Capstone Project

Santiago Bacaro Bravo

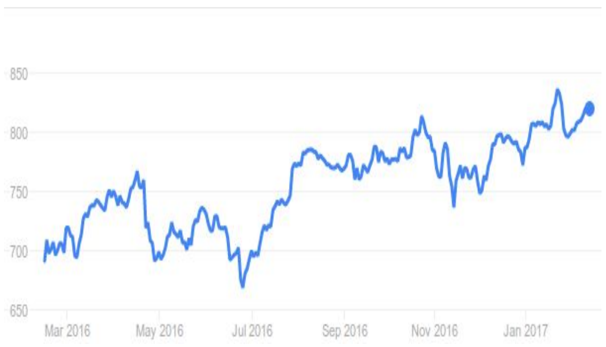
Machine Learning Engineer Nanodegree

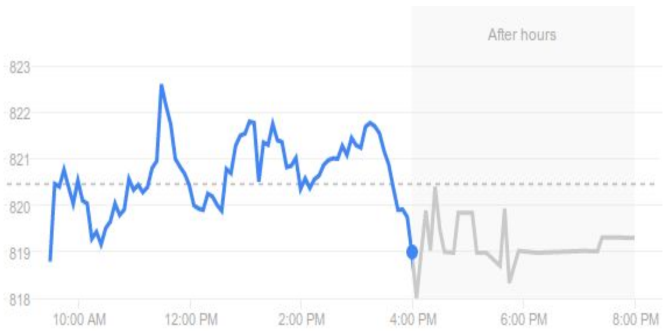
August 7th, 2017

Using Machine Learning To Predict Stock Prices

Definition

Mark Twain, the american writer, said: "*OCTOBER: This is one of the peculiarly dangerous months to speculate in stocks in. The other are July, January, September, April, November, May, March, June, December, August, and February.*"[1]. This quote illustrates the volatile nature that the stock market has or has been believed to have. To further elaborate on this point, here's a couple of images taken from Google that show the behaviour of the Alphabet stock both in one year and in one day.





Looking at the graph in the left, it could be possible to think that the data roughly adjusts to a line, but looking at the other graph, the data appears to be completely random. In the stock market world, that's what stock prices usually seem to be: random; and if that were truly the case, stock prices wouldn't be predictable. This project is an approach to predicting stock prices using machine learning.

Now, even if this data is believed to behave randomly, several approaches to predicting it have been developed, ranging from regression, to a combination of SVMs with genetic algorithms[2], to neural networks. And as seen in Udacity's Machine Learning for Trading course, systems to predict stock prices can be developed using a variety of ML techniques, from both supervised and reinforcement learning.

[1] Twain, M. The Tragedy of Pudd'nhead Wilson. New York: New American Library, 1964
[2] Choudhry, R., Garg, K. A Hybrid Machine Learning System for Stock Market Forecasting, World Academy of Science, Engineering and Technology 15, 2008

In order to talk about the kind of problem this is and the potential solutions, it's better to look first at the data that will be used. For this project, Google Finance was considered best, because it's simple to use and provides all of the basic data of interest. Sample data obtained from a downloaded CSV file for a given stock looks like this:

Date	Open	High	Low	Close	Volume
7-Aug-17	929.06	931.70	926.50	929.36	1032239
4-Aug-17	926.75	930.31	923.03	927.96	1082267
3-Aug-17	930.34	932.24	922.24	923.65	1202512
2-Aug-17	928.61	932.60	916.68	930.39	1824448
1-Aug-17	932.38	937.45	929.26	930.83	1277734

From this data, it's possible to observe that from all the mess that goes on throughout the day as was seen in the graphs above, only the opening, lowest, highest, and closing prices are returned. However, the only data of interest here, is the close price. All of the other values, except for the date, will be thrown away. Now, there's a couple of ways this problem could be addressed, and the type of the problem is determined depending on which one is chosen. It could be a regression problem if the idea was to predict a precise, continuous value that the stock will take on a given date or date range; it could be a classification problem if the idea wasn't to get an exact price for the stock, but perhaps to determine whether it will go up or down; and it could be a reinforcement learning problem, if the idea was to find a policy to decide if it's best to sell, buy or do nothing with a stock given some state. In this project, despite considering the other approaches to be interesting and potentially as good or even better, the regression approach will be taken. An attempt will be made to predict a precise, continuous value for some given stocks.

For this project, as a benchmark model, the actual stock prices were used. This was made possible by the fact that the project allowed to set some old dates to predict for, thus making it possible for us to compare the predicted values with the real ones.

As an evaluation metric, both mean absolute error (MAE) and R^2 were used. MAE is a good metric for this problem because we care about how far our predictions were from the actual values This metric is calculated as follows:

$$\frac{\sum_{i=0}^{n} |y_i - \hat{y}_i|}{n}$$

Where Yi is the true Y value, Ŷi is the predicted one and n is the number of samples.

On the other hand, R^2 is a good metric, because it portrays the relevance of our features, by giving an insight about how much of the variance in our dependent variable can be explained by our independent variable.

Hence, by applying this metric we'll get a general idea of how far is the average of our predictions from the true values.

In this project, ensemble learners were used. Some different approaches were attempted, but finally a random forest was found to be best.

Analysis

As was mentioned before, Google Finance was used to retrieve some data to work with. Yahoo FInance had initially been selected, but unfortunately, Yahoo decided to remove the service. The data that can be retrieved from Google Finance has been shown above, but from that data only the Close prices were used. In order to train our models, some features had to be calculated.

As can be seen in the table above, the data was retrieved by date, from newest to oldest, so the first thing that had to be done was to reverse it. Leaving something like this:

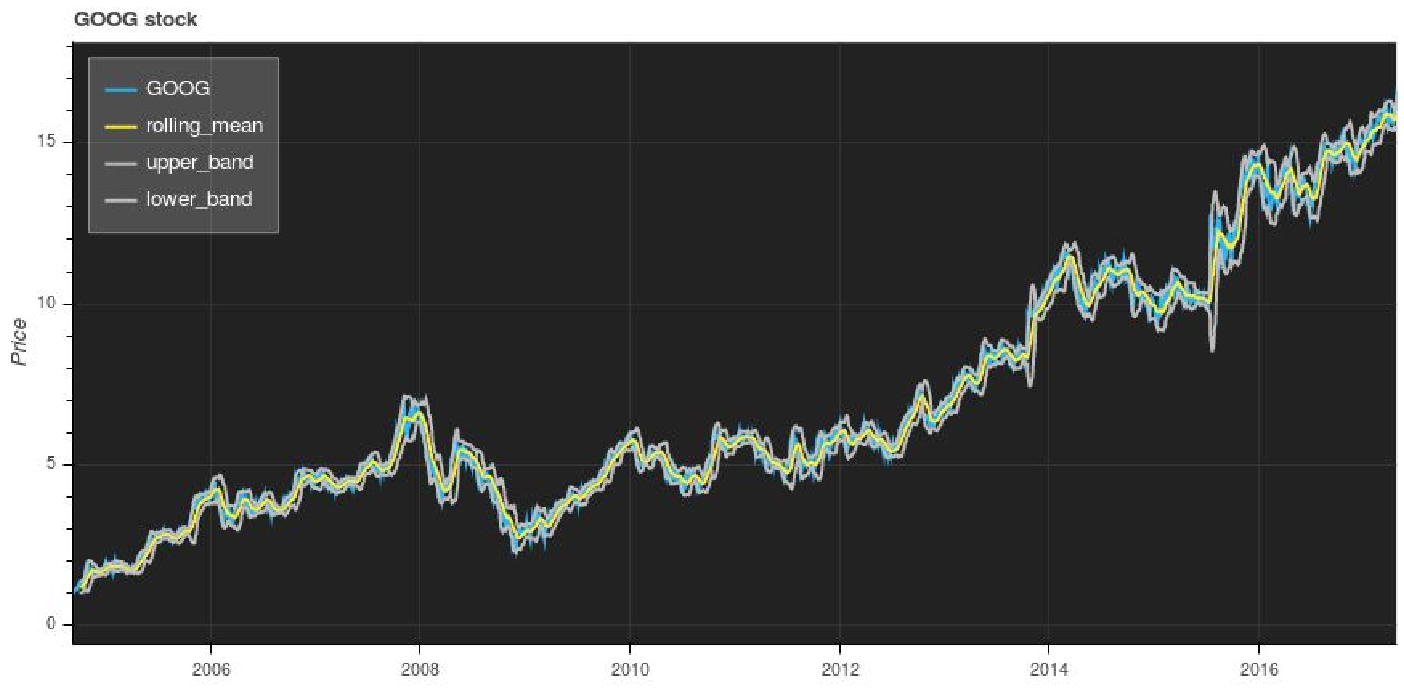
```
                 GOOG
Date
2004-09-10    52.612476
2004-09-13    53.696398
2004-09-14    55.689407
2004-09-15    55.944152
2004-09-16    56.928171
```

Notice that at this point, all the other data had already been, like Open, High and Low. The next thing that needed to be done was to normalize the data. To achieve this, the values of every day were divided by the initial value, to show the movement of the stock price relative to the initial price. This left something like this:

```
         GOOG
Date
2004-09-10    1.000000
2004-09-13    1.020602
2004-09-14    1.058483
2004-09-15    1.063325
2004-09-16    1.082028
```

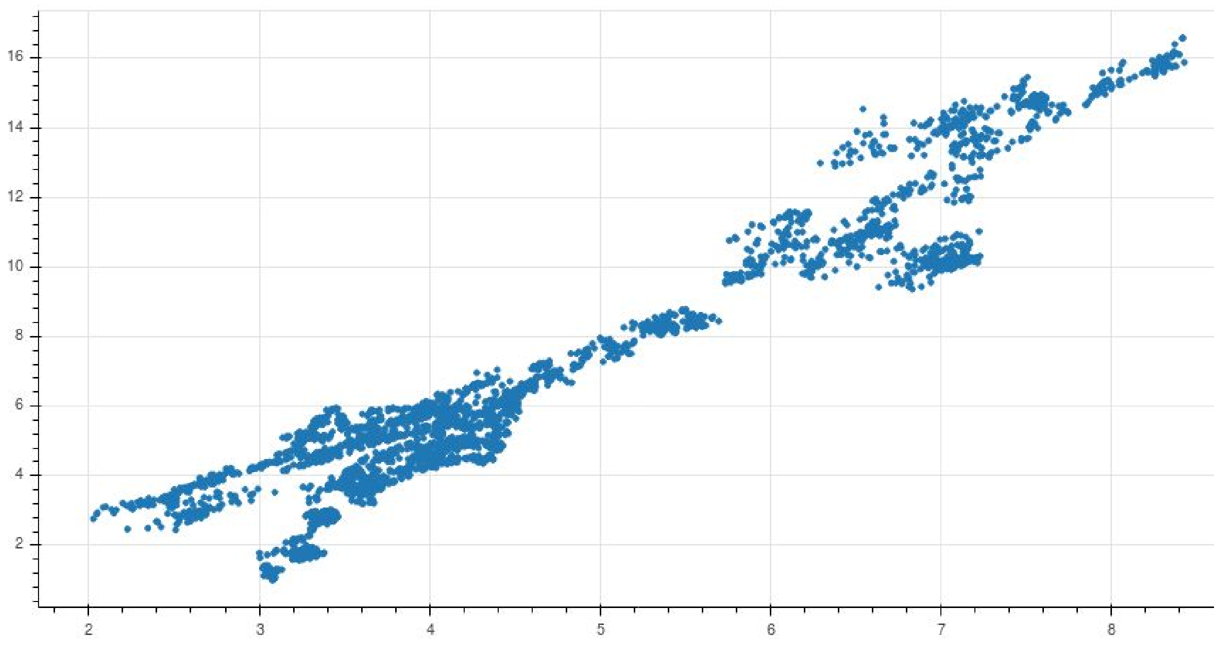
It was decided that some useful indicators would be: the Simple Moving Average (SMA) along with Bollinger's bands, the S&P 500 index price (SPY), Beta, the Finantial Stress Index and the Sharpe Ratio.

The SMA and Bollinger's bands were thought to be important since it had been shown by professor Tucker Balch in his lectures of the ML for Trading course, that when the stock price crossed either of Bollinger's bands, it could be in general a signal that the stock price was about to change drastically. Here's a chart produced for the project that illustrates the Google Stock that was being worked with along with the calculated SMA and Bollinger's bands



It's not too easy to notice in the image, but at some points, it's possible to see that when the stock price crossed one of the bands, it rapidly shifted significantly in the opposite direction.

The SPY index price was considered important because it's a financial index that should reflect changes from the market in general. It's an index that groups stocks for 500 companies from the NASDAQ and NYSE markets. Here's a scatterplot that was produced in order to show how it related to the stock that was being worked with.



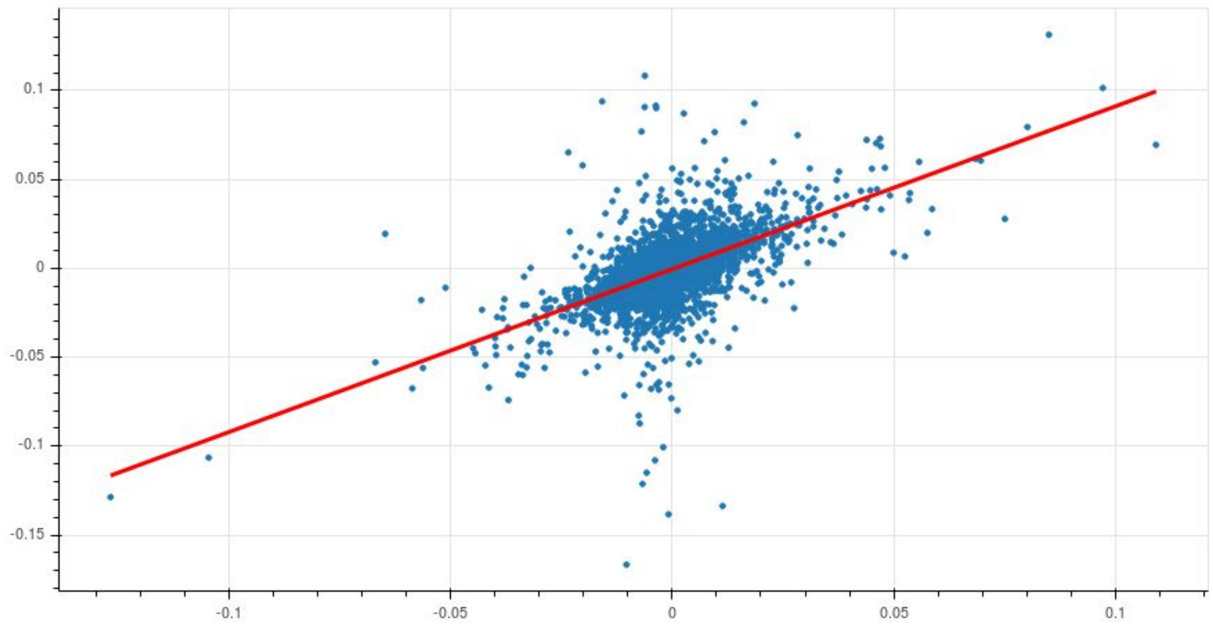
The dots in the scatterplot are really close to a diagonal line, showing that most of the times, when the price of the index increased (which should reflect an overall increase in the market), so did the test stock price in a rather proportional way.

Beta is an indicator that simply represents the slope of a line. It generally is used as the slope of the daily returns of a benchmark model of the market (such as SPY) vs the daily returns of a given stock. This was done by doing exactly this and calculating beta up to every day of the stock. At this point, the dataset looked like this:

	GOOG	**rolling_mean**	**upper_band**	**lower_band**	**SPY**	**beta**
Date						
2004-09-10	1.000000	NaN	NaN	NaN	3.076276	0.000000
2004-09-13	1.020602	NaN	NaN	NaN	3.086344	0.000000
2004-09-14	1.058483	NaN	NaN	NaN	3.092602	6.188370
2004-09-15	1.063325	NaN	NaN	NaN	3.069202	7.311342
2004-09-16	1.082028	NaN	NaN	NaN	3.078453	1.943303

The NaN values were present because to calculate the SMA, and along with it, Bollinger's bands, a window needed to be given, in this case it was a window of twenty days.

A graph representing beta at the last day of received data looked like this:



The graph is another scatterplot, this time of the daily returns of SPY vs those of the Google stock price, and on top of it, a red line representing beta was plotted. This was done by making a linear regression.

For this project, the used financial stress index was the St Louis Financial Stress Index (STLFSI). According to the Federal Reserve Economic Data (FRED), this index groups data from 18 weekly data series, each of which captures an aspect of financial stress. The data for this financial stress index is available for downloading in CSV format. However, since the data is weekly, some extra work had to be done with the dataset, until something like this was achieved:

```
             GOOG       rolling_mean  upper_band  lower_band  SPY       beta      STLFSI
Date
2004-09-10   1.000000   NaN           NaN         NaN         3.076276  0.000000  -0.585
2004-09-13   1.020602   NaN           NaN         NaN         3.086344  0.000000  -0.585
2004-09-14   1.058483   NaN           NaN         NaN         3.092602  6.188370  -0.585
2004-09-15   1.063325   NaN           NaN         NaN         3.069202  7.311342  -0.585
2004-09-16   1.082028   NaN           NaN         NaN         3.078453  1.943303  -0.585
2004-09-17   1.115447   NaN           NaN         NaN         3.091540  1.774003  -0.612
2004-09-20   1.133200   NaN           NaN         NaN         3.072961  2.028043  -0.612
2004-09-21   1.118770   NaN           NaN         NaN         3.086349  1.526005  -0.612
2004-09-22   1.123896   NaN           NaN         NaN         3.047824  0.568817  -0.612
2004-09-23   1.147062   NaN           NaN         NaN         3.031431  0.638700  -0.612
```

```
2004-09-24    1.137663       NaN    NaN    NaN    3.045365       0.514630       -0.619
2004-09-27    1.122757       NaN    NaN    NaN    3.025966       0.128543       -0.619
2004-09-28    1.204405       NaN    NaN    NaN    3.040447       0.444669       -0.619
2004-09-29    1.244470       NaN    NaN    NaN    3.055748       1.199942       -0.619
```

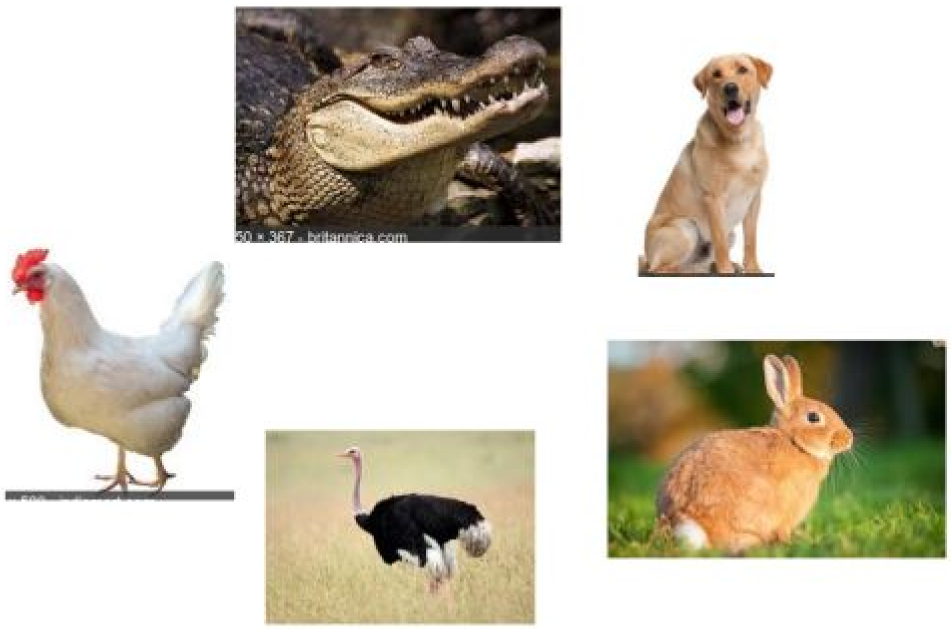
Finally, the Sharpe Ratio was calculated and added to the dataset. This ratio is an indicator of the risk adjusted return which takes into account some portfolio's return and it's risk free rate of return. After calculating it, the dataset was finally something like this:

	GOOG	**rolling_mean**	**upper_band**	**lower_band**	**SPY**	**beta**	**STLFSI**	**sharpe_ratio**
Date								
2004-09-10	1.000000	NaN	NaN	NaN	3.076276	0.000000	-0.544	0.000000
2004-09-13	1.020602	NaN	NaN	NaN	3.086344	0.000000	-0.585	0.000000
2004-09-14	1.058483	NaN	NaN	NaN	3.092602	6.188370	-0.585	-1.000000
2004-09-15	1.063325	NaN	NaN	NaN	3.069202	7.311342	-0.585	-1.273562
2004-09-16	1.082028	NaN	NaN	NaN	3.078453	1.943303	-0.585	-1.074637

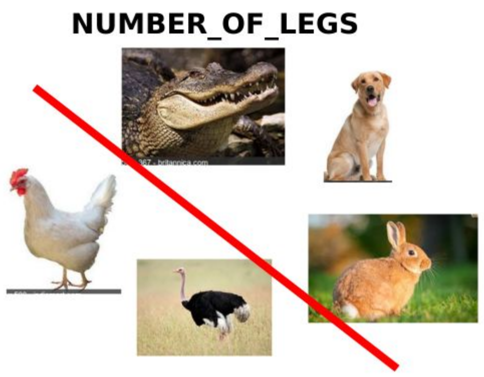
Those are all the indicators that were calculated to populate the dataset for this project.

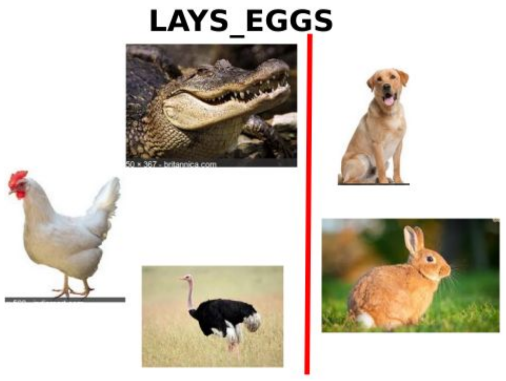
The Machine Learning algorithm that will be used is a Random Forest. This algorithm is an ensemble of learners. It's an ensemble of decision trees, getting its name because it's an algorithm that groups many trees. The 'Random' part of the name, comes from the fact that the algorithm chooses random features to make a split from a subset of some given number of best features. Let's look at this in a little bit more detail.

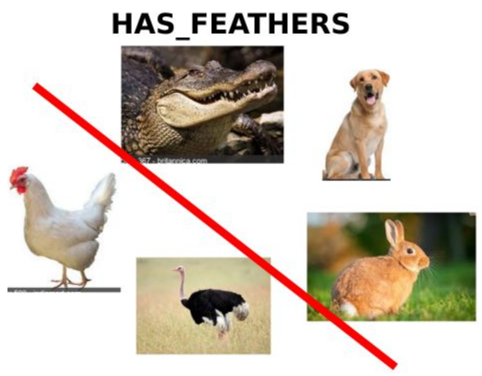
A decision tree works by making splits on the data. So, say you wanted to classify a certain number of animals by their features, and let's consider that the features you choose to do so are: *name*, *age*, *has_feathers* (which indicates if the animal has feathers), *has_fur* (as opposed to has_feathers), *number_of_legs* and *lays_eggs.* And let's suppose that you have the following animals to classify:

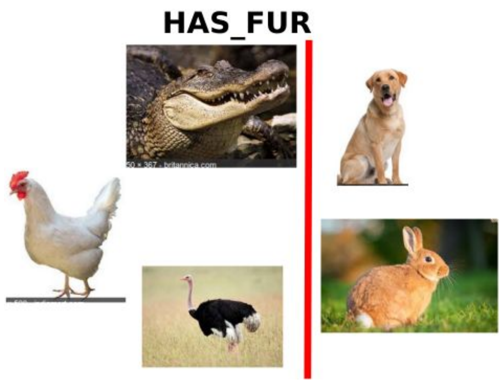


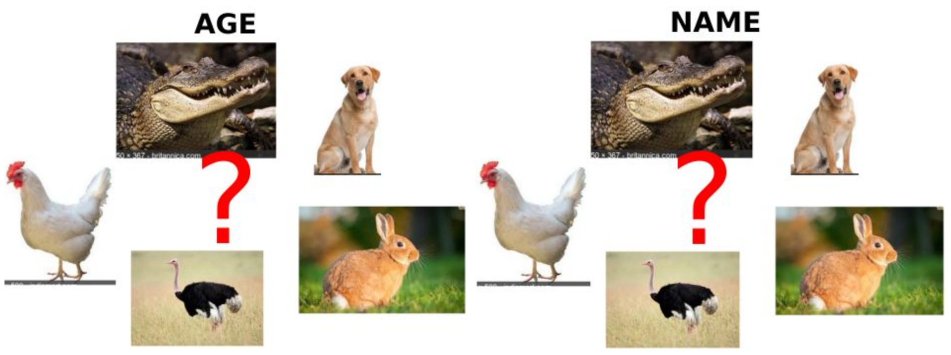
Then, the decision tree would have to choose some feature to make a split. Here are the splits that would result according to the different features.











Looking at this example, the first four features would be good for a split, and after splitting by one of those, say *number_of_legs*, you could split the above group by another one like *has_fur*, leaving with sort of correctly classified mammals, birds and reptiles. However, if we made a split by features like name and age, different groups might be formed, leading to incoherent classification. This ability to make a good split, a split that will put apart animals or things that can be differentiated, is a way to look at the information gain. The information gain is the way to calculate if, after a split, the resulting separated data will still look all the same or have fewer things in common. It's a very important concept in decision trees.

The Random Forest, then, is an algorithm that groups multiple decision trees, and these decision trees can split the data by selecting randomly one or two features of a subset of the best features. So, looking back at our example, we could have four trees, that made splits by *number_of_legs*, *lays_eggs*, *has_fur* or *has_feathers*, since these features all provide a good information gain.

The process, then, to train a Random Forest, would consist of taking a large amount of data, such as animals with their features, and deciding which series of splits would provide a better classification of the animals. Then, to make a prediction, a new animal's features would be looked at and it would go around the best overall path to tell which kind of animal it is.

Methodology

Regarding data preprocessing, a couple of adjustments had to be made. The first one was to obtain only the Close price from the originally retrieved CSV file and to reverse it, since the prices were listed from most recent to oldest. This was a simple procedure with the functionalities provided by the Pandas library.

After having that single price ordered from oldest to most recent, the data was normalized, dividing every value by the first one, so that the first value would become 1 and serve as a reference point to see how the stock price varied from then on.

Most of the indicators that were calculated could be seen as preprocessing, since most of them were obtained only with the stock price. The first indicators that were calculated are the rolling mean and Bolinger's bands. The rolling mean takes as input some date and a number of days for a window. Then, the average of the window is calculated and returned as the output. What this produces is something similar to the overall movement of the stock price, but much more smooth, less volatile, as can be seen in the cart on page 4.

The rolling standard deviation can be calculated similar to the rolling mean. Pandas Dataframes' have methods that make this easier. After calculating the rolling mean and stdev, Bollinger's bands can be calculated. These bands, which can also be seen in the chart on page 4, are two rolling stdevs above and below the rolling mean. What this produces could be seen as an outlier detection system, and whenever the stock price crosses one of the bands, it's usually likely to regress rapidly to the rolling mean, thus indicating either a selling or buying point.

After these indicators, beta was calculated. This was one of the features that required greatest effort. It required special effort because to calculate it, daily returns had to be calculated both for the stock price and for the benchmark model (which in this case was the SPY index). After these daily returns were calculated, a linear regression had to be performed and then the slope had to be calculated.

The last indicator that had to be calculated was the Sharpe Ratio. This is an indicator of adjusted return and is calculated by dividing the mean of the daily returns minus the risk free rate of return (which is expected to have a value very close to 0) by the standard deviation of the "portfolio" return. However, since the daily returns had already been calculated to obtain the beta indicator, only the average and standard deviation of these had to be computed.

For the training process, the first that needed to be done was to generate the labels to train the model. This was accomplished depending to the scenario in the following way:

- Scenario 1:
  This scenario was when the user wanted to predict using all of the data and only predict for a given number of days ahead. So if for example the user wanted to predict the price of some given stock for the next five days, this is what would happen: All of the stock data would be used except for the last five days. And each row of data would be the features for the label that would be the close price for five days ahead.

- Scenario 2:
  This second scenario was when the user wanted to get predictions for some given date range. So the user would specify the date range, then, if for example the span between the given dates were of five days, the same process for scenario 1 would be made using all the data from the beginning up until the initial date.

After generating the labels, the data left was an *X* matrix of training features and a *y* vector of labels. This data would then be fed to a machine learning algorithm. The initial idea was to use ensemble learners, specifically with AdaBoost and a base classifier which would be a Support Vector Machine regressor or a K-nearest neighbours regressor. However, after multiple parameter tuning tests, a Random Forest regressor was chosen.

Random Forest regressors have proven to compare favorably to AdaBoost when looking at the error rates[3]. Considering this problem, it's really important not to overfit, but rather to generalize. Random Forest regressors meet this requirement because the error rate always converges to a number, thus preventing them from overfitting. A core idea is to split by a random feature amongst the n best features to split by, and typically, a good number of features to split is 1 or 2. This is also good in our case, given that our dataset consists only of eight features. This, however, doesn't mean that Random Forests perform poorly on datasets with a large number of features. In his experiment[4], Breiman experimented with a dataset that had 1000 input variables, 1000 examples and 4000 test samples, and achieved an accuracy comparable to Bayes' rate.

The chosen values for the parameters of the regressor are the following ones:

- N_estimators:

  This was assigned a value of 50. It represents the number of "trees" in the forest. As was discussed above, overfitting isn't that much of a problem, so choosing a high number of trees wouldn't be harmful for the model.

- Criterion:

  This parameter was assigned to 'mae', which stands for Mean Absolute Error, the metric that is being used to measure our results.

- Max_features:

  This parameter was left to its default value, which is 'auto' and basically considers all of the features when looking for the best split. This can be done without much harm, because our dataset consists of only eight features, thus it's not as costly to take them all into account.

- Max_depth:

  This parameter was also left to its default value, which is None. And when None is passed, the nodes will be expanded "until all leaves are pure" according to sklearn's documentation, which means that the nodes will be split until no split can be made. Or if a min_samples_split is specified, splits will be made until nodes have that minimum number of samples.

- Min_samples_split:

[3] Y. Freund & R. Schapire, *Machine Learning*: *Proceedings of the Thirteenth International conference*, ***, 148–156

[4] Breiman, L. Machine Learning (2001) 45: 5. https://doi.org/10.1023/A:1010933404324

This parameter, as briefly mentioned above, specifies the minimum number of samples that a node has to be made in order to allow for a split. This parameter was set to 6.

- Min_samples_leaf:

  This parameter was left at its default value, which is 1. It specifies the number of samples that a node needs to have in order to be considered a node.

- Max_leaf_nodes:

  This parameter was set to None, which allows for an unlimited number of leaf nodes to exist.

- Min_impurity_split:

  This parameter was left untouched. It defaults to 1e-7, and indicates that if the impurity of a node is above that threshold, it splits, if it's not, it's considered a leaf.

- Bootstrap:

  This parameter was left to its default, which is False, indicating that no bootstrap samples are being used to build the trees.

- N_jobs:

  This parameter indicates the number of jobs to run. It defaults to 1, and was left as such. We aren't really working with massive amounts of data or features to say that the computing power required is too big, so for this project, 1 job is enough.

- Random_state:

  This parameter was set to 42. Indicating that 42 is the seed that should be used by the random number generation

- Verbose:

  This parameter is good to see what's going on in the process of the algorithm. However it was finally left to 0, since it's really not that relevant to see that information up on the console.

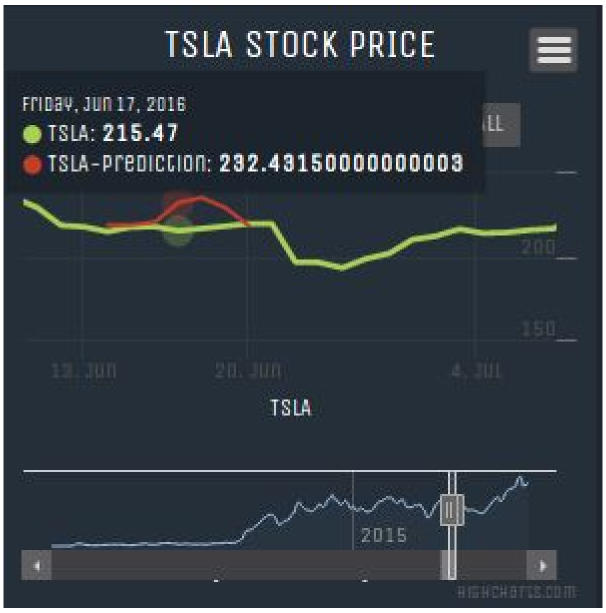
- Warm_start:

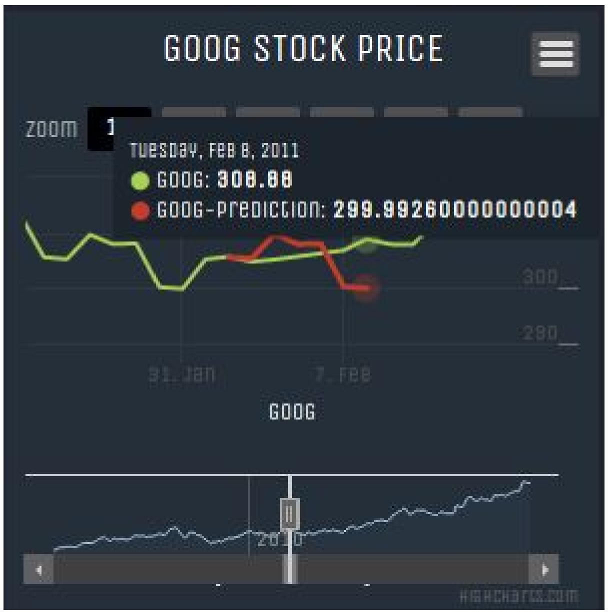
  This was left as false, to indicate that it was desired to fit a whole new forest each time instead of using the result of the last one.

Before deciding to go with Random Forests, some attempts were made with AdaBoost using Support Vector Regression and K-nearest neighbours regression as base classifiers. However, the results that were yielded by these algorithms weren't too good. Using a polynomial function as the kernel for the SVR took an exaggerated amount of time. Using a linear function ended up returning the same values of the last n days queried. Using K-nearest neighbours as the base classifier yielded better results, but the results were pretty close to the ones obtained with the Random Forests. So, Random Forests were kept.

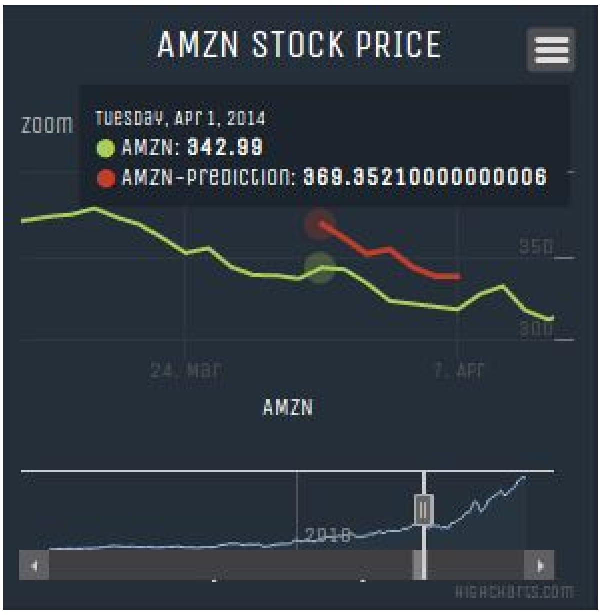
Results

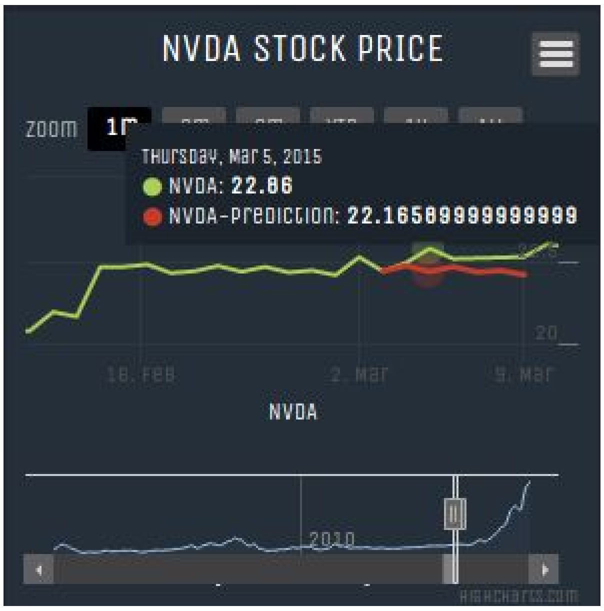
Below are a couple of screenshots taken from the web application that was developed, showing some points in time with the actual stock value and the predicted one for different stocks.



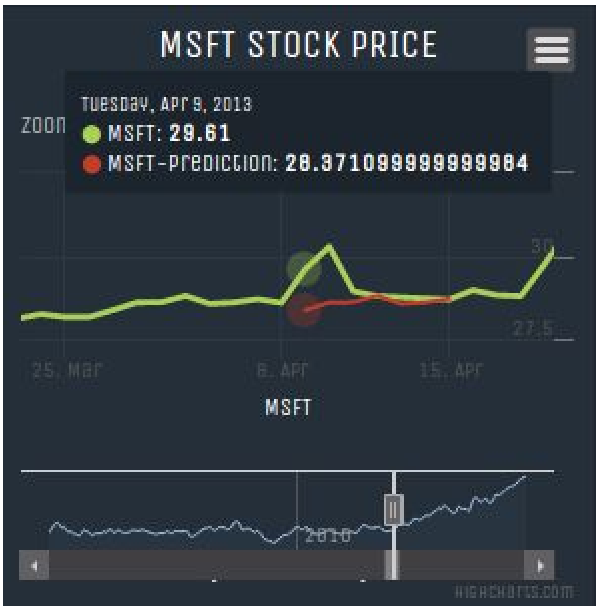


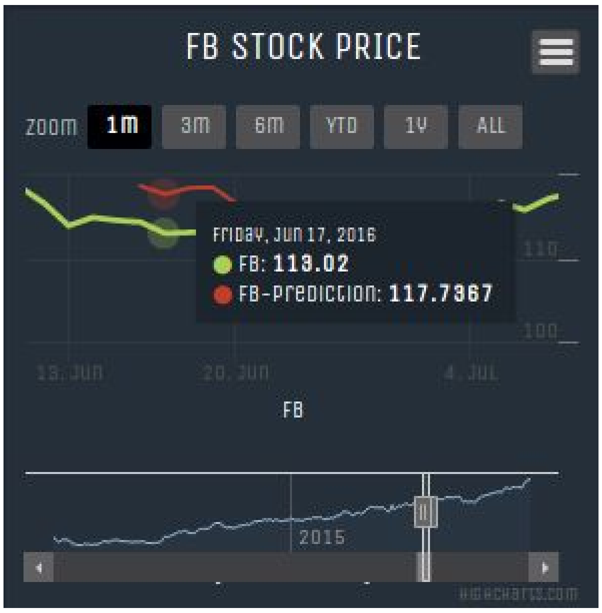
As can be seen in these charts, the predictions for one week were off by a maximum of 17 dollars in the case of the Tesla stock, which is about 8%, and around 9 dollars in the case of the Google stock, which is about 3%





On this second set of images, the differences were of about 4.7% and 3.06% respectively. It's possible to see that even though the range of the stock prices is very different, the predictions stay within close ranges.

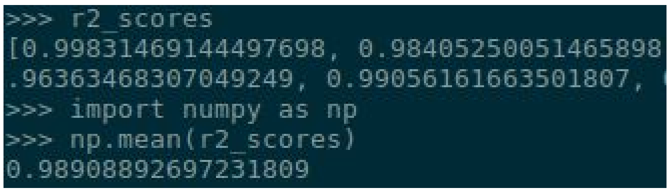




On this last two charts, the predictions are off by about 6.7% and 4.16% respectively. However it's important to note that in some of the charts seen above the predicted values were somehow more volatile than the actual ones, and in some other charts, the predictions failed to predict sudden fluctuations that the actual stock price took. However the results are very good, considering that in each case only the data until the initial date was used, leaving in some cases plenty of data unseen.

To calculate the R² score, previously a train test split had been generated, and the R² score had been calculated, obtaining a score of 0.99999, however, this is not a good way to really measure the R² score, because the data corresponds to a time series, and when a train-test split is made, the data gets randomly shuffled. This leads to future data getting mixed with past data and sometimes the test data falls in between, allowing for an extremely high score. This flaw was indicated by the reviewer the first time this project was submitted.

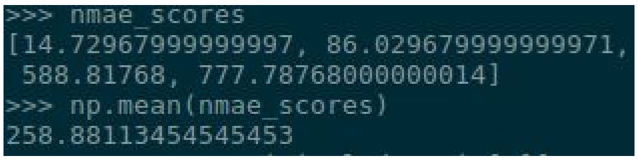
The new way to calculate the R² score, consisted of splitting the data into smaller datasets of about 300 samples each. Then, the first 290 samples would be used to train, taking as labels the close prices from samples 5 to 295. Features from 290 to 295 would be passed to a model's predict method, thus obtaining five predicted y values. Which would be compared with the close prices of samples 295 to 300. Then, considering that this splits made a lot more sense, since the weren't being shuffled, the R² score was calculated for each of the new smaller datasets. Finally, the R² values were averaged.



```
>>> r2_scores
[0.99831469144497698, 0.98405250051465898
.96363468307049249, 0.99056161663501807,
>>> import numpy as np
>>> np.mean(r2_scores)
0.98908892697231809
```

The new obtained value for R² was of 0.989, which is really good but nowhere near the previous and absurd 0.99999. This value of 0.989 indicates that the variance in the dependent variable can be explained pretty well by the independent variables.

To recalculate the Mean Absolute Error, a similar process had to be made. Previously, we had obtained a ridiculous value of only 0.006, that translated back to dollars is only about 0.3 dollars, which makes no sense. From the split described above, the y_test for every sub dataset was kept and predictions were also made for each sub dataset. This predictions were fed to sklearn's mean_absolute_error metric. Then, the MAEs for all the datasets were normalized, converted back to dollars and averaged.



```
>>> nmae_scores
[14.72967999999997, 86.02967999999971,
 588.81768, 777.78768000000014]
>>> np.mean(nmae_scores)
258.88113454545453
```

The new Mean Absolute Error obtained is of 258.88 dollars. This is a much higher value than the one we had previously obtained, and it's also one that makes more sense. 258 dollars seems like a pretty high value. However, this is an absolute error value, meaning that sometimes the predictions were off by predicting the stock would cost more, and sometimes it predicted that the stock would cost less than it actually cost. So, every margin here adds up, and it's possible to observe that as the stock price increased (namely in the last couple sub datasets), so did the error.

Conclusion

As was seen along this project, it's not that difficult to create a simple stock price predictor. Nevertheless, this doesn't mean it's an easy task either. There were some particularly challenging parts to the project. It's not that easy to choose a metric, specially when there are so many metrics available. And even considering the type of project, it would be possible to think that the number of relevant metrics would drop drastically, but it doesn't necessarily do so.

Another challenging parts emerged when developing the web application with its GUI. Working with different UI libraries and linking everything with the backend isn't that easy. Designing how to display everything was definitely one of the most challenging parts.

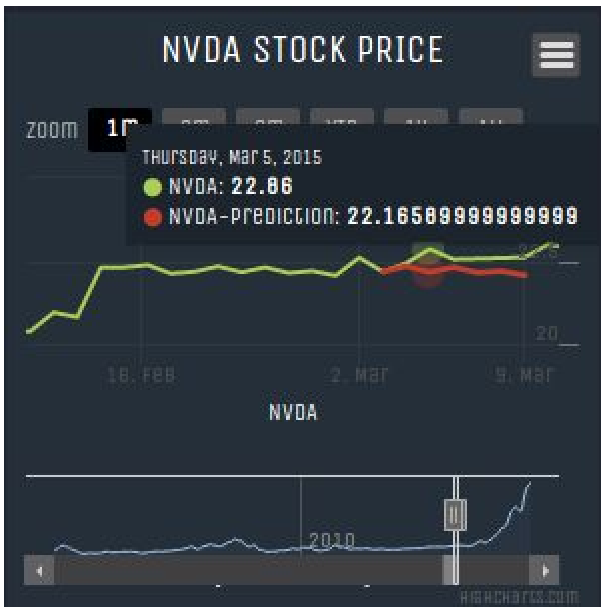
Other parts were challenging but perhaps not as much as the ones already described. But it's not easy to try and immerse into the world of finance and trading for a project. It's not a trivial task to choose indicators that might be relevant for making predictions.

All of these things that needed to be learned are, despite the challenge that they represent, very interesting.

Developing a project from the ground up helps to see how important every step of the way is. From the gathering and preprocessing of data to the parameter tuning of the model. Regarding that first aspect, there was a problem with an API that was going to be used but was closed to the public when a lot of things from the project had already been developed. This challenge forced the author to spend valuable time searching for similar resources.

Working on a project in this fashion also helps to appreciate the role of every person in a team. It's likely that one will usually feel more comfortable working in some special area, sometimes not giving other areas the credit they deserve. Having to work in tasks that aren't usually one's area of expertise, really forces one to appreciate the fact that others usually do that job.

The results were satisfactory. The project might show that Random Forests are a good algorithm to use when predicting stock prices. However they must be combined with some relevant features to yield good results. It's probably still too rough to rely on it, and the stock market is extremely unreliable, so it probably wouldn't be such a good idea to buy or sell stocks based on this system, specially considering that the obtained error was pretty big. But results are satisfying for a simple project.



Research in areas such as Deep Learning is advancing everyday, and Deep Learning techniques are already being used to predict stock prices. It would be interesting to come back at this problem after studying such techniques and see if the results are improved.

It would also be a really interesting approach to solve the problem as another type of problem (as was mentioned in the first pages). Like, for example, building a trader using RL that suggested whether to buy, sell or hold a given stock. But for now, this project has been more than enough in the financial field.