Capstone Project: Forex Market direction predictor

Eyal Perelmuter - December 26, 2016

I. Definition:

Project Overview:

The foreign exchange market, which is usually known as “Forex” or “FX”, is the largest financial market in the world. Compared to the $22.4 billion per day volume of the New York Stock Exchange (NYSE), the forex market has about $5.3 **trillion** per day trade volume. The currency market is over 200 times bigger (although the NYSE makes more “noise” in the news).

What is traded is money (we can think of buying a currency as buying a share in a particular country, like buying stocks of a company). The price of the currency is usually a direct reflection of the market’s opinion on the current and future health of its respective economy.

In forex trading, when you buy, say, the Japanese yen, you are basically buying a “share” in the Japanese economy. You are betting that the Japanese economy is doing well, and will even get better as time goes. Once you sell those “shares” back to the market, hopefully, you will end up with a profit.

In general, the exchange rate of a currency versus other currencies is a reflection of the condition of that country’s economy, compared to other countries’ economies and what is traded are thus, currency pairs.

The major currencies traded are: USD (US dollar), EUR(Euro), JPY(Japanese Yen), GBP(British pound), CHF(Swiss franc), CAD(Canadian dollar), AUD(Australian dollar), and NZD(New-Zeland dollar).

The currency pairs from the above list involving the USD are called Majors (USDEUR, USDJP, etc..) and currency pairs from the above list not including the USD are called crosses (EURGBP, NZDJPY, etc...).

In this project we will try and use techniques from supervised machine learning to predict the direction of the exchange rate for two currency pairs (up or down) for various predictions windows (1 day, 3 days, 7 days and 14 days) to use as support

in discretionary trading decisions.

Problem Statement:

I am a discretionary retail Forex trader. What that means is that I trade the forex market as an individual trader with a small capital relative to the market major players like banks and funds (this is the retail part) and I do it through the interpretation of price charts trying to decipher the price action trail and thus speculate on the future exchange rate of a specific currency pair (like EUR/USD, for example).

Here is an illustrative example of such a chart [1]:

Machine learning techniques can greatly aid in this “manual” process which is highly subjective and therefore not as consistent as quantitative approaches.

The objective is to build a directional currency rate predictor given the spot price for the daily time frame or the OHLC set of prices (Open, High, Low and Close) along with any indicators calculated from these set of prices over a set of dates for the daily time frame, and estimate the closing price direction (up or down) for given intervals into the future.

The intervals are: 1, 3, 7 and 14. The logic behind the choice of these numbers is the typical holding period of trades (this statement holds for me, since it is highly

variable from trader to trader based on personal preferences of exposure to the market) in the different time frames, which for me, varies from one day to maximum of two weeks, based on my trading history.

The idea is to use mainly supervised learning methods to estimate the direction of the market after these periods (up, down) and couple it with my own discretionary interpretation of the price action, thus providing some kind of positive or negative signal about my discretionary trade ideas.

Once a model is chosen, the features (represent market information calculated from the price) are plugged in and the direction is given as output, for the different prediction periods, and this should aid me in calibrating and analyzing my trading views. For example, I am biased long on the EURUSD but the model predicts is going down in 7 days and thus I need to either analyze a short opportunity or restrict my holding period for less than 7 days or maybe ignore the model (also a valid option if I am confident in my price action reading skills).

The currency pair I have chosen from the universe of currency pairs is one that is traded the most by me: EURGBP.

There are many more that I trade, but for starters and for the purpose of this project, these give me a good starting point to see how well these techniques work in the contest presented here.

Metrics:

The main metric I will use is accuracy that takes into account true positives and true negatives proportional to the entire data set. The main reason to use this metric for quantifying each model (not to determine whether it is good or bad, for that we need the benchmark that will be discussed later) is that a valid prediction of either up or down has equal weight since in trading we can profit from either direction (via long or short positions):

$$\frac{\#\_True\_UP+\#\_True\_DOWN}{TotalDataSize}$$

This metric is going to be used in quantifying each specific model. In order to judge whether a given a model is good or bad this accuracy score will be compared against other models accuracy score and against the benchmark.

II. Analysis:

Data Exploration:

The price chart shown above is called a candlestick chart and every candle represents one unit of the time frame of the chart. So, for a daily chart each candle represents one day. The way the candles are created is by taking the open price, close price, high price (the highest during the time frame we are looking at) and the low and form this candle.

I had two major options to get hold of these data: use the data sets from quandl (www.quandl.com), that contain the spot exchange rate but are restrictive when it comes to Forex, or using a licensed software (I own a legal license) called Forex Tester (www.forextester.com/ft3), which is designed for Forex trading simulations and testing of strategies and therefore give access to many currency pairs with data in the 1-min resolution (OHLC for 1-min).

The downloaded data are saved in the form of .csv files to be later processed. The data is very raw and can't be used as is. In the preprocessing part we will discuss the exact steps that needs to be done in order for us to be able to train and test different algorithms. Here I will bring a snapshot of the first couple of rows of the raw data:

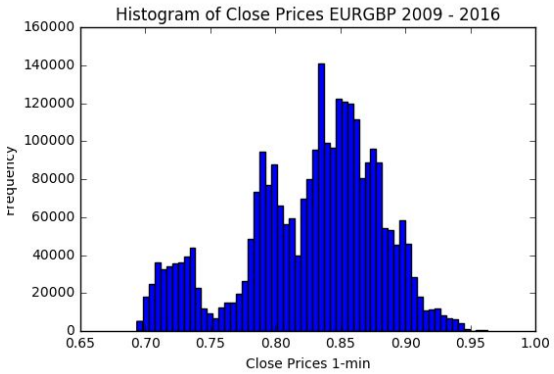
<TICKER>	<DTYYYY	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>	<VOL>
EURGBP	20090102	2100	0.9551	0.9551	0.955	0.955	4
EURGBP	20090104	2300	0.9545	0.9546	0.9545	0.9546	4
EURGBP	20090104	2301	0.9545	0.9545	0.9542	0.9543	4
EURGBP	20090104	2302	0.9544	0.9547	0.9544	0.9546	4

What we can see here is the 'TICKER' which is the symbol name (EURGBP in this case), the date where this data point was recorded (we are dealing with time series data here), the time of the day of this data, and an estimate for the volume traded. Now, in the Forex market, since it is not a centralized exchange like in the stock market, there is no reliable figure for volume and quoted values for volume have little to no meaning because they represent the volume passing through a particular broker. Also, <TIME> is of no relevance for me since I am using a daily aggregate for analysis.

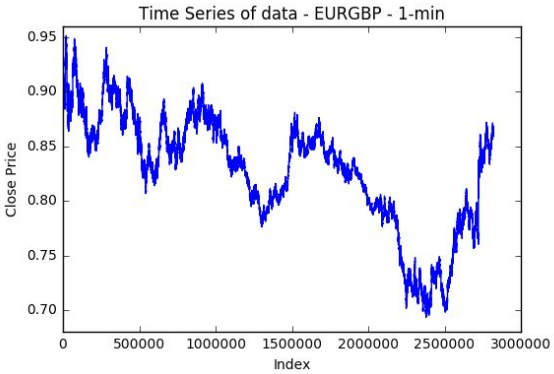
The most important parts (price wise) are the four columns that capture the OHLC exchange rate prices on a minute basis for this currency pair. So, for example, the first row above indicates that at 20090102 at the time interval [2100,2101], the open exchange rate was 0.9551 (that is one Euro is exchanged for 0.9551 GBP), the close rate was the same and so on (during this particular minute).
We will take advantage of Forex Tester for data and will attach all the csv files (Forex Tester as part of the license allows to export data to .csv for later analysis).

Exploratory Visualization:

Here we will take a look at the raw data although at its current form it is not extremely useful, but we can get an idea with respect to the spread and non-normality of the closing price data on a 1-min time frame:



We have several peaks due to trending data and thus prices that were “visited” less. If we take a look at the time series itself (with the x-axis being an integer index for simplicity) we can see this phenomena:



This is a “good” data set in the sense that various market regimes are captured - uptrend, downtrend and even some ranging behavior. It is important to let the

different algorithms to “see” data that “belongs” to different market regimes, because if only highly up trending environment is captured in the data, the model will not have a good predictive power for a ranging market, for example.

Algorithms and Techniques:

The way I have defined the problem to be addressed, we are facing a binary supervised classification problem. From this basic observation there are several algorithms that we can deploy.

The reason classification is the better way to go (compared to regression and actually predicting the exchange rate) is that based on my experience, I have little faith that the exchange rate can be predicted with accuracy and a more practical approach for me as a trader is to have predictions of of the market direction that coupled with discretionary analysis can give better results.

In light of this, the algorithms I will use and compare are Decision Trees, SVM, Neural Networks, knn and Adaboost.

I chose a mirage of algorithms due to the highly non-linear complex and dynamic nature of the market. These algorithms are good choices for supervised binary classification problem as the one we face here.

Decision trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features, which in our case are price technical indicators and thus mimic, in a sense, what a discretionary trader is doing and can have many more rules than what a trader can run in his/her head.

Support vector machines are great for binary classification problem and should find the minimal set of vectors that actually matter for the decision boundary which is very appealing in our context, although we will have to play with the kernel functions since my domain knowledge does not help me in picking up a kernel function over another.

Neural Networks are great for non-linear function approximation and can be used for classification. I suspect I will explore them the least (in this work) due to their “black-box” nature and I will use the default settings as found in sklearn for

MLPClassifier (Multi-layer Perceptron classifier). For an instance based algorithm (which is used in the Udacity course “Machine learning for Trading” for regression), we will use kNN which makes sense to me since the voting of neighboring days with respect to the outcome should have an impact on the current day prediction. The boosting algorithm is used since I believe an ensemble algorithm can be quite useful in the sense that many weak learners together can give a good prediction in such a complex environment, and I chose AdaBoost since in the core it concentrates on areas where the learning and results were poor when it iterates. As it can be seen, there are many good algorithms to try and the trick is to pre-process the data in such a way that running them all won’t require data wrangling for each one individually and since all the classification algorithms in sklearn have the same API, it can be done.

For each one, we will fine tune some of the parameters:

- Decision trees - criterion (‘gini’ or ‘entropy’), max_depth - too complex tree will over-fit and too shallow will have low bias and won’t capture the complexity of the data, min_samples_split - we would like to search for a minimum number of days that will give a good predictive power, min_samples_leaf - the same principle in the “other” direction, we need a minimum number of days to make sure here the tree ends and we can make a decision.
- SVM - since the domain knowledge is mainly represented through the kernel function and I don’t have any special preference, we will use ‘rbf’, ‘linear’ and ‘sigmoid’. Also, the regularization parameter, ‘C’, will be fine tuned and ‘gamma’ for ‘rbf’ and ‘sigmoid’
- NN - the main thing to be checked here is the activation function and I will not play too much with the default network architecture as given in MLPClassifier of a hidden layer with 100 nodes [2].
- KNN - the main parameter to be tuned is K, which in our problem domain represents the number of days we are letting to influence on our decision in the training set which should not be too many.
- Adaboost - since I afraid any one learner will not be able to capture the complexity of the market, a good idea is to use a boosting algorithm that

concentrated on the “hard” parts using many decision trees. Also, we can use the best fine tuned parameters found for each decision tree classifier and use them here, and fine tune the number of estimators or in other words the nuber of decision trees in the ensemble.

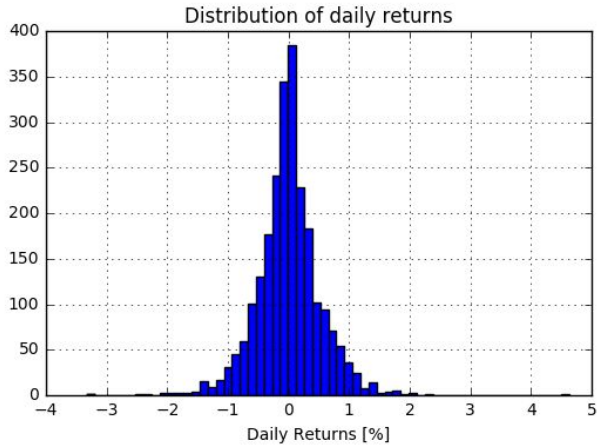
We will split the data into training set from the start of the data up until the end of 2015 and for the testing data we will use 2016. The fine tuning will be done either through a GridSearch scheme and sometime through manual looping of parameter values, for simplicity.

Benchmark

In classical economics the markets are assumed to be efficient (the efficient market hypothesis) [3]. In practice that means that a trader does not truly have an edge and the market behavior is no different from a random walk.

That made me think that a good benchmark model would be to draw values from a normal distribution with mean equal to the daily return and standard deviation equals to the standard deviations of the daily returns, and to “walk randomly” for the time window we are trying to predict (for example, 14 days).

The use of a normal distribution is plausible by eye-balling the following histogram of daily returns:



The average return is highly concentrated at 0 which give support the random walk for benchmark use.

The above procedure will land us at a certain exchange rate which we will

associate with an appropriate market direction prediction (Up or Down relative to where we have started). Now, this process is only one realization of a random process and therefore will be simulated N times and a good sample size to use is N = 1000.

This seems to me a good benchmark model and its performance should be compared to the algorithms used.

Although we will discuss at length at the data preprocessing section all the stages we need to perform in order to transform our raw data on a 1-min basis to the daily time frame and calculate several technical indicators that we will use as features, I will try to elaborate a bit now how we calculate the benchmark given that we have already pre-processed our data and calculated our features.

The exact code can be found in the Jupyter notebook that accompany this report, but the important parts regarding the benchmark are as follows:

- We draw random values from a normal distribution:
  `randomValues = npr.normal(dailyChangeMean, dailyChangeStd, period)`
- We record the current daily closing price and walk forward randomly using these random values, and simulate it for sampleSizeSimulation times.

```
for i in range(sampleSizeSimulation):
    randomValues = npr.normal(dailyChangeMean, dailyChangeStd, period)
    tmpPrice = startClosePrice
    for i in range(period):
        tmpPrice = tmpPrice + randomValues[i] * tmpPrice
    simulatedPrices.append(tmpPrice)
```

- The benchmark value is the mean of all the final prices which we have reached during the different iterations:
  `benchmarkPriceResult = pd.Series(simulatedPrices).mean()`
- At this point we convert the result, using another custom function, to a binary result of UP or DOWN.

Using this algorithm, we can calculate the benchmark value for every data point in the test set to calculate the benchmark accuracy score on the test set, and this accuracy is the number our different learning models should outperform.

So, as it can be understood from the above, our benchmark is actually a model that believe in absolute randomness and no edge for traders in the market and its predictions will be compared with the different machine learning models.

III. Methodology

Data Preprocessing

The first part of preprocessing the data includes transforming the 1-min time series to a daily time series. In order to achieve that we calculate the daily OHLC. The daily high and low are the maximum and minimum prices of the the 1-min data grouped by day, and the daily open and close prices are the average of the first and last 5 minutes of each day.

After that we calculate the output variables for each day, which is the binary direction of the market (Up or Down) for each prediction interval (1 day, 3 days, 7 days, 14 days).

Now we have a pandas dataframe with daily prices and all our output variables, but we still lack predictive features to feed our model with. These features represent a subset of technical indicators which are widely used in trading and presumed to have predictive power. These indicators are trying to catch market characteristics like volatility and momentum and their values should combine in such a way as to give prediction about the market direction.

The indicators we are going to use are:

- Current daily return which is the change in price relative to the previous day
- SMA(20) / ClosePrice - SMA stands for Simple Moving Average and 20 is the look-back period for calculating this average. [4]
- ATR(20) - ATR stands for Average True Range and measures volatility. [5]
- Bollinger Bands values (upper band value and lower band value) - also measures volatility and looks how price has “drifted away” from the SMA. It is a very popular indicator. [6]
- MACD (Moving Average Convergence Divergence) - It is a popular momentum indicator that computes the difference between two moving averages of different periods. [7]
- RSI (Relative Strength Index). This indicator is a momentum oscillator that

relates to zones of oversold and overbought of a financial instrument. [8]

Here we 7 features which I consider to be quite good since over crowd the models with indicators will result in a model which won't be readable. Traders do not like (and I among them) totally black boxes and therefore I prefer to use small subset of indicators that many traders use and feed them into our models.

Each of the above features as a dedicated function that calculates it and creating the right column in our pandas dataframe. Here is the list of these functions (can be found in the Jupyter notebook):

- calculateSMA(data, lookbackPeriod = 20)
- averageTRvalues(TR, lookbackPeriod = 20), calculateATR(data, lookbackPeriod = 20)
- calculateBB(data,lookbackPeriod = 20, distance = 2)
- calculateMACD(data, fastPeriod = 12, slowPeriod = 26, signalPeriod = 9)
- calculateRSI(data, averagePeriod = 14)

It is also important to note that all indicators are calculated from mathematical formulas that are using only the OHLC set of prices.

After all these features are calculated we drop all the rows with NaN values and all the non-feature columns (volume, OHLC) and remain with a "clean" data set with daily based features and four output columns for the different prediction intervals, ready for use by our different learning models.

Implementation and Refinement

The first thing was to split the data into training and testing sets. Now, the training set consists of the last 6 years and the testing set consisted of 2016:

trainData = dfDailyEURGBPClean.loc[:intToDate(20151231)]

testData = dfDailyEURGBPClean.loc[intToDate(20160101):]

After the split we had 2137 days in the training data (on weekends the Forex market is closed) and 220 days on the testing set.

Next, we calculated the benchmark value for our test period (2016) and calculated the accuracy score of the benchmark model for the different periods.

For the machine learning algorithms we used the library sklearn and at first implemented the classifiers with parameters out of the box.

We started with Decision Trees and moved to SVM, NN, kNN and a boost algorithm (AdaBoost) with many decision trees.
Here is an example code for the first algorithm tried, which is Decision tree:

```
clf = DecisionTreeClassifier(criterion = "entropy", random_state = 23)
X = trainData[features]
Y = trainData[['return_direction_only_14_days']] #we run it for the two weeks direction prediction
clf.fit(X,Y)
clf.score(X_test, Y_test)
```

The out of the box models did not give strong results compared to the data frame so I proceeded for fine tuning parameters (as described in Algorithms and Techniques section of this document).
In order to do that I divided the training set into folds to be able to run a cross-validation scheme and use GridSearch to perform an exhaustive search over some parameters.
For that purpose we can't use the usual KFold techniques since we are dealing with a financial time series and ordering is important because we do not want to have look-ahead bias. Therefore, I used TimeSeriesSplit with 8 splits and for each period of prediction performed a grid search:

```
cvSplit = TimeSeriesSplit(n_splits = 8)
```

This split gives a set of indexes to perform the cross-validation that respect the time ordering.
I kept the best model (judged according with its accuracy) along with its parameters, for every period and this was the best the particular algorithm tried was able to do:

```
clf = GridSearchCV(estimator = tree, param_grid = p_grid, cv = cvSplit, scoring = 'accuracy')
```

After that, each best model was tested on the unseen test data (the year 2016) and this result was compared to the benchmark model result.
Other important refinement details are concerned with the boost model and tuning procedure.

For the AdaBoost model I have used the best decision tree parameter configuration as found in the fine tuning part and for most of the model I created a function to aid in the tuning procedure:
fineTuneParameters(bestScores, bestParams, bestScoreTestSet, model, pGrid)
This function takes as its first 3 arguments empty dictionaries to store the results of the best score of the best model (as found in the grid search), the parameters of that best models and the score of this model on the unseen test data.
The last two arguments are the basic learner and the particular parameter grid of the algorithm we are currently want to run.
In the next section we will review the results of all the different models along with their parameters, comparison to each other and to the benchmark results.

IV. Results

Model Evaluation and Validation

Here we will present the results obtained during the run of the different models. We will summarize our results in tables capturing the best performance of each algorithm for the different prediction intervals. Each cell in the following table is the accuracy score of the best model on the unseen test data after it was trained and optimized through cross-validation (in most cases) on a set of parameters.

	1 - day	3 - days	7 - days	14 - days
Decision Trees	0.4863	0.4454	0.5181	0.4681
SVM (linear)	0.5181	0.5272	0.3863	0.3590
SVM (rbf)	0.500	0.4954	0.4454	0.3590
SVM (sigmoid)	0.4863	0.5318	0.5090	0.3590
Neural Networks (logistic)	0.5181	0.4454	0.5045	0.4045
Neural Networks (tanh)	0.4727	0.4590	0.4772	0.4272
Neural Networks (relu)	0.5318	0.4772	0.5045	0.4954
kNN	0.4681	0.5545	0.4681	0.4363
AdaBoost	0.5454	0.4909	0.5	0.4409

The terms linear, rbf and sigmoid for SVM's are different kernel functions (I included the linear since I thought maybe the problem is linear in seven dimensions :-)) [9]

The terms for the neural network are different activation function [10].

Justification:

Here we will examine the results obtained for the benchmark model:

1 - day benchmark result	0.5363
3 - day benchmark result	0.5136
7 - day benchmark result	0.4363
14 - day benchmark result	0.4272

As it can be seen, the random walk benchmark results are not that bad... The market is very efficient and that is logical but there are models that outperform this benchmark result.

First thing to notice is that we have different models for different prediction periods that outscore the benchmark. That is to be expected since different players enter the market on the different time scales. Daily traders hold positions during a day and swing traders for a couple of days. Also, more fundamental factors play during a week or two weeks of market activity. Due to that I suspect each time period prediction has its own uniqueness and thus can correspond to a different model.

For 1-day interval AdaBoost has the best performance.

For 3-day interval kNN has the best performance.

For 7-day interval Decision Tree has the best performance.

For 14-day interval NN has the best performance.

As mentioned previously, for each model the testing was done on unseen data and the fine tuning of parameters was done on 8 folds of the training set for each interval of interest.

The parameters of the best model for each interval are as follows:

1 - day	3 - day	7 - day	14 - day
AdaBoost: n_estimators = 40, The DT of each weak learner are that of the 1-Day: min_samples_split = 6, criterion = 'entropy', max_depth = 6, min_samples_leaf = 9	kNN: 'n_neighbors' = 20 'weights'= 'uniform'	DT: 'min_samples_split' = 6 Criterion = 'entropy' 'max_depth'= 6 'min_samples_leaf'= 7	NN: Default hidden layer with 100 nodes, activation = 'relu' Solver = "lbfgs"

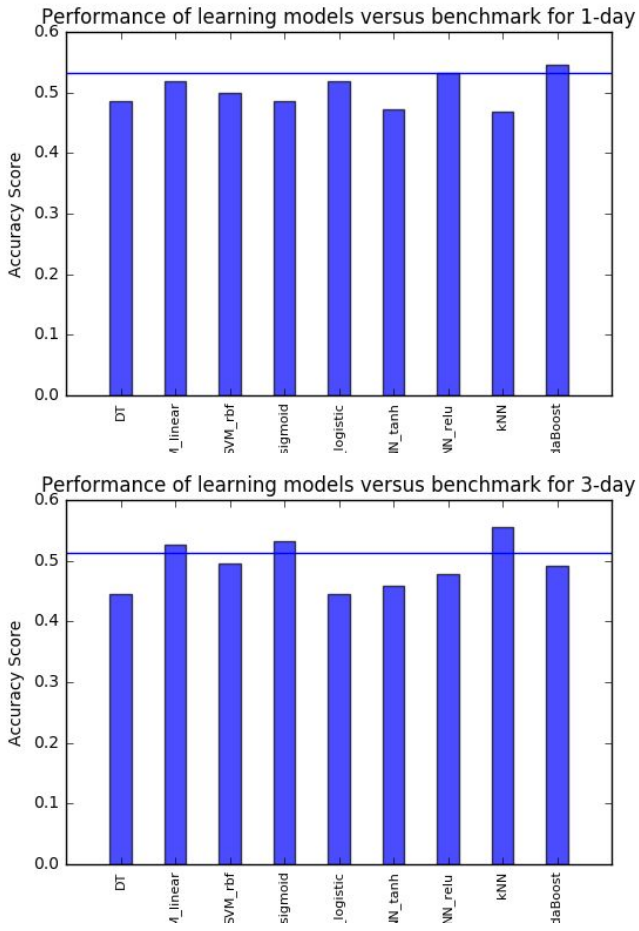
Now, these final solutions can be further improved, I presume. This is something we will discuss at the next section, but the important takeaway from this part is that for each prediction interval we have found a model that out perform the benchmark, at least a little which, as a trader, I interpret as a good result and will use these models to help me in my discretionary trading views on the market.

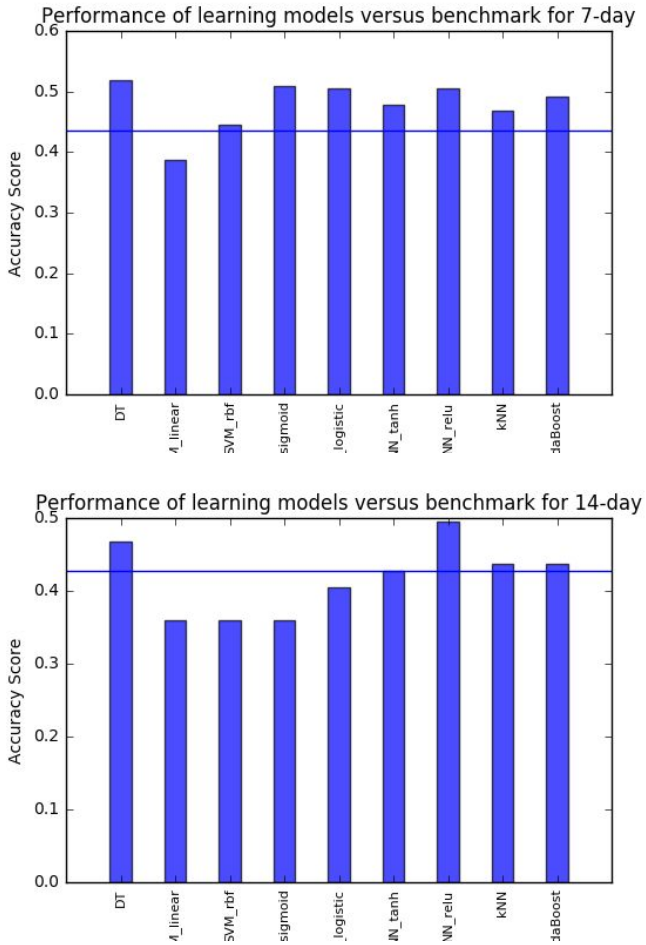
V. Conclusion

Reflection

This work had the objective to try and use machine learning algorithms and techniques to predict the future direction of the Forex market with respect to one chosen currency pair, EURGBP. These predictions were confined to four periods of 1,3,7 and 14 days which are based on my personal trading preferences for

holding periods. The data was obtained from an external data provider (under a legal license) and contained data of exchange rate on the 1-min time frame.
The data was transformed to a daily time frame basis and seven technical indicators were chosen and calculated to be used as predictive features.
Then, a mirage of learning algorithms for supervised binary classification were trained and tested using cross-validation and parameter fine tuning along with a final unseen testing data.
The results were compared to a benchmark result that predicted the market direction based on a random walk path Monte-carlo simulation.
Here is a visualization of the results:





The most important take away is that for 3-Day, 7-Day and even 14-Day predictions we have several models that can out perform the benchmark while for 1 day prediction we are out performing the benchmark very little and only in one model. The reason that is, according with my understanding, is that due to the random nature of the market and the noise on a day-to-day basis it is more difficult to predict the correct direction versus longer horizons since in the later case, fundamental factors dominate over the noise a little bit and not every random news, rumors or other factors move the market as much compared with the daily case.

All in all, I am happy with the fact that the benchmark was out performed, but I did expect at the beginning of this project to have a performance of about 60% accuracy, which I was not able to achieve but I got to a very good start.

Improvement

In light of the results there are several improvements I think need to be made in future work to make the models more reliable. First of all, more predictive features should be looked up and segmented by prediction interval, so we can find ourselves with different sets of features for different prediction intervals with some degree of overlap between the features.

Another improvement would be to dig in a little bit further to other ensemble algorithms with other weak learners or combination of different learners that had a good performance and maybe their aggregate would do even better. Also, different architectures of neural networks should be tested out, something I did not explore here and can have a high impact on the final result.

Last but no least, is to change approach and use reinforcement learning that instead of a prediction will learn a trading strategy and we can then simulate this strategy on past unseen data (backtest the strategy). This is a difficult task but would be a very beneficial thing to do. Such a model will not be used as an aid to a discretionary trader but rather will serve as an autonomous trading robot. See [11] for a discussion on reinforcement learning for trading.

References

[1] - http://www.google.com.co/search?q=forex+candlestick+charts+example&biw=1366&bih=662&source=lnms&tbm=isch&sa=X&ved=0ahUKEwixtI7TI_bPAhWC6SYKHQavD1QQ_AUIBigB#imgrc=SAaFtZoPEGhG1M%3A

[2] - http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[3] - http://www.investopedia.com/terms/e/efficientmarkethypothesis.asp

[4] - http://www.investopedia.com/terms/s/sma.asp

[5] - http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:average_true_range_atr

[6] - https://en.wikipedia.org/wiki/Bollinger_Bands

[7] - http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_average_convergence_divergence_macd

[8] - http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_index_rsi

[9] - http://scikit-learn.org/stable/modules/svm.html - section 1.4.6

[10] - http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier

[11] - https://classroom.udacity.com/courses/ud501/lessons/4930572236/concepts/49546703660923#