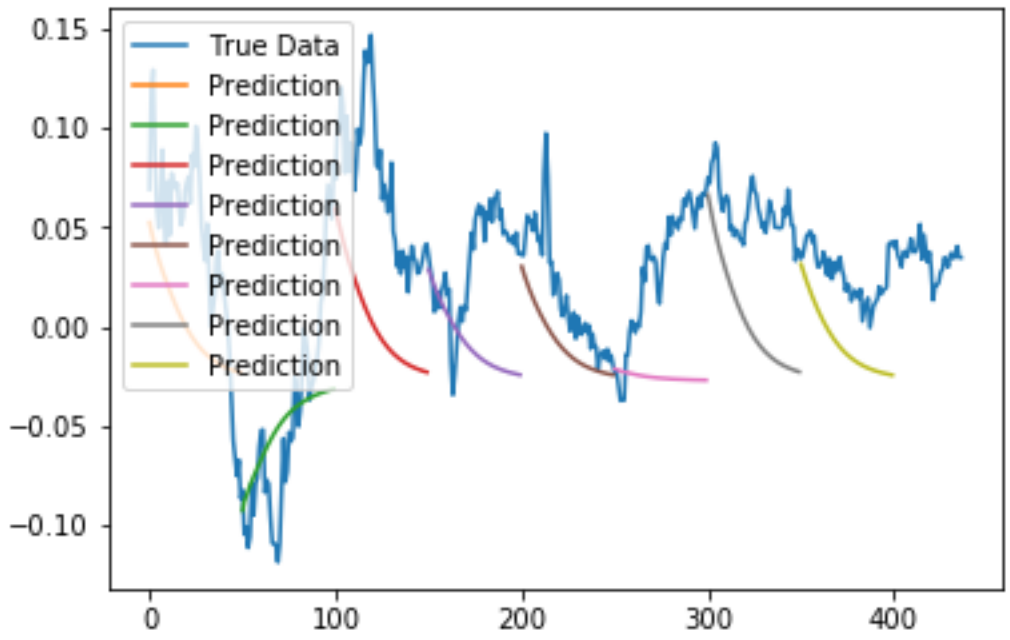
Conclusion

Free-Form Visualization

I'm going to be honest here and say that the result in the graph below has surprised me slightly because we are trying to predict future price movements from purely historical price movements on a stock index (due to the fact that there are so many underlying factors that influence daily price fluctuations; from fundamental factors of the underlying companies, macro events, investor sentiment and market noise…) however checking the predictions of the very limited test we can see that for a lot of movements, especially the large ones, there seems to be quite the consensus of the model predictions and the subsequent price movement.

In fact when we take a look at the graph below with the epochs equals to 500 we see that actually it now just tries is predict an downward momentum for almost every time period.

There are many, MANY reasons why the above "promising looking" graph could be wrong. Sampling errors, pure luck in a small sample size… nothing in this graph should be taken at face value and blindly followed into a money sucking pit without some thorough and extensive series of backtests.



Reflection

In this project I tried to use deep Recurrent Neural Network to predict Adjusted close prices for S&P 500.I first downloaded historical data from the S&P 500 stock market, Microsoft Corporation (MSFT), Amazon.com, Inc. (AMZN) and The Walt Disney Company (DIS) obtained for free from Yahoo! Finance. Since I was interested in Adjusted Close price, the other characteristics of the data was dropped out. A graphical visualisation of the different stocks was provided. Also a description of the various characteristics of the data was provided such as mean, median, count, standard deviation etc for its various adjusted close prices. Statistical analysis was performed on the S&P 500 such as daily return, Rolling mean and Bollinger bands. The data (adjusted close price for S&P 500) was then loaded and normalised and then passed in the LSTM (Long Short-Term Memory) model I built.Trainng was then carried out. When training was done, prediction was carried out on the test set and it was subsequently evaluated using R2 score and mean squared error. A benchmark model was also provided which is linear regression. Here what we tried to predict was the adjusted price 7 days later. We did this by introducing a new column which is adjusted price 7 days later which is our label. Our features will be Open, High, Low, Close, Volume and Adjusted Close. The data is then trained and prediction is done on the test data. Evaluation is done using the same metric as LSTM.Comparism is then done on both model.

The most interesting aspect I found in this project was to experience how powerful LSTM could be. With intelligent use, LSTMs have proven again and again that they can effectively predict outcomes from even the most complicated time series or even sequence based data. The challenge I faced was to be able to understand the concept of LSTM which was a little bit complex and implement the concept into code.

Improvement

While LSTMs performed well out of the box in this context, be careful making any decisions based on these kinds of simplistic demonstrations. Tuning and cautious testing could reveal many areas for improvement. This simple LSTM does a decent job here, but prediction success could vary widely for different time intervals and different companies. No doubt the predictions could also be improved

by adding in other relevant variables and financial expertise such as fundamental factors of the underlying companies, macro events, investor sentiment and market noise etc.

References

1 https://en.wikipedia.org/wiki/Stock_market_prediction

2 https://www.ijsr.net/archive/v6i4/ART20172755.pdf*

3 https://en.wikipedia.org/wiki/Moving_average

4 http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

5 http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Ne