Machine Learning Engineer Nanodegree

Capstone Project

Alexander Keller
August 26th, 2017

I. Definition

(approx. 1-2 pages)

Project Overview

Forecasting time series, in the stock-market, futures market, currency market or any other market can be beneficial to personal and commercial motivated investors. It will give the individual the necessary knowledge and assurance to make an educated guess rather than using random choices or the gut feeling. Methods to do exactly that are available from the beginning of a market and evolve ever since. In general there are:

- Random Walk analysis - the idea that price fluctuations are totally random and unpredictable
- Efficient Market Hypothesis - similar approach as before, but added idea that all known information is (already) reflected in the price
- Technical Analysis - historical data shows patterns/trends/... which repeats itself in the future
- Fundamental Analysis - relates the price to supply and demand, also to input given by media
- Sentiment Analysis - uses the traders gut feeling or the general mood of the market to predict

While there is way more to distinguish, like the markets affect each other, it should be outlined that this project is about using technical analysis indicators and utilize soft computing algorithms, a machine-learning algorithm, to make and educated guess and try to predict the future price in different time-frames.

It's not new to use SVM or SVR in order to predict the FOREX market. Moreover there are also ANN approaches which are very promising to predict the well known currency markets. All of them predict in medium-term(days) to long-term(months, years) time-frames. In this project I like to figure out if I can do it (just as many others before) on a small scale and if a very low error means good potential profit. There is plenty of data available and for free as well. The dataset consists of EUR/USD tick-data (one datapoint per price change) from May 2009 to June 2017. The data was downloaded in zipped CSV files seperated in on file per month from [http://www.truefx.com]

Given the high amount of data(about 26GB of CSV data) only the resampled data is part of the report.

Problem Statement

The aim is to successfully implement a Python script which forecasts the next days close price. Based on its nature I will use a regression algorithm to solve that issue. More specific a Support Vector Regression, since prior small experiments showed that Logistic Regression might not that powerful. To solve the set task, I like to go through the following steps:

- get the data and resample it to one or more suitable timeframes
- find a benchmark to compete against
- extract and choose some features from the dataset
- train the regression model
- tune the model
- define a scenario (profitable or not)
- conclude and discuss further steps

The result should be a Jupyter notebook which can be used as a hint whether or not it's worth to put more effort in that field and maybe implement a machine-learning algorithm in a trading-bot.

It should not be the aim to 100% predict the next days close, as stated above only to guess if its possible to make a good guess and if it's profitable under a specific condition.

Metrics

For this problem I chose the R2 score for a quick indicator if I am on the right track by determining the features. Later on I added the Mean Absolute and Mean Squared Error to evaluate how good the model performs in general and how good the predictions are under specific condition. Given the fact that it is a regression problem, I chose those mentioned metrics. Reasons are for one that it is best practice for that type of project and it's a good indicator how well the data fit the data. Another reason is of course that it was thoroughly discussed during the lectures. In more detail, the

- R2 score is defined as:
  - $$1 - \frac{\sum\left[\left(y_{true} - y_{pred}\right)^2\right]}{\sum\left[\left(y_{true} - \bar{y}_{true}\right)^2\right]}$$
  - where y_pred are defined as the predicted values, y_true the real values and y_true with overscore is the mean value of the real data.
  - it's pre-defined in the score method of the SVR algorith in Scikit-Learn
- Mean Absolute Error defined as:
  - $$\frac{1}{n}\sum\left|y_{true} - y_{pred}\right|$$
  - where y_pred are defined as the predicted values, y_true the real values and n is the number of samples
- Mean Squared Error defined as:
  - $$\frac{1}{n}\sum\left(y_{true} - y_{pred}\right)^2$$
  - where y_pred are defined as the predicted values, y_true the real values and n is the number of samples

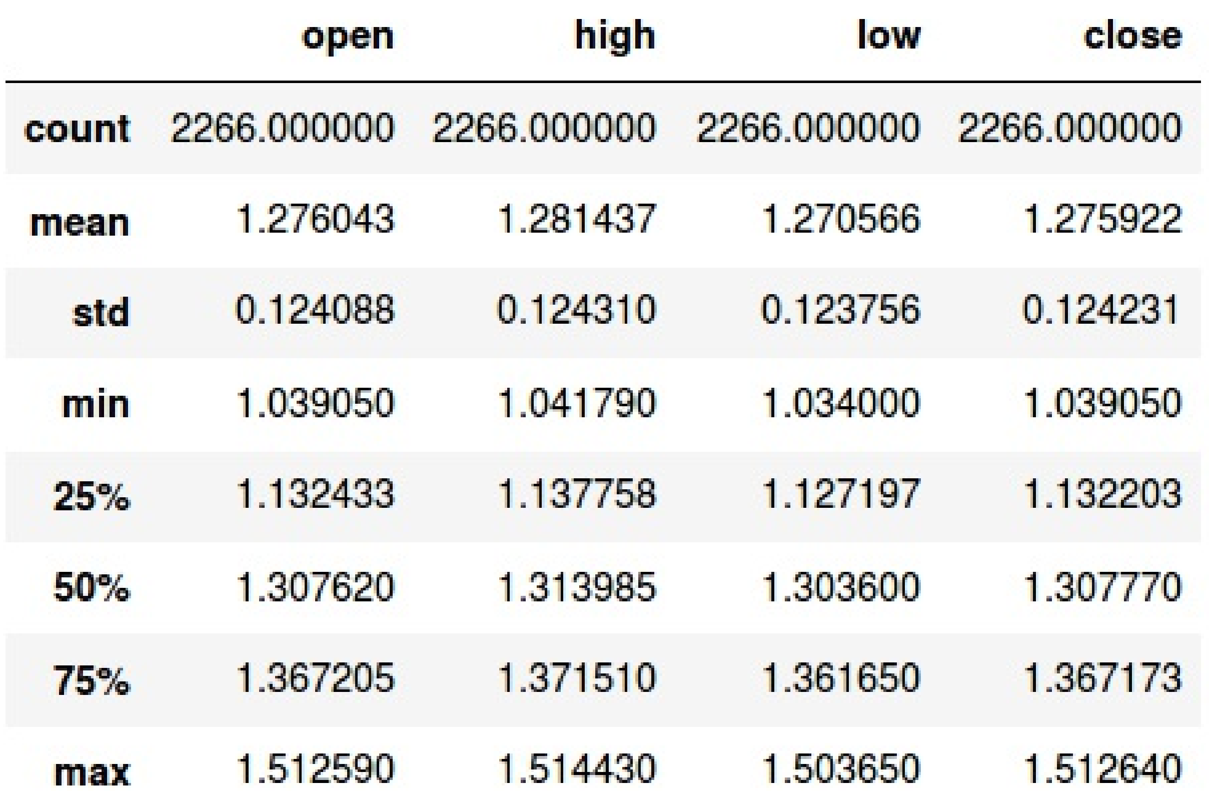
II. Analysis

(approx. 2-4 pages)

Data Exploration

The data for the project is public available and was downloaded from [http://www.truefx.com]. The downloads needs a free and obligation-free registration. The dataset consists of tick-data from May-2009 to June-2017. Each month is provided in a separate zipped CSV-file. Each sample provides the Data, the Symbol, the Ask-price and the Bid-price. The first obstacle was the to resample the datapoints. There occurred a lot f memory errors with pandas and numpy due to the high amount of datapoints during the resample. The resampling was done in in two steps to solve that issue. The first step took a years data (12 files) and resampled it to the specific timeframe and saved it to one CSV-file for each year. The second step then combined the years data and saved it to one final CSV-File. I was interested in the hourly and daily timeframe so two classifiers were trained. The resampling resulted in a Low-, High-, Close- and Open-price for each timeframe. There were also NaN values in the dataset, since the foreign exchange is only open during weekdays and the resampling generated those values. Those NaN values were simply dropped. The features for the dataset had to be generated/calculated. For that problem I chose:

- n-days past close price - I had to figure out which might be a good choice for past close prices in order to predict the next days close price, I tested and used the score for each to choose an appropriate n
- Simple Moving Average SMA - the SMA describes the average price over a specific period of time in a given timeframe (a weighted average), the periods were 5, 8 and 13 since they are recommended in [http://www.investopedia.com/articles/active-trading/010116/perfect-moving-averages-day-trading.asp]
- Exponential Moving Average EMA - similar to the SMA it gives an average price, where the EMA is more sensitive to recent price-changes and removes some lagging effects of the SMA, the periods here were 12, 26 since it's a recommendation in [https://www.dailyfx.com/forex/education/trading_tips/trend_of_the_day/2014/01/20/The_3_Step_EMA_Strategy_For__Forex_Trends.html] I also tried an oscillator indicator like the MACD or RSI but it just made the predictions worse. So, I decided to only use oscillators as an independent indicator for over-bought or under-sold situation in order to make a final decision (if so). The resampled data gave 2266 data points (before feature creation) for the daily data. Pandas describe method



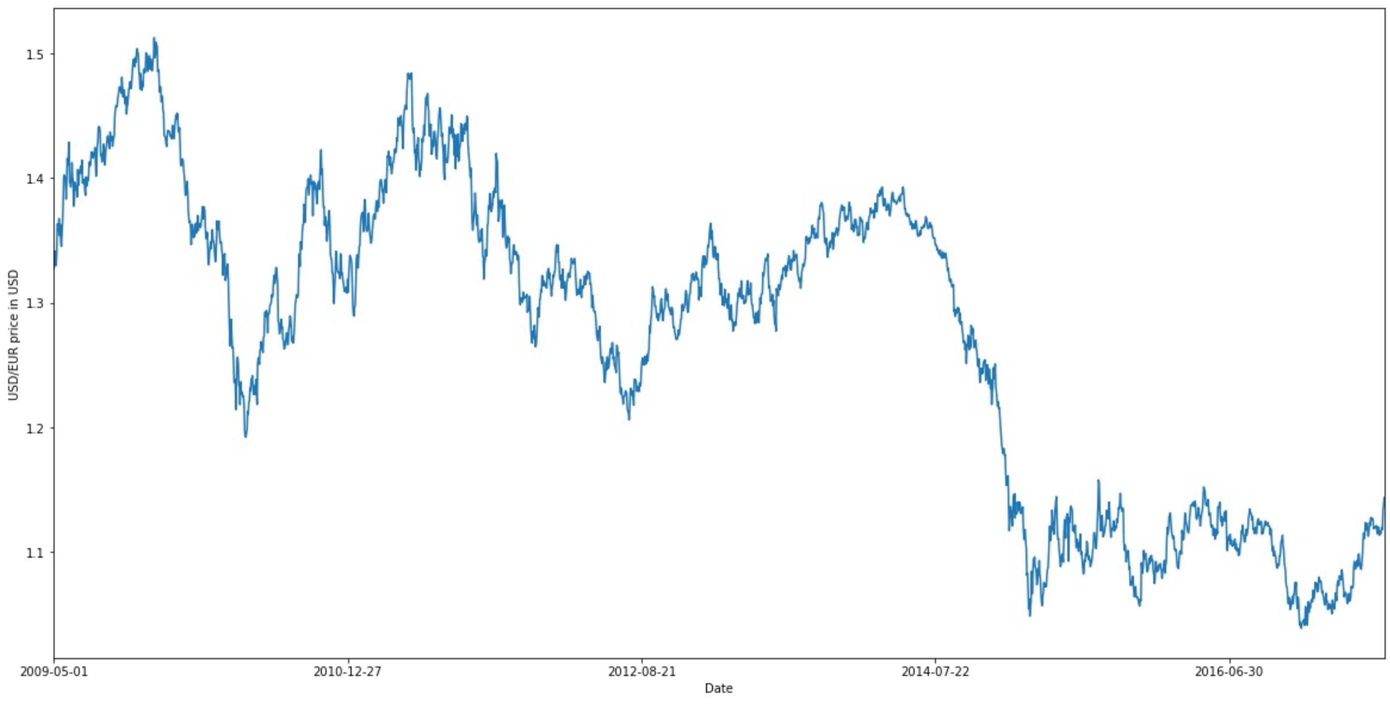
	open	high	low	close
count	2266.000000	2266.000000	2266.000000	2266.000000
mean	1.276043	1.281437	1.270566	1.275922
std	0.124088	0.124310	0.123756	0.124231
min	1.039050	1.041790	1.034000	1.039050
25%	1.132433	1.137758	1.127197	1.132203
50%	1.307620	1.313985	1.303600	1.307770
75%	1.367205	1.371510	1.361650	1.367173
max	1.512590	1.514430	1.503650	1.512640

shows that the data is well balanced. Given a definition of an outlier to be more that two standard deviations from the mean away, the dataset has no outliers. Given a kurtosis for the 'close' values of about -1.10 the dataset does tend to have outliers or heavy tails. Given the description of the data, the close values seem very symmetric (quartiles). The skewness of about -0.357 shows the data is little bit more to the left or more lower values. Which might indicate falling prices.

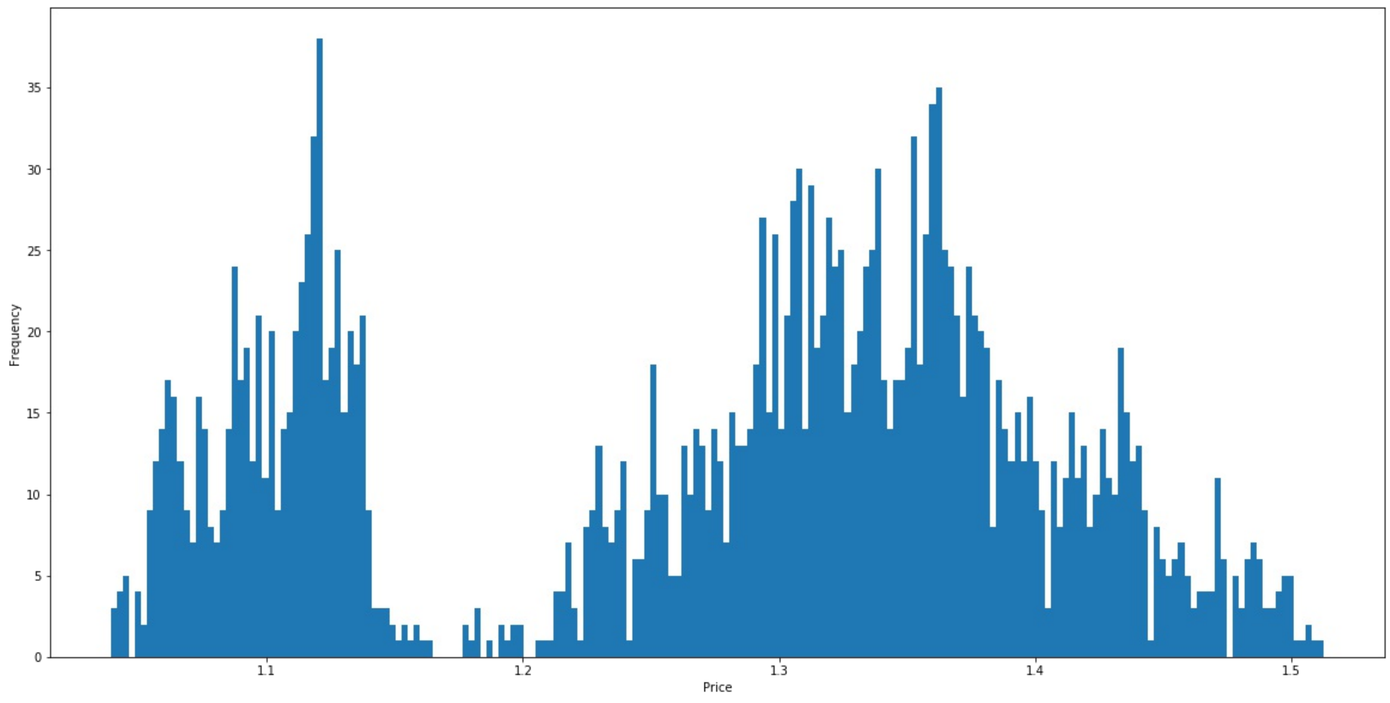
The NaN values were already dropped as part of the preproccessing.

Exploratory Visualization

The following shows the daily closing price data for the EUR/USD currency pair from May-2009 to June-2017.



The price chart looks like the price for EUR was falling until the end of 2014 and the went sideways until the end of the data. Taken the Histogram



It looks very interesting that there are somehow two 'populations', one around 1.13 with a little bit skewed to the left and another more symmetric one around 1.36.

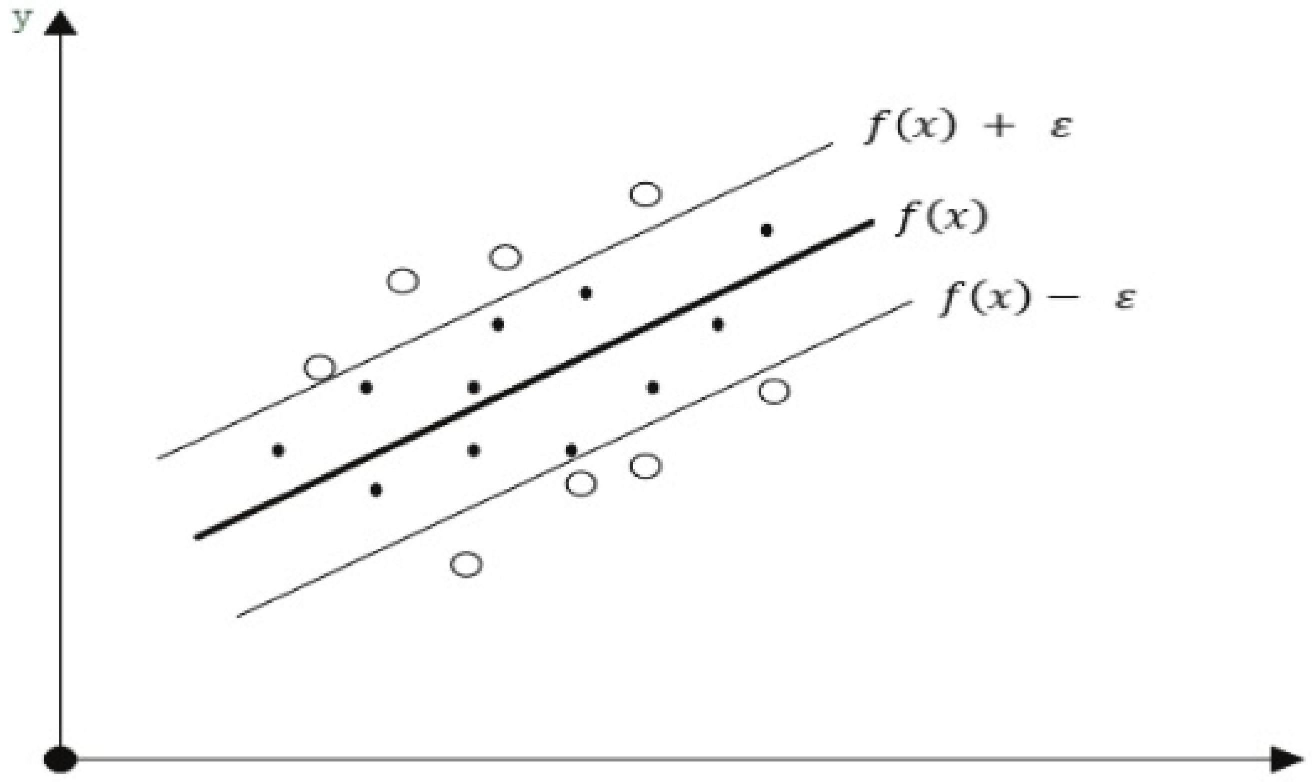
Algorithms and Techniques

For regression problems, the first thing to use might be Logistic Regression or Ridge Regression for that matter. Furthermore Neural Networks are a good way to go (many publications available) and Support Vector Regression are good choices. In that problem I decided to go with Support Vector Regression. The reason not to choose is stated in [Financial time series forecasting using support vector machines, Kyoung-jae Kim, 2003

ELSEVIER] and essentially Neural Networks don't generalize that good as Support Vector Machines (although newer algorithms might change that). In a first approach I quickly compared Logistic Regression with Support Vector Regression (SVR) and the latter performed slightly better, both with standard values from the Scikit Learn Python Library. The SVR algorithm takes the dataset {(X,y)} with X as the feature vectors as input and y as the target output vector. Assumed that data represents an unknown function, in order to predict future data there has to be a function

$$f(x) = \sum_i \omega_i \phi_i(x) + b$$

that approximates the unknown function well enough. Phi(x) represents the non-linearly mapped input data while omega are the coefficients of the support vectors and b is the distance of the support vectors spanned hyperplane from the the origin. SVR is hereby the application of the in the lectures introduced Support Vector Machines. In contrast to SVM to separate different data from their classes as much as possible, the SVR tries to find a hyperplane that fits the data points within a given tolerance epsilon.



The black dots within the "channel" defined by +-epsilon don't account for the loss function where the distances y of the circles to f(x) are taken in account to the total loss and influence the weights of the support vectors. The SVR algorithm tries to solve

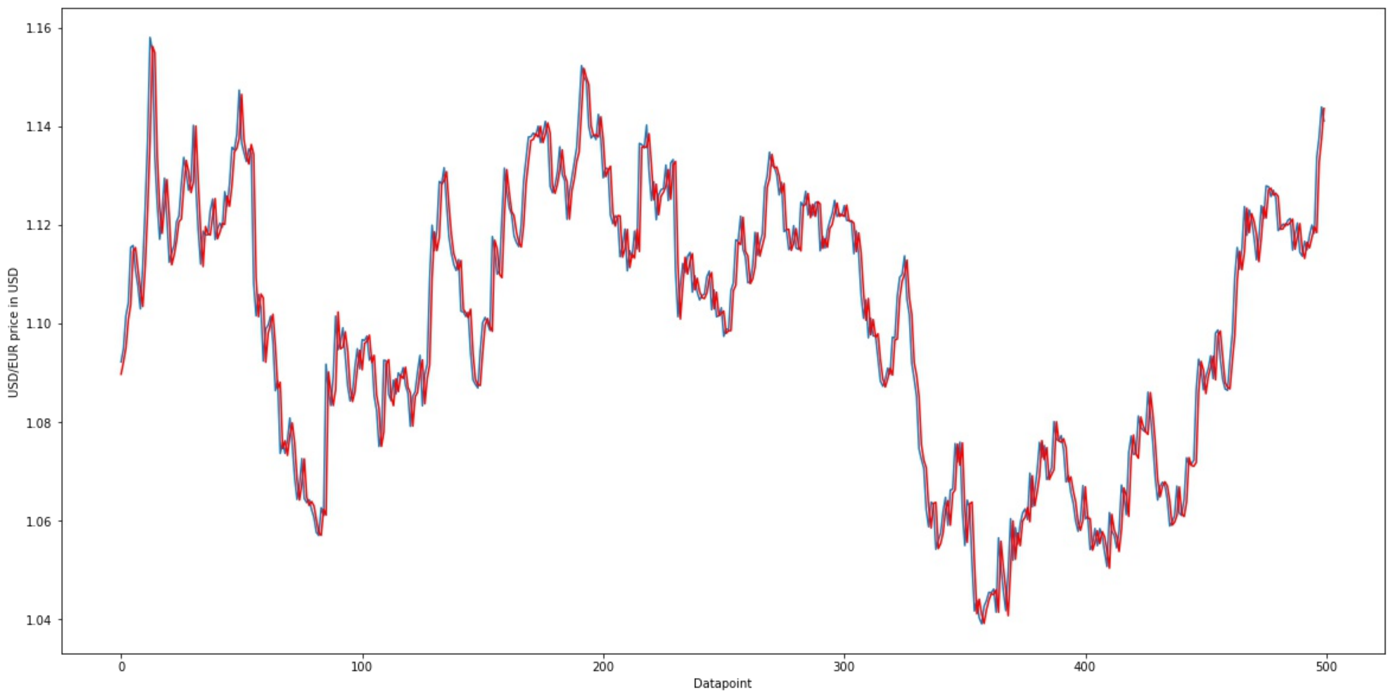
$$\min_{\omega,b} \frac{1}{2} \parallel \omega \parallel^2 + C \sum_{i=1}^{n} \ell_\varepsilon(f(x_i) - y_i)$$

Where C is a constant greater zero and affects the generalization of the function. l_epsilon is a epsilon insensitive loss function l(z) where z is the distance of the data point to f(x). So, l(z) is 0 for |z| <= epsilon and |z| for |z| > epsilon. So, the SVR tries to find a hyperplane that fits all the input data within a specific range epsilon to it.

To decide which time window (how many past closed prices) I trained the a LinearSVR and then checked the score (MSE in SVR classifier) for the 1-50 past close prices and chose the one with the lowest error. The parameters for the SVR were the standard parameters given by the Scikit Library. Later I let Grid search from sklearn mode_selection library find out, which were the best parameters to use.

Benchmark

I like to use an ARMA rolling forecast model as an benchmark for comparison. Those models have been widely studied and used for forecasting timeseries [Time Series Analysis: Fore- casting and Control]. So, for comparison the MSE and MAE of a standard ARMA model (p = 5, d = 1, q = 0) is the benchmark model to beat or not... The training data was about 80% and the test data, which was incrementally added and retrained as part of the rolling forecast, was about 20%. The algorithm was build as described in [http://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/] Usually for the ARIMA model the autocorrelation function is used for deciding the AR-MA parameters, but the values would have been around 150 and that was too much for my computer. The performance for the given problem is as follows:



It's very impressive to see how well that well known algorithm the data approximates(red). Although the lagging given by the nature of the moving average is noticeable. The metrics are approximately:

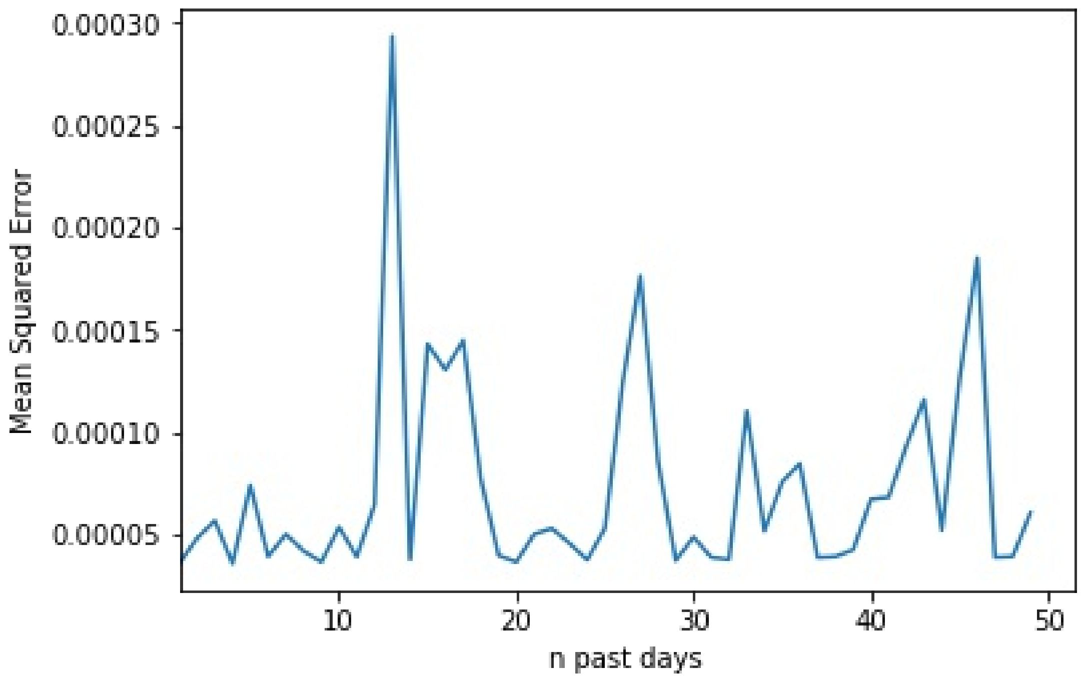
- R2 Score: 0.95
- Mean Absolute Error: 4.48*10^-3
- Mean Squared Error: 3.59*10^-5

III. Methodology

(approx. 3-5 pages)

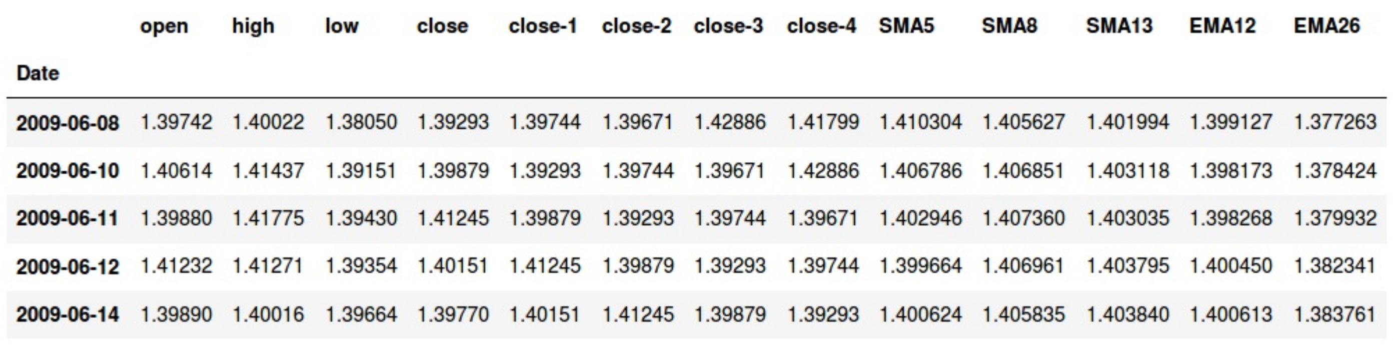
Data Preprocessing

As explained above the first thing to do was to create one (big) dataset out of the small one. The algorithm for daily data and other time-windows are the same with the difference of the time-window. I used pandas to read the data in and resampled the data and concatenated it to a new data frame for each month of each year. After that saved it as a separate CSV file. Codecells 1 - 3 in the 'preprossess.ipynb' Jupyter notebook. Further processing was to concatenate the years previous data and drop all NaN values, again with help of pandas (Codecells 4-5). In order to find out a good choice of past closing prices as features, I tested all values (from 1-50 days) recorded the MSE for each value of past days (code in 'find_feature.ipynb'). The following graph shows the MSE over each (n-1) past day period.



With the 'min' function from Python the past days value was found at 4 days(in Codecell 5).

The remaining features were calculated with help of the 'Ta-Lib'. Namely the SMA5, SMA8, SMA13, EMA12, EMA26 were calculated (preprocess.ipynb, Codecells 14-15) and added to the Dataframe. After preprocessing the Dataframe was saved as "data_complete_1d.csv". It contains all data points and features for the model.



Date	open	high	low	close	close-1	close-2	close-3	close-4	SMA5	SMA8	SMA13	EMA12	EMA26
2009-06-08	1.39742	1.40022	1.38050	1.39293	1.39744	1.39671	1.42886	1.41799	1.410304	1.405627	1.401994	1.399127	1.377263
2009-06-10	1.40614	1.41437	1.39151	1.39879	1.39293	1.39744	1.39671	1.42886	1.406786	1.406851	1.403118	1.398173	1.378424
2009-06-11	1.39880	1.41775	1.39430	1.41245	1.39879	1.39293	1.39744	1.39671	1.402946	1.407360	1.403035	1.398268	1.379932
2009-06-12	1.41232	1.41271	1.39354	1.40151	1.41245	1.39879	1.39293	1.39744	1.399664	1.406961	1.403795	1.400450	1.382341
2009-06-14	1.39890	1.40016	1.39664	1.39770	1.40151	1.41245	1.39879	1.39293	1.400624	1.405835	1.403840	1.400613	1.383761

Implementation

The metrics used are the Score function, the Mean Absolute Error (MAE), and the Mean Squared Error (MSE). The Score function is the built in function in the SVR algorithm and represents the the residual sum of squares(RSS), the MAE and MSE are part of scikits preprocessing library. For me the overall metric was less important than the accuracy given for a specific difference in price data. It was important to accurately predict a higher price move rather than minor moves. The spread and therefore the cost of a position is about 1-3 points depending on time and currency pair. Therefore at least 8 points or more should be relatively good predicted.

After preprocessing the data, the feautures were copied from the dataset and divided in a training set and test set. Where 500 data points were used for the test set and the rest 1737 data points for training (roughly 80%). The initial algorithm was a Linear Support Vector regression with its standard values from scikit. The result was disappointing, since it was lower than the one with only the past close data.

In later steps, I considered Gridsearch to refine and improve the performance of the model.

The model implementation was straight forward, given the good explanation in the lectures and the good data (no outliers etc.) as explained in 'I'.

Refinement

As previously mentioned I used the GridearchCV method from scikit model_selection to refine and tune the model. The Gridsearch algorithm had to chose between the following parameters:

- Kernel - 'sigmoid', 'rbf', 'poly', 'linear'
- C - 0.1, 1, 10, 100, 10000
- degree - 2,3,4,5,6
- epsilon - 0, 0.1, 0.01, 0.001

The algorithm gave the best parameters as follows:

- Kernel - linear
- C - 100
- degree - 2
- epsilon - 0.001

After the general direction was clear, I decided another round of GridearchCV with finer parameters. Since a linear kernel was the best choice, the degree has no effect. I tried the following parameters as well:

- C - 50, 100, 200, 500, 1000, 2000, 5000
- epsilon - 0.01, 0.001, 0.0005, 0.0001, 0.00005, 0.00001

The final parameters and the related scores based on the test set are reported in the following table.

Kernel	Parameters	Score (RSS)	MAE	MSE
LinearSVR	default	0.887	7.46*10^-3	8.17 * 10^-5

SVR	kernel = 'linear', C = 100, epsilon=0.001	0.951	4.47*10^-3	3.57*10^-5
SVR	kernel = 'linear', C = 50, epsilon=0.0008	0.950	4.47*10^-3	3.59*10^-5

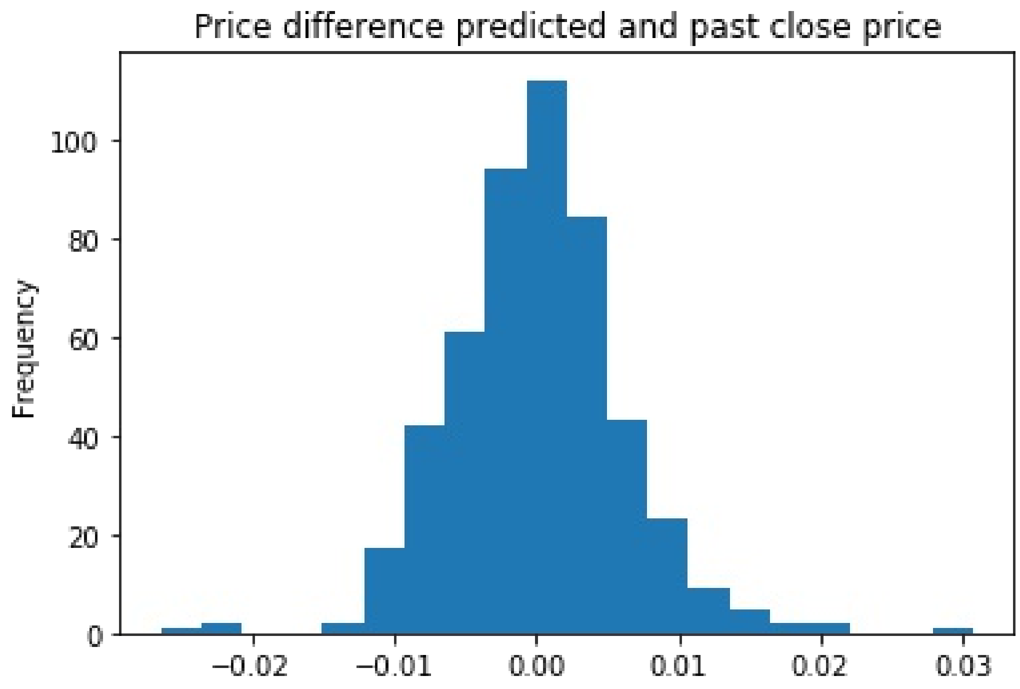
It's interesting that the score with the parameters from the second Gridsearch round are slightly worse on the test set. This could mean that some overfitting occured. Therefore the final model parameters from second row were used.

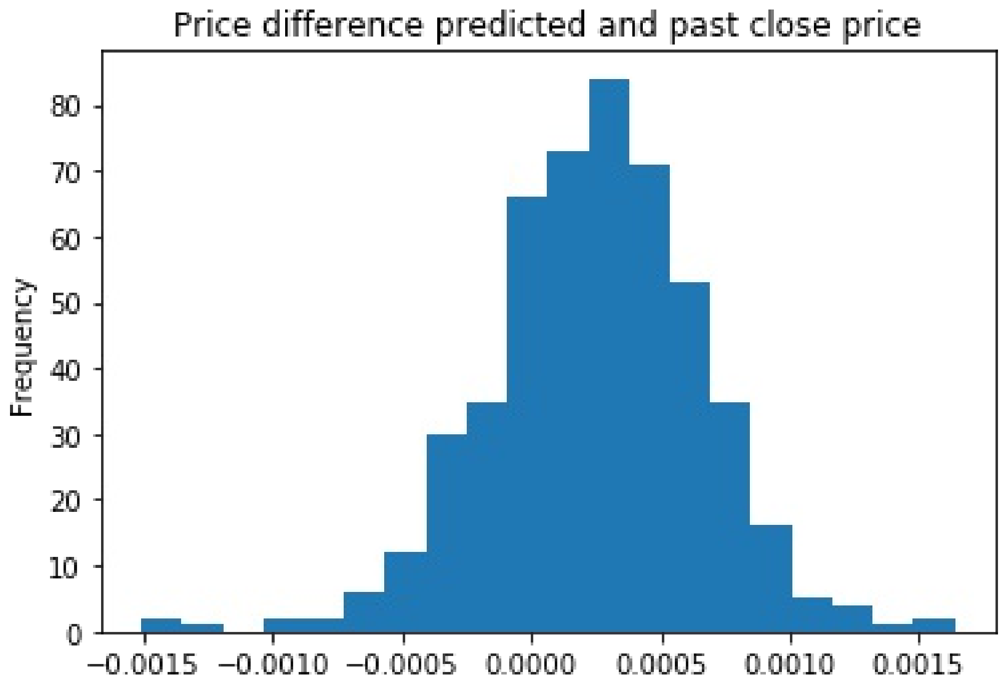
IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

The model was trained on a training dataset and tested on a testing dataset as reported above. As mentioned before, the overall score is of less importance. It is more important to give the right prediction for higher (more potential profitable) price movements. For example the condition is to consider only predictions more than 10 pips (at 1/10000 EUR) up and get that score or the accuracy for that event. The histogram of the differences of the predictions and the past days close price shows almost a zero-mean distribution. So, most of the predictions wouldn't make any profit given a high spread. The real differences are little bit skewed to the right and also shows some higher/lower values compared to the predictions.





To consider how accurate the predictions were, I compared the predictions for more than or equal 8 points higher and more or equal 8 points lower than the previous days close. For evaluation purposes, if the predicted price is bigger or equal the true price it would be counted as right and wrong if not and vice versa for the other side. For potential profit considerations, a simple strategy would be to simply buy at the prediction and hold for one day... Another safer strategy would be, to put a stop-loss in the other direction and for this simple idea hold it for one day as well. The maximum loss would be equal to the stop which was 10 points in the opposite direction. Of course that assumes that the volatility on a specific day would'nt take the stop loss first before it swings up. Also, the spread between Ask and Bid price is not included. The results are in the following table:

Condition	Accuracy	Simple profit	Stop-Loss
More than 8 up	0.42	0.6 points	533 points
More than 8 down	0.8	82 points	194 points

Justification

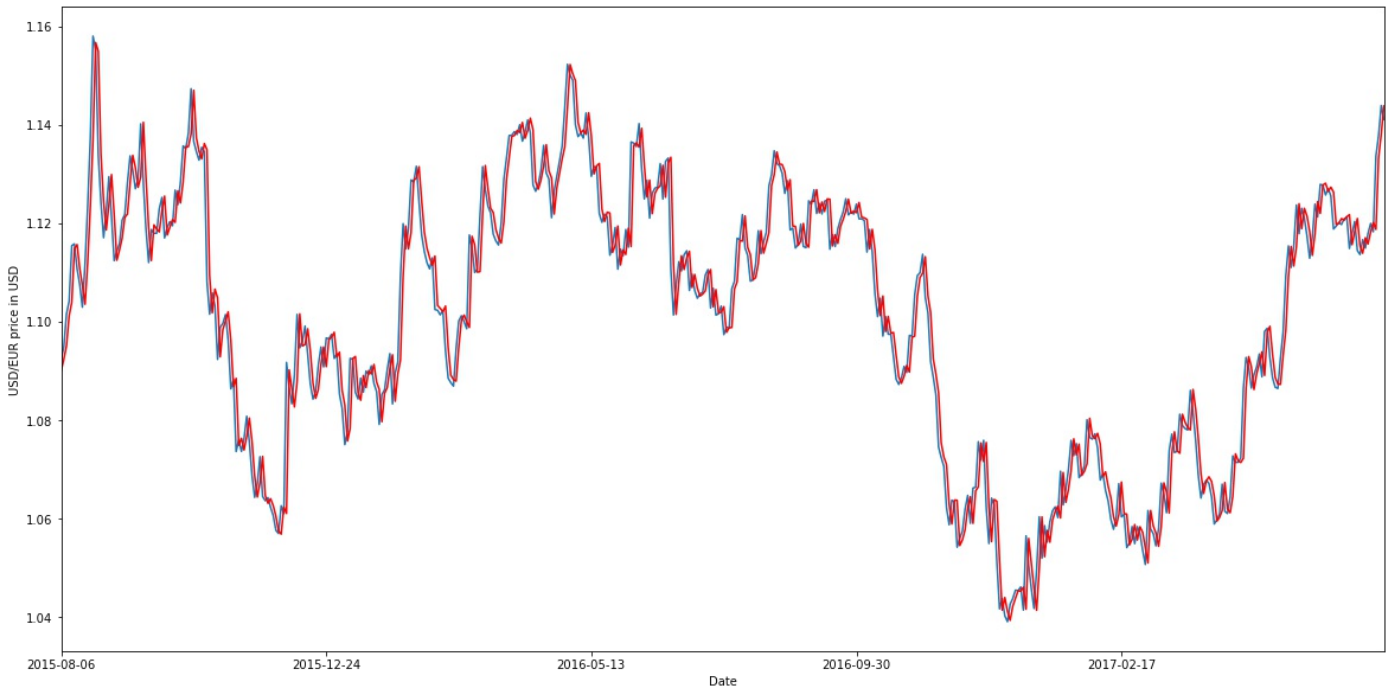
Compared to the benchmark model, the SVR model performs a little better but in general the predicted time series and the performance look almost the same to me. The big difference though, the ARIMA model predicts only one day ahead of the whole data but it's easier to realize. Taken the correlation of the predictions of the benchmark, the SVR and the true values it's a very tight result. Just compare the two graphs, it's hard to say which one has a better performance. Both MSR are very low, and the correlation factor between the prediction and the true values for the ARIMA model is about 0.9751 and for the SVR model it's about 0.9753. Where as the MSE for the ARIMA model is about 3.59*10^-5 and for the SVR model the MSE is 3.57*10^-5. What is the difference though, that the ARIMA model updates and retrains itself before making another prediction for the next days close. Therefore the usage of more sophisticated algorithms is justified. Moreover the predicting performance and training performance in terms of time is better with the SVR.

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

Taken the graph of both, the predicted next days close price and the true one, it's very impressive how good a fit is. Although the predictions have a low error and a very high correlation it seems to be lagging for about a day or so.



Reflection

The work flow of the project can be summarized in the following steps:

- first of all find a project - I chose something related to time series prediction
- decide what kind of problem - regression
- get the necessary data - the EUR/USD historical tick-data
- preprocess the data -
    - resample in a specific timeframe
    - remove NaN values
    - find and create features
    - split the data in training and testing set
- train a model to get a baseline - LinearSVR
- refine or tune the model - SVR in conjunction with Gridsearch
- define a scenario - predict only specific price movements

In general the process is very smooth and was a lot of fun. Given all the preprogrammed libraries and tools the main problem was to figure out the features. It's possible to create endless features and import indicators fr that matter. The problem was that the model would easily overfit. So, the hardest part was (and still is) to choose good features.

Improvement

There has been a recent publication were SVR and ARIMA -> ARIMA-SVR model (Stock Forecasting Method Based on Wavelet Analysis and ARIMA-SVR Model) were created and show promising results. So, one angle could be to pair other well known algorithms with new ones. Or even add Neural Networks like the LRSNN to further improve performance and robustness. Another way could be to train multiple models in different timeframes and combine them. In simple terms take the prediction for the week and use it as a limit for the daily base. Or daily for hourly and so on. Further improvements as mentioned above might bring new features. Especially in that case, event based features (Media, News, Financial Services) might do well.