Machine Learning Engineer Nanodegree

Capstone Project: Predicting stock prices

Jörn von Weihe

Tuesday, June 5, 2018

I. Definition

Project Overview

Stock markets are known to be as hardly predictable. To decide if to buy, sell or hold shares of a stock, investors face the challenge to determine the future value of a stock. According to the efficient market hypothesis the current stock price reflects all relevant information, and it is not possible to predict the future price of a stock.[1]

Also traditional approaches to predict stock prices as the fundamental or technical analysis cannot prove to make better predictions of the stock market than random guessing. Empirical studies show that in history professional managed investment funds could hardly outperformance the market if management fees were considered.[2]

Now investment firms invest heavily in machine learning and AI to make better investment decisions. BlackRock for example uses now machine learning to replace investments managers to decide which stock to pick for an investment portfolio.[3]

Investors need to predict future stock prices, to make the decision if to hold, buy or sell a stock. The stock price predictor should support making this decisions. Based on historic prices for a stock or an index (e.g. Google, S&P500), the stock price predictor should predict the future daily stock prices for certain timeframe (e.g. next seven days). Even if exact predictions are unlikely, the predicted stock prices should be at least be slightly better than random guessing, and so give an indication for investment decisions.

Problem Statement

Based on historic data, predictions for the future adjusted closing price of the Standard & Poor's 500 (S&P 500) stock market index will be made. The stock price predictor should solve a classical time series prediction problem. To learn the sequential patterns in the data and under consideration of long-term dependencies of the data points, a Recurrent Neural Network (RNN) with a Long Short-Term Memory should be used. The solution should be

The solution of the problem will follow this steps:

1. Gathering and exploration of the stock price data
2. Preparation of data for training and testing
3. Creation of a benchmark model (linear regression) to evaluate the solution
4. Design of the architecture of the LSTM-network in Keras with a Tensorflow backend

5. Training and fitting of the network with training data
6. Prediction of stock prices and evaluating the model with test data
7. Improvement of the model and tuning of parameters
8. Visualization and discussion of results

The final solution of the stock price predictor should make predictions for the adjusted stock price of the S&P 500 for multiple days in the future (e.g. 50 days) using a sequence of past stock prices.

Metrics

The performance of the model will be evaluated by the mean squared error (MSE) which is a classical metric for the performance of regression problems:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_{i-1}\right)^2$$

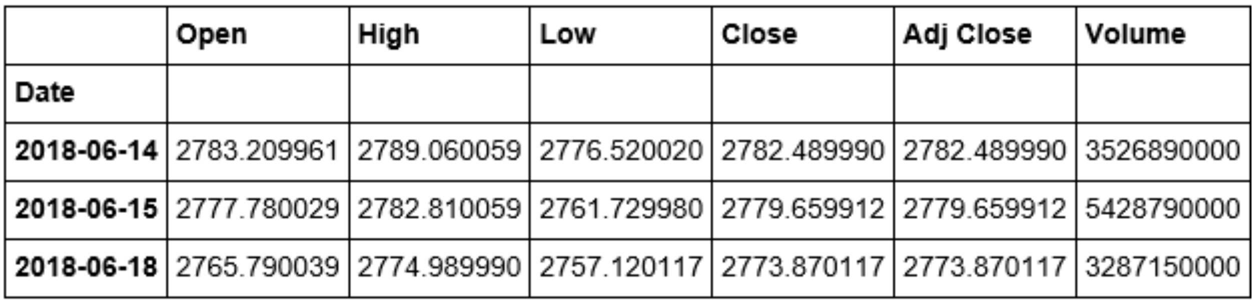
The MSE measures the average squared deviations between vector of observed values (Y) and the vector of predicted values ($\hat{Y}$) for n data points. In this case this are the true stock prices (target values) of out testing data and the corresponding predicted stock prices. Squaring of the difference penalizes larger errors more than smaller errors and eliminates the effect of sign. As smaller the MSE as closer the prediction is to the true values.

II. Analysis

Data Exploration

For this project I used the data of the Standard & Poor's 500 (S&P 500) stock market index. This index is based on the market capitalizations of 500 large American companies and often considered as one of the best representations of the U.S. stock market. For investors a prediction of the index is not only interesting to get an indication for the trend of whole American stock market, with investment funds as exchange-traded funds investors can also directly participate in the wins and losses of this index.

For this project I downloaded the data of the S&P500 from Yahoo finance. The dataset consist of 17226 entries from 1950-01-03 to 2018-06-18 on a daily (trading days) basis. An example of the data you find below.



	Open	**High**	**Low**	**Close**	**Adj Close**	**Volume**
Date						
2018-06-14	2783.209961	2789.060059	2776.520020	2782.489990	2782.489990	3526890000
2018-06-15	2777.780029	2782.810059	2761.729980	2779.659912	2779.659912	5428790000
2018-06-18	2765.790039	2774.989990	2757.120117	2773.870117	2773.870117	3287150000

For each day the open, the closing, the lowest, the highest and the adjusted closing price is provided. Furthermore, the trading volume of the day is included in the data. In this project I will focus on the adjusted closing price. The price is adjustment based on corporate actions before which take place before next day stock opening e. g. stock splits, dividends or right offereings. The adjusted closing price is usually used for the analysis of histrocial stock prices because it reflects best the real value of a stock.

Below you find basis statistics of the data.

	Open	High	Low	Close	Adj Close	Volume
count	17226.000000	17226.000000	17226.000000	17226.000000	17226.000000	1.722600e+04
mean	551.383918	554.660570	547.924604	551.512920	551.512920	8.998692e+08
std	650.004758	653.337121	646.403181	650.119925	650.119925	1.534129e+09
min	16.660000	16.660000	16.660000	16.660000	16.660000	6.800000e+05
25%	85.524998	86.250000	84.870003	85.552500	85.552500	8.752500e+06
50%	164.819999	165.449997	164.010002	164.790001	164.790001	8.882500e+07
75%	1079.235016	1088.949982	1070.205048	1079.265015	1079.265015	1.121850e+09
max	2867.229980	2872.870117	2851.479980	2872.870117	2872.870117	1.145623e+10

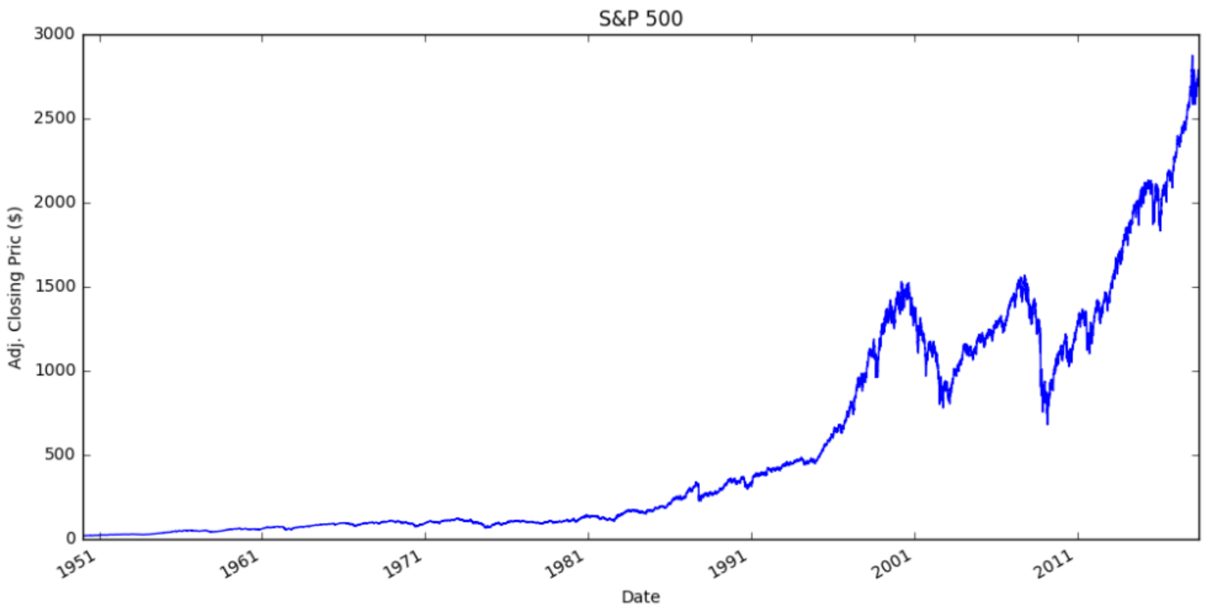
The difference of the lowest (16.66$) and the highest value (2872.82$) is quite big (adjusted closing price). Also the standard deviation of 646,46$ is high. This has to be considered during the data processing step in the implementation.

The data is pretty clean I didn't find missing or implausible values, which would require a special treatment. Before the year 1962 there are no differences between Open, High, Low and Adj. Close price are reported as you see below:

	Open	High	Low	Close	Adj Close	Volume
Date						
1961-12-20	71.120003	71.120003	71.120003	71.120003	71.120003	3640000
1961-12-21	70.860001	70.860001	70.860001	70.860001	70.860001	3440000

As this project concentrates on the adjusted closing price, and this values are far in the past, I assume this will have no negative impact on the predictions.

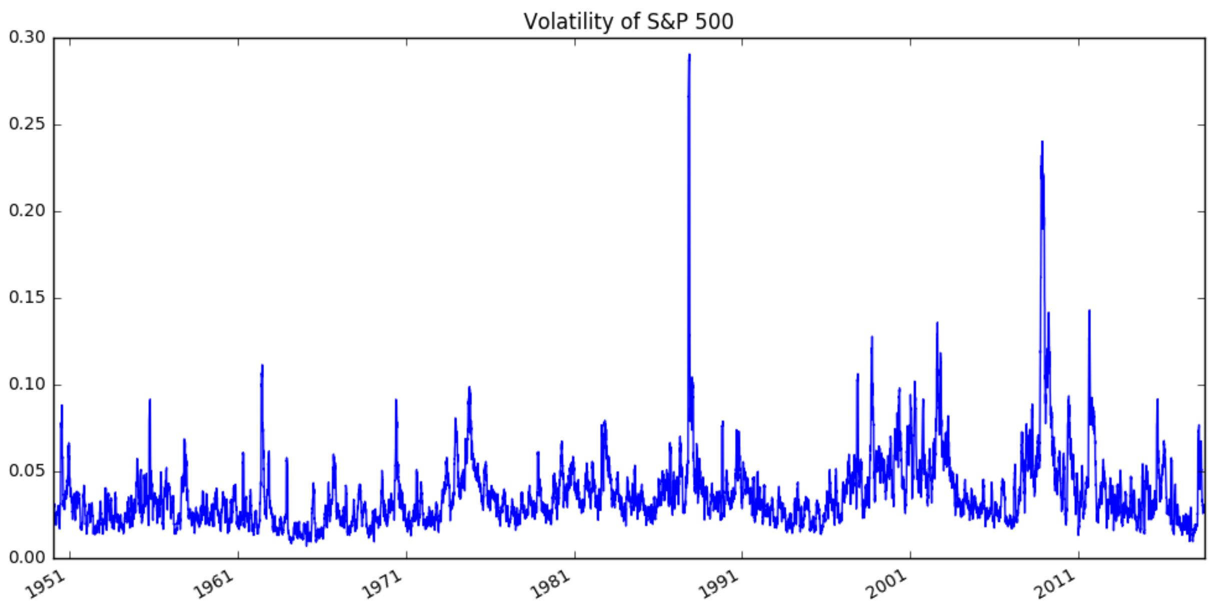
Exploratory Visualization



Above the history of the daily price of the S&P500 index is shown. Overall the price shows a clear positive long-term trend of the index, but incidents can affect the index in the short term heavily in both directions. In the year 2000 during the “dot-com-bubble” the S&P 500 reached a peak, followed by a down turn of the index until a low was reached in October 2002. The old peak was reached again in 2007, followed by a crash caused by the “subprime mortgage crisis”. During the last years the index moved mostly upwards with some downturn movements reaching an all-time peak in the beginning of 2018. Investors face the prediction challenge, if the upward trend will remain, if a plateau reached or will if a crash happen soon? Beside this long-term characteristics, the graph shows lot of up and down movements in the short term. This noise is caused by the trading activities happen on the market, based on the different subjective expectations how the stock prices will develop. A common measure of the degree of variation of a market price series is the volatility (σ), usually measured by the standard deviation of logarithmic returns. As we compare the volatility day by day, it is calculated by:

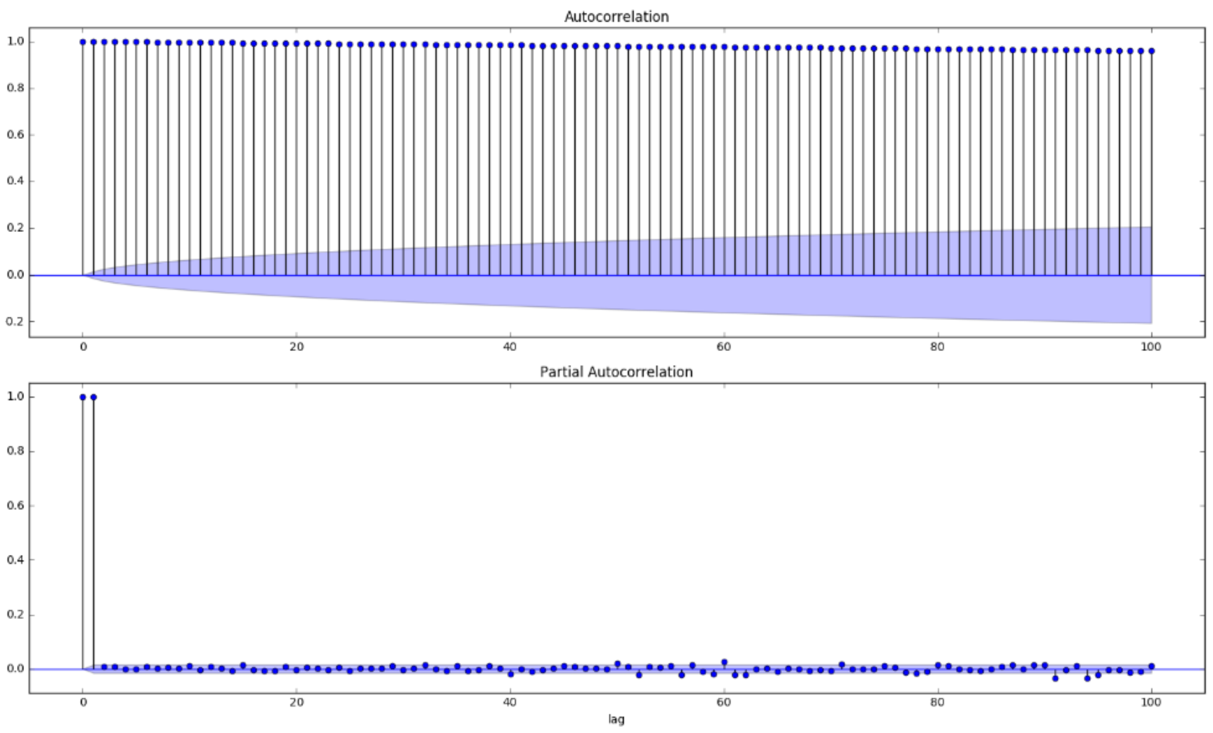
$$\sigma = ln \frac{adjusted\ closing\ price_{t1}}{adjusted\ closing\ price_{t0}}$$

In the graph below the volatility for the S&P 500 is shown.



Beside outliers caused by crashes or skyrocketing markets, the market faces always a fluctuating degree of volatility. This makes predictions of the stock market more difficult. Beside predicting the long time trend, the predictor has to include the volatility, to predict short term price movements.

To predict the adjusted closing price of the S&P 500 based on past prices, the prices have to be related to older or lagged prices. The autocorrelation measures the correlation with the lagged prices.



The autocorrelation and partial autocorrelation for the adjusted market price of the S&P 500 for lags until 100 is shown above. The blue area marks the 95% confidence interval, values above or below are likely correlated. The first graph shows the autocorrelation of a adjusted closing price and the lagged values. The graph shows a correlation between the values, which can help to predict the prices. The second graph shows the partial autocorrelation, removing the indirect correlation between the lagged values and showing just the direct correlation with a specific time step. Only the one day lagged value falls into 95% confidence interval, so values with a lag greater don't correlate direct with the value, which makes a prediction based on historic values more difficult.

Algorithms and Techniques

The previous analysis showed the complexity and non-linearity of the stock prices time series. In the last years machine learning gained popularity for various prediction tasks, and showed that it can deal with complex and non-linear relationships. Especially deep neural networks (DNN) made big improvements in performance. The stock price predictor uses special type of DNN called recurrent neural network (RNN).

RNN are good in extracting patterns in sequential data as in time-series data.[5] They have a feedback loop to past decisions, so they add a kind of memory to the network, that takes in account the information which persists in the sequence of the data itself. This important for forecasting stock-prices, because we have to take to in account the sequence of historic prices to predict future prices.[6] Simple RNN are good to deal with short-term dependencies. For stock-price predictions we have to deal also with long-term dependencies , that why a special RNN is used for this project, the Long Short-Term Memory (LSTM). LSTM can selectively remember and forget things, and so is able to learn long-term dependencies.[4]

The performance of an LSTM can be affected by the preprocessing of the data, the configuration of the network and the training parameters.

Preprocessing of the input (see also: Data Preprocessing)

- The split of training and testing sets
- Normalization and smoothing or other manipulation of the data

Configuration of the network architecture:

- Configuration of the input layer
- Number, configuration and type of hidden layers (In this case LSTM Layer)
- Configuration of the output of layer (linear activation function used)
- Number of nodes of a layer (various combinations tested)
- Drop-out layers (to avoid overfitting)

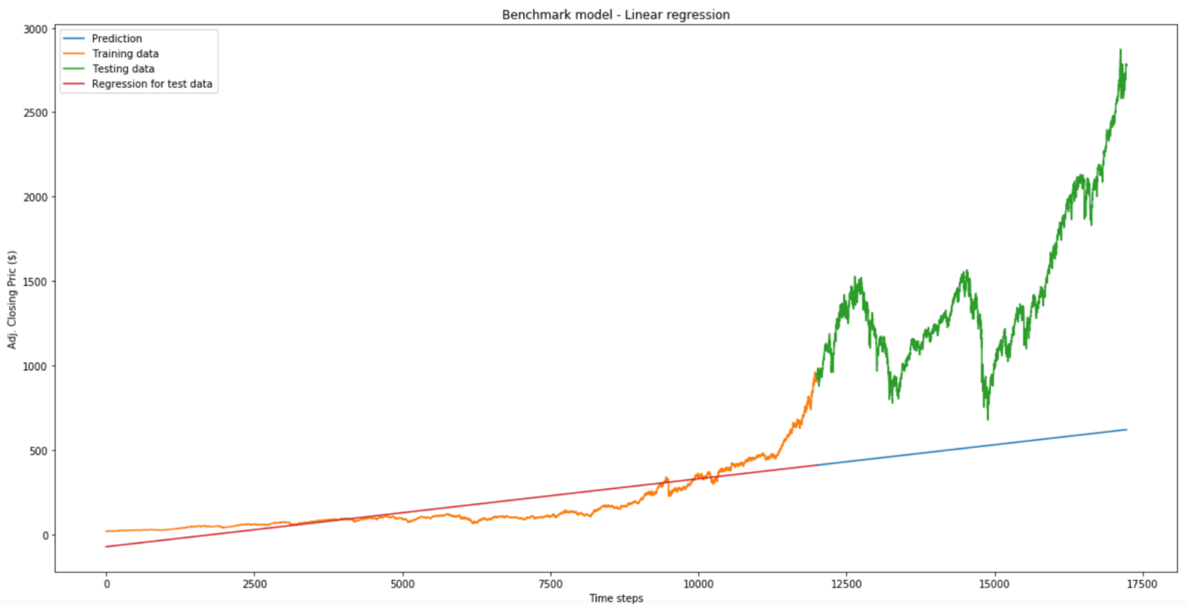
Training parameters:

- The optimizer function (e.g. Adam) which tries to minimize the error
- The loss function which calculates the error (in this case MSE is used)
- The size of the validation set
- The batch size for a training step (how often the weights of the network are updated)

- The number of epochs to train and criteria when to stop the training

Benchmark

A linear regression model is used as benchmark for this project. The linear model was trained by the training data and extrapolated for the testing period. The results are shown below.



For the training period the model can show the general long-term trend of the adjusted stock price, but at the end of the training period, the model fails to match with the rapid increase of the stock price. The extrapolation for the testing period, shows an underestimation of the long-term trend.

III. Methodology

Data Preprocessing

First historic stock market data for has been downloaded from Yahoo finance and stored in a data frame.

Splitting of test- and training data

The data has been split into a testing and a training dataset. I used the first 12.000 data points for training and the rest (5225) for testing.

Normalization

The MinMaxScaler was used to scale the data between 0 and 1. Because of the huge differences in the value range (from 16.66$ to 2872.82$ for the adjusted closing price) of the period, I split the data in six batches of 2000 to maintain the information of the earlier years for the learning.

Smoothing of the data

Because the data contained a lot of noise, I used a rolling mean for 5 days to smooth the data.

Generation of the target variable

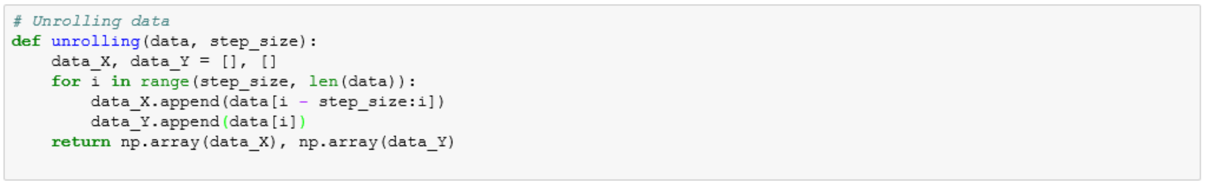
At time step $t_0$want to predict the stock price for t_1. To generate the target variable y_0 the the stock data has to shifted one time step ahead. The data for the last time step has now to be deleted, because it contains now NaN values for the target variable.

Generation of input sequences

Dependencies between past stock prices should be considered by the model. So the input data X_0 at the a time step t_0 includes of a sequence of i features for a defined time window (from $t_{t=0}$ to $t_{t=0-n}$) of n time steps into the past.

$$X_0 = [x_{0,0}, x_{0,1}, \dots, x_{0,i}], [x_{-1,0}, x_{-1,1}, \dots, x_{-1,i}], \dots, [x_{t-n,0}, x_{t-n,1}, \dots, x_{t-n,i}]$$

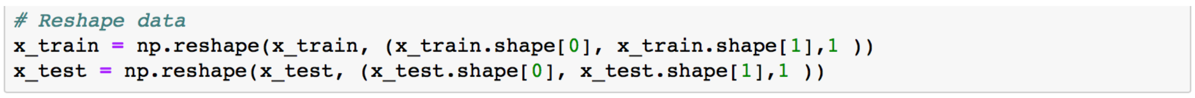
For the stock price predictor the history of 5 trading per data point is considered. For every time step an input sequence with the past stock data for this period is generated.



```
# Unrolling data
def unrolling(data, step_size):
    data_X, data_Y = [], []
    for i in range(step_size, len(data)):
        data_X.append(data[i - step_size:i])
        data_Y.append(data[i])
    return np.array(data_X), np.array(data_Y)
```

Reshaping of the data

The input data is reshaped into a 3D array to fit LSTM network.



```
# Reshape data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1 ))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1 ))
```

Implementation

LSTM-Network architecture

The first step was the implementation of the LSTM-Network architecture, which was iterated in many steps. The final solution is a sequential model with three LSTM-layer (200,200 and 150 nodes) each followed by dropout layer (dropout = 0.2) to avoid overfitting. Finally a dense layer with a linear activation function was added, to output the predictions.

```
#Building sequential model
model = Sequential()
#Input layer
model.add(LSTM(200,input_shape=(window,6), return_sequences=True))
model.add(Dropout(0.2))
#1. Hidden layer
model.add(LSTM(200,return_sequences=True))
model.add(Dropout(0.2))
#2. Hidden layer
model.add(LSTM(150,return_sequences=False))
model.add(Dropout(0.2))
#Outputlayer
model.add(Dense(6,activation='linear'))
```

Then the model was compiled and fitted. Using the mean squared error as loss function and the optimizer "Adam". Early stopping was used for to stop the training when the difference of the validation loss between fall below a threshold (0.00005) and certain number of periods no improvement was made. 2% of the data were used for validation.

```
#Compile the model
model.compile(loss='mse',optimizer='adam')
#Fit the model
hist = model.fit(x_train, Y_train, epochs=100, batch_size=2500,
                 callbacks = [EarlyStopping(monitor='val_loss', min_delta=5e-5, patience=20, verbose=1)],
                 validation_split=0.2
```

One day ahead predictions

Based on the model, a simple prediction was made using the predict function and the test data.

```
prediction = model.predict(x_test)
```

This delivered predictions cover the whole timeframe of the test data, but every prediction is based on the train data of the day before. So this are only one day ahead predictions, and not real multiday or long-term predictions.

Multiday predictions

To predict more than one day into the future a more complex procedure is necessary. The predictions have to be done step by step for every time step. To predict two days ahead, the prediction for the previous day has to be considered. To predict three days ahead, the predictions for the previous two days have to be considered and so on. The input sequence of stock data of the previous time steps has to be updated for every prediction step.[9]

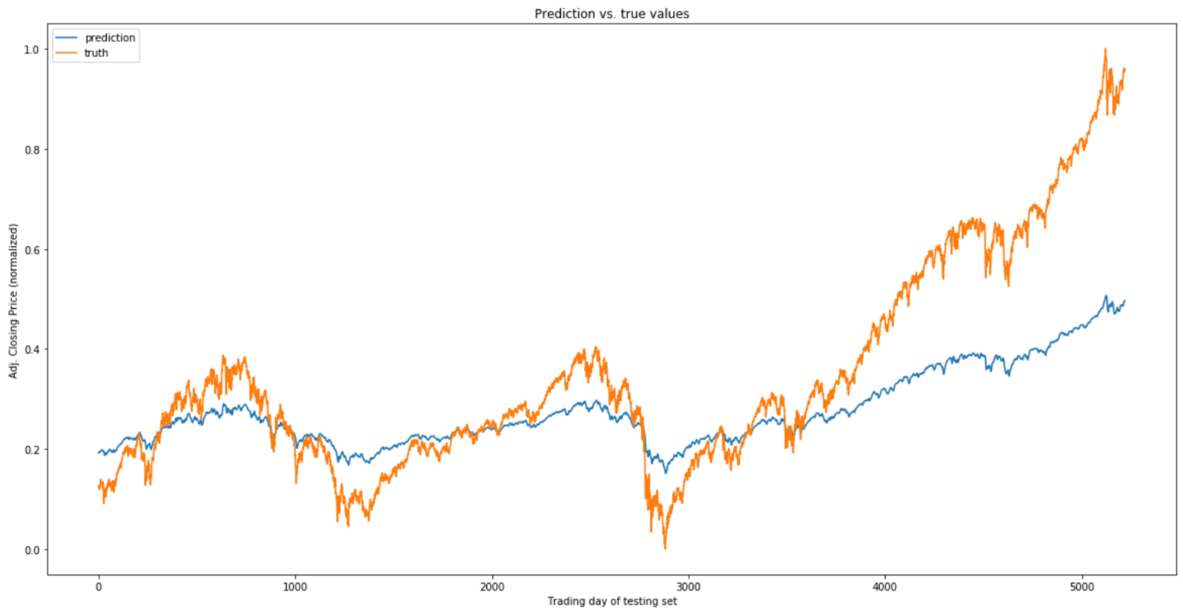
```
#Number of predictions to be made
prediction_seq = 30
#Lenght of a prediction cycle
prediction_len = 50
prediction_all = []

#Predict a sequence of multiple predictions
for i in range(prediction_seq):
    prediction_frame = x_test[i*prediction_len]
    predictions = []
    for i in range(prediction_len):
        prediction = model.predict(np.array(prediction_frame).reshape(1,window,6))
        predictions.append(prediction)
        prediction_frame = prediction_frame[1:]
        prediction_frame = np.insert(prediction_frame, [window-1], predictions[-1], axis=0)
    prediction_all.append(predictions)
```

Refinement

I started the project with a very simple network:

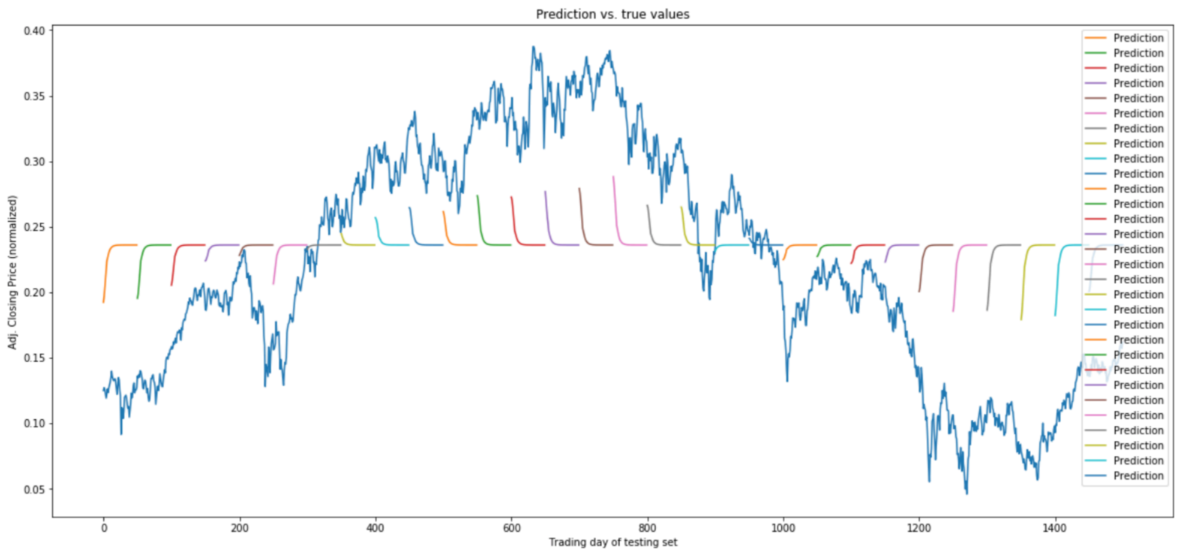
```
model = Sequential()
model.add(LSTM(16,input_shape=(window,1), return_sequences=True))
model = Sequential()
model.add(LSTM(32,input_shape=(window,1), return_sequences=False))
model.add(Dense(1,activation='linear'))
```

I used for training and testing only of 5 timesteps, trained the network for batches of 2000 over five epochs. The prediction quality for the one step prediction was not statisfying:

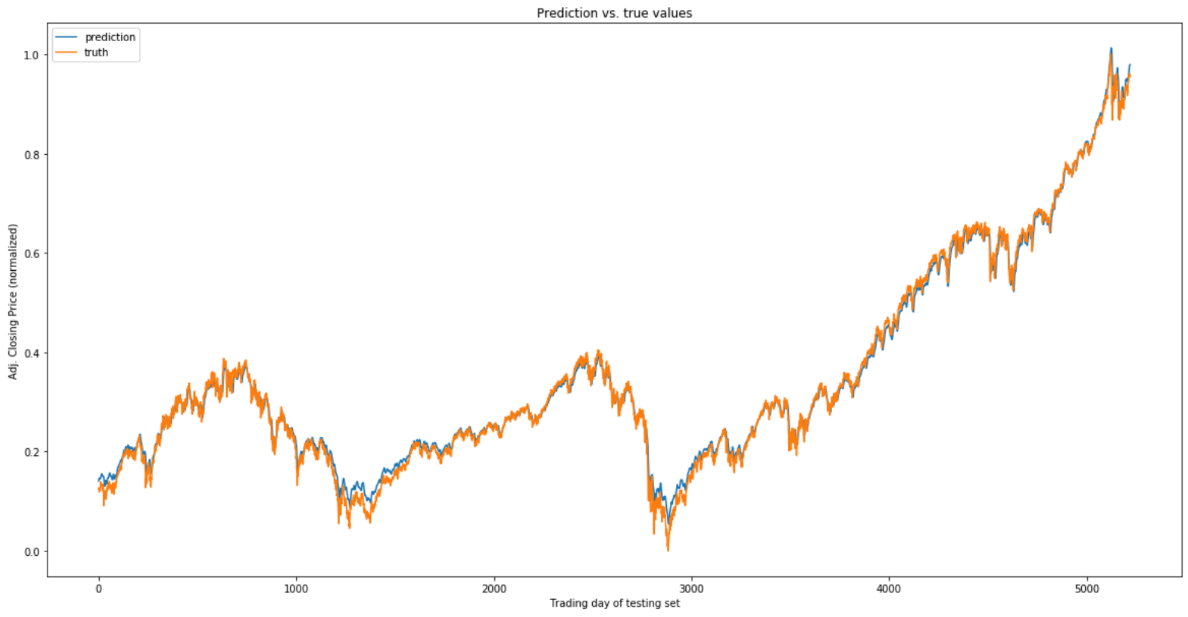


```
Mean Squared Error:  0.0221715383938
```

The results of the multiday predictions were even worse. There was no meaningful learning visible:

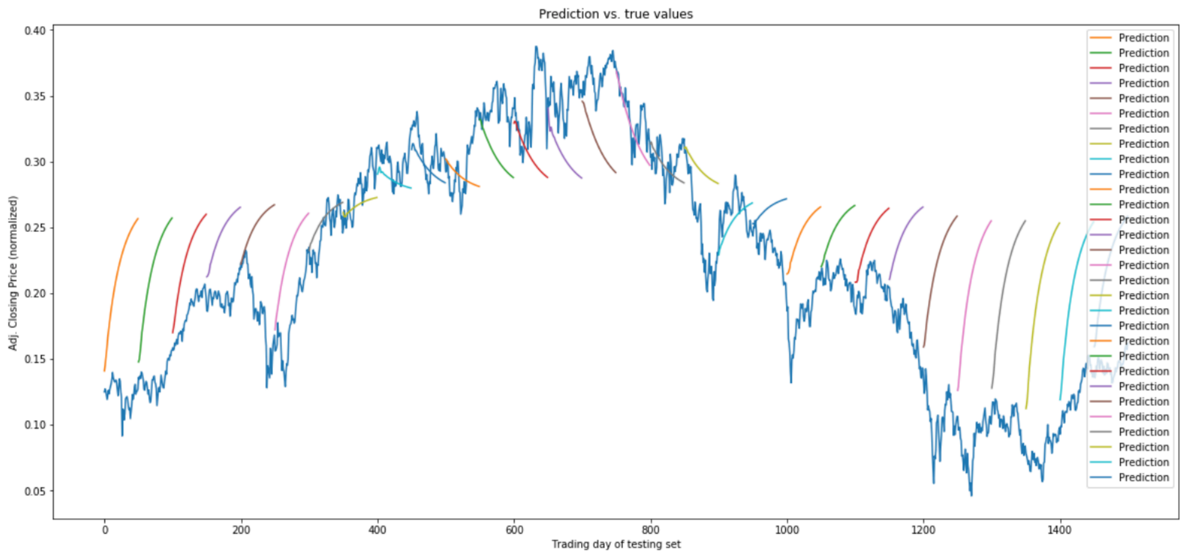


After reducing the batch size to 1000 and allowing the training for 31 epochs, the quality of the one day predictions increased clearly:



```
Mean Squared Error:  0.000240002148027
```

The predictions for multiple time steps stayed unusable, but there was some kind of learning of some trends noticeable:



Many different parameter combinations were tried, to obtain the final solution. The network architecture was extended. As shown in the implementation chapter, two extra hidden layer were implemented, the number of nodes was increased and drop out functions were implemented to prevent overfitting.

Different types of optimizer were tried (RMSProp, Adagrad, SGD, Adam) but Adam show the best performance.

Smoothing the data with a five days rolling mean helped zo reduce the noise in the data.

The window size how many time steps back should be taken into account for training and testing of the features, had a huge impact I tried lots different sizes between 1 and 252 time steps. The best result I obtained with a window of five time steps.

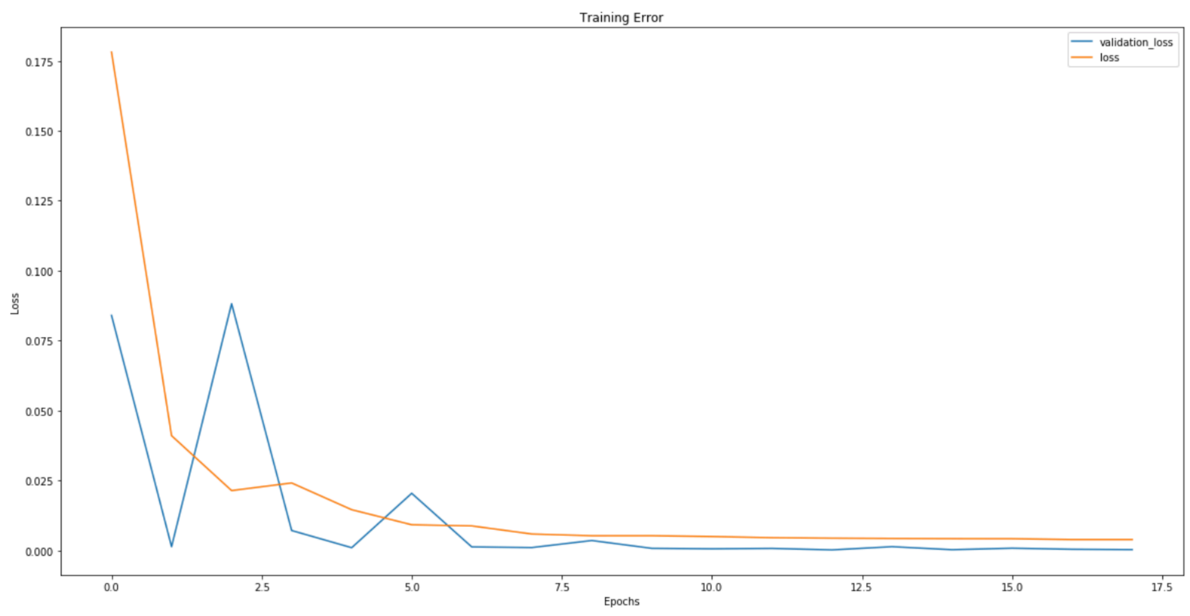
The batch size also had a big impact on the predict performance. I tried different batches sizes between 1 and 5000. With a batch of 2500 the best results were obtained, I was surprised that even little changes of this affected the performance badly.

I also tried to use the other features (open price, closing price, the lowest price , the highest price and the trading volume) for prediction. But it didn’t seem to have a positive impact on the predictions. One problem is, that for a multiday prediction sequence this features also have to be predicted, and the predictions more than one day ahead are based on this predictions. Another problem is that the difference between the different price values for one day is not so big as the price changes over many days itself, so they may not able explain long term trends of the price.

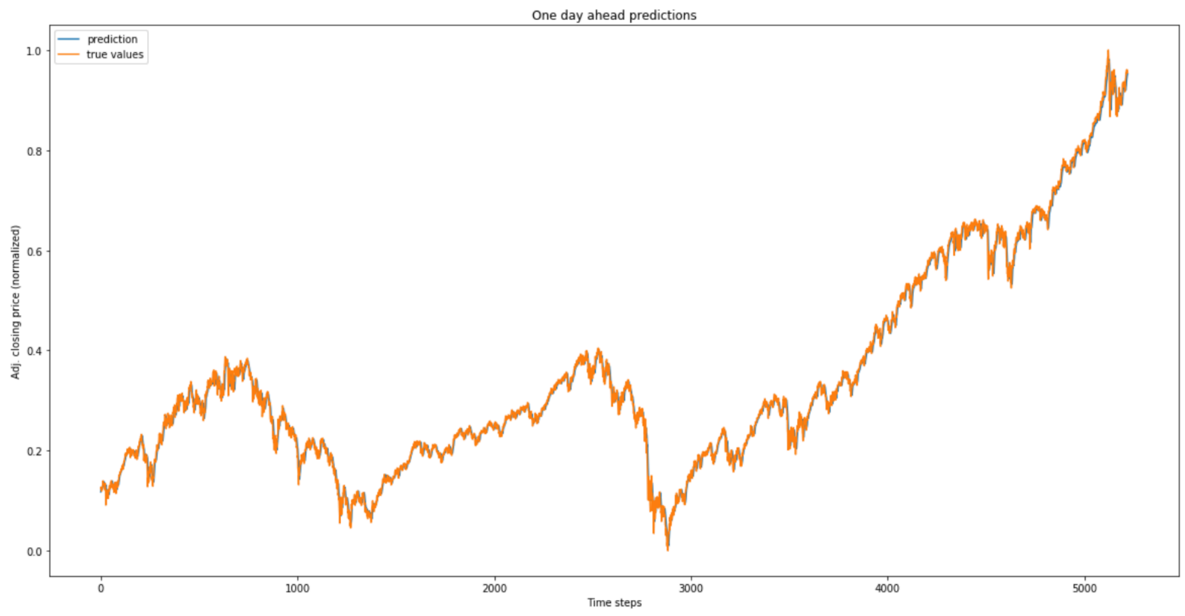
IV. Results

Model Evaluation and Validation

After many iterations of tuning the parameters, the final model showed a good performance with a very low training and validation loss as shown below.

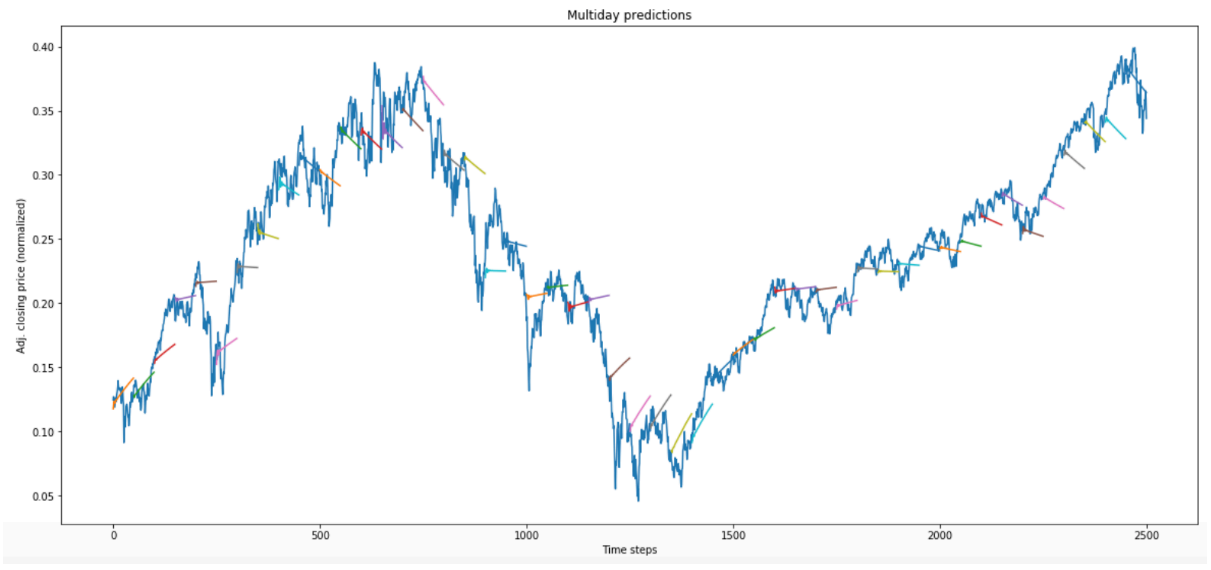


After six epochs the loss reached a low level with stayed stable until epoch 17. The one day ahead predictions showed also a very good match with the true values and a very low MSE as shown below.



```
Mean squared error:  0.000136034033906
```

The multiday predictions for fifty days ahead started finally to show some kind of learning, even if the match to the true values is not very accurate. Below are shown, how fifty predictions each over fifty days (the small colored lines) perform compared to the true values for the testing data (blue line).



Even if the predictions are not very accurate, the most of them seem to anticipate a general trend of the stock price. But after approximately timestep 400 and 1600, it is noticeable that the decreasing prices are predicted and not increasing prices. The analysis shows that reliability and robustness of the model for multiday predictions is limited. Lots of iterations of parameter tuning and different network configurations could not improve the performance of the predictor more.

I assume, one reason is a degree of randomness of the stock market. Another reason are the various other factors that affect stock prices (as economic crises, wars, politics, availability of new technologies…) which are not included in the model.

Justification

The MSE for the single day predictions (0.00014) is compared to the benchmark model (9647.05656) shows a huge advantage of the LSTM-model compared to the linear regression model. The plots of the predictions compared to the true values confirm this.

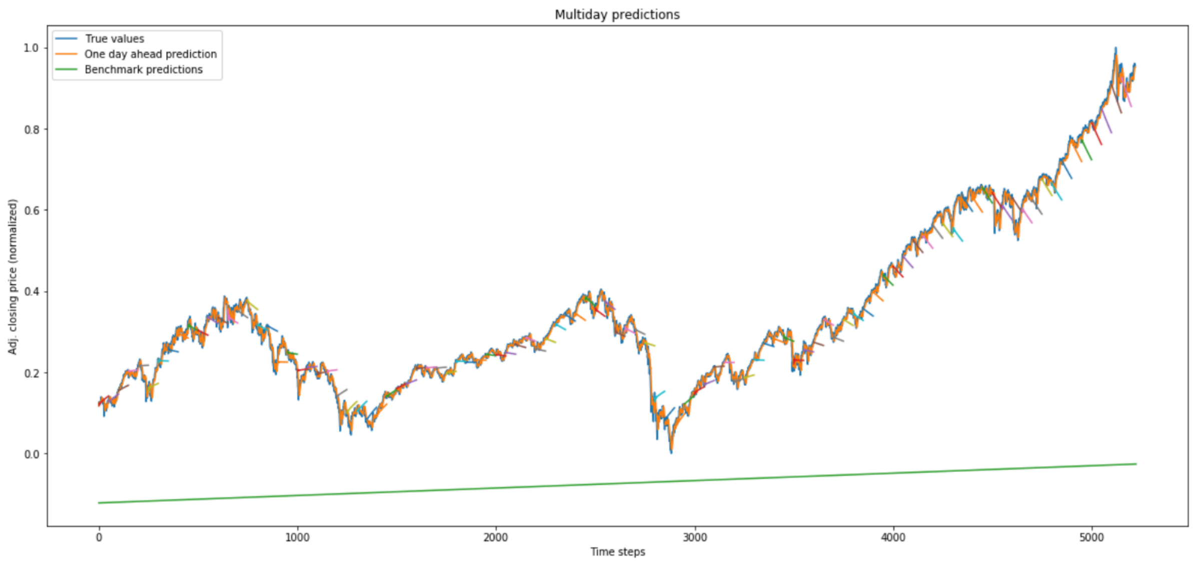
But to support long-term investment decisions, this might not sufficient, because changes between two days of the price are usually not so big as the long-term changes we saw during the data exploration. The multi days prediction approach tries to address this problem, but the visualizations of the predictions not show not sufficient reliability to guide investment decisions alone.

One problem could be, the training data shows different trends and other patterns than the testing data. As we saw in the visualizations of the explorative analysis, that the stock price followed in the training period mostly steady but slow growth. But in the testing period we see big ups and down and an overall growth, which exceed the historic trend a lot. This could be caused by factors as economic crises, wars, politics, availability of new technologies and many more, which are not included in our model. If we want to predict the future behavior of the stock market, we have to take into account these factors. The model developed in this project, could contribute into a more holistic stock prediction approach, were also other factors are included.

IV. Conclusion

Free-Form Visualization

The final visualization compares the true prices of the S&P 500, the one day ahead predictions, the multiday predictions and the benchmark model for the whole period of the testing data.



The one day ahead predictions match the true data so good, that the plot of the predictions mostly overlap the true values and it is hard to spot the plot of the true values.

On the other hand, the prediction based on the benchmark model underestimates the rise of the stock price heavily.

The multiday predictions (the small colored lines starting close to the plot of the true values), showing a mixed picture, they don't match with the true values but at some points they give an indication of the future trend. But when the stock price rises strongly and moving into the direction of a peak, the multiday predictions guide wrongly the false direction (falling prices). As mentioned before, the testing data period didn't contain this kind of pikes and rapid increases of the stock price.

The visualization shows results of the stock price predictor exceed the prediction of the benchmark model. But it also shows that the reliability of the results of the multiday predictions limits its use for guiding investment decisions.

Reflection

I started to downloading the Data for the S&P500 and exploring it. It was interesting to see the characteristics of the data, especially how noisy the stock market data is, what make predictions so difficult.

The challenges of the preprocessing step were to bring the data in the right 3D-shape for the LSTM algorithm and generating sequences of multiple time steps for each data point. Normalization and smoothing of the data was also conducted.

The most time I spent with the configuration of the LSTM-network and the tuning of the parameters. I was surprised by the big impact of the batch size and the length of sequences of time steps feed into the network on the prediction performance.

The performance of the one day ahead predictions exceeded my prior expectations, but I think the use of these for real stock investment decisions is limited, because the changes of stock prices between two days are usually not so big.

To find an algorithm to predict and visualize multiday ahead predictions was a quite laborious task. But the reliability of the predictions is limited, I would not put my money on these. But it is a good demonstration of the challenges to predict the stock market. And it could be a starting point to develop a predictor, which combines different methods of prediction and different sources of input data.

Improvement

Including more features

In this project the adjusted stock price was used for the prediction. The dataset contained more features, which I didn't for the final solution, my attempts to used them for prediction were not successful and didn't deliver better predictions. But maybe there are other stock market features are available or the features could be engineered to enhance the prediction performance.

Including other features

As discussed before, a lot of factors influence the behavior of the stock market. The model should not only rely on historic stock data. For the prediction of the stock prices, the data of the performance of companies or of the development of markets in general could be useful. Many sources of financial data and other economic indicators could be included. Furthermore companies or and analysts release regular report of the current and expected performance of companies, which could be a valuable source of input data.

Combination with other prediction approaches

Different approaches for stock price prediction are available. An example are sentiment analysis which can be used to predict the behavior of stock market related to news (e. g. from news channels or twitter) which are related to a company or the business in general. Good news likely cause an increase of the stock price and vice versa. Combining different approaches could increase the prediction performance.

Use of the approach for intraday predictions

It would be interesting to investigate the use of this approach for high frequency intraday trade. How would the model fit to this data? Are there patterns in intraday data which can be used for predictions? Or is the noise in this data even bigger.

References

1) Efficient market hypothesis, Makiel, Burton G. (1992), New Palgrave Dictionary of Money and Finance, Macmillan, London
2) https://www.ft.com/content/6b2d5490-d9bb-11e6-944b-e7eb37a6aa8e
3) https://www.ft.com/content/4f5720ce-1552-11e8-9376-4a6390addb44
4) Hochreiter, Sepp & Schmidhuber, Jürgen. (1997), Long Short-term Memory, Neural computation
5) Melvin, Sreelekshmy & R, Vinayakumar & Gopalakrishnan, E. A. & Menon, Vijay & Kp, Soman. (2017). Stock price prediction using LSTM, RNN and CNN-sliding window model. 1643-1647. 10.1109/ICACCI.2017.8126078.
6) Ganegedara, Thushan (2018), Stock Market Predictions with LSTM in Python (https://www.datacamp.com/community/tutorials/lstm-python-stock-market)
7) https://keras.io/
8) http://colah.github.io/posts/2015-08-Understanding-LSTMs/
9) http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction