MACHINE LEARNING PROJECT

Predicting BTC Price Direction using Machine Learning

Klaas Schoenmaker
22 December 2017
Gosia Migut

The Issue

If you have any experience with Bitcoin trading (or asset trading in general), you probably know that it's very frustrating to buy a specific asset, and then seeing its value plummeting the next day. It would be perfect to have a program that could tell you wether the value of your specific asset will rise or fall the next day, so you could anticipate that and buy/sell on time. The Bitcoin market specifically is very volatile and its price can easily vary 30% on a daily basis. Being able to see patterns in this seemingly random string of numbers could be invaluable to professional investors and could give people insights in how and why the market will behave at a given date.

This is why during this project I will try to give a simple 'yes' or 'no' to question of wether the Bitcoin will increase on a given day, knowing the closing price of the days preceding it. In order to be able to answer this question, I'll be using several machine learning techniques with different data sets to see which approach works best for this specific issue.

The Existing Literature

A paper that influenced a good amount of the decisions made during this project was the paper called "Predicting Stock Price Direction using Support Vector Machines" by Princeton student Saahil Madge. This paper doesn't look at one stock specifically, but rather at stock indexes like the S&P500. These indexes are collections of several stocks combined, and thus not the same as Bitcoin which is only one stock. They do however exhibit a lot of the same properties like momentum and volatility, which are the two main features Saahil uses in his paper, and which I'll be using as well. Saahil, like me, also used the scikit-learn package and Python in order to create his classifiers, which made for a good reference point for my own research.

The Data

First of all, I had to find a dataset that correctly portrayed the value of Bitcoin over time. I found several datasets on websites like Kaggle, but none of them really fit all my needs. Either the sets were too small, or they had missing data points. In the end however, I came by a dataset that was pretty much perfect: the Coindesk API[2]. This API contained the daily closing prices of Bitcoin from the 1st of September, 2013 until the 1st of September, 2017[1].

After finding this data I had to transform it slightly in order for it to fit the machine learning algorithms I was going to use later. For this to work I started of by removing the date labels that the object originally contained, and decided to just turn it into an array (from oldest to newest closing price). Next up I had to pick the features and classes that the set was going to consist of. I ended up picking the following features, based on the features Saahil Madge used in his paper on stock price prediction:

Feature Name	Formula	Description
Volatility	$\frac{\sum_{i=t-n+1}^{t} \frac{C_i - C_{i-1}}{C_{i-1}}}{n}$	This is the average of the relative change of the price over the past n days.
Momentum	$\frac{\sum_{i=t-n+1}^{t} y(i)}{n}$, where $y(i) = 1$ if $C_i > C_{i-1}$, else 0	This is the average of the price's upward momentum. One is added to the upward momentum whenever a day in the range has a higher price than the previous.

*Here **t** is the number of entries in the set of all prices. **C_i** is price at point **i**.*

As you can see in the features, there's one returning variable that isn't really defined anywhere else yet, namely n. What n basically is, is the number of days that you take to predict the day after those days. So, for example if you take n=7, this means you look at the prices from the week before the day you want to predict in order to try to predict its value.

Since I didn't know yet which value of n would be optimal for predicting the class, I decided to create three different datasets for **n=7** (1 week), **n=14** (2 weeks) and **n=14** (±1 month) and tried all these datasets on

my different classifiers.

Next up, I had to define my classes, which was relatively straight forward. The data is divided in two classes; 'up' and 'down.' This basically means that if on a given day the price was higher than the last, it's classified as 'up', and otherwise it's classified as 'down.' When a price went up, I gave it the value **1**, and when it went down, it got the value **-1**.

This resulted in a number of compact sets of JSON data like the following:

```
[…, {"X":[122.39902142857143,0.5714285714285714],"y":-1},
{"X":[122.28307857142856,0.6428571428571429],"y":1},…]
```

Last of all, this set was split into a training and a test set. The ratio training to test was approximately 2 to 1, and the respective sets were shuffled afterwards.

Note: The code I wrote for generating a correct dataset from an array of prices can be found in the *datasetcreator.js* file.

The Classifiers

The next step in trying to find correct Bitcoin price direction predictions was finding the right classifiers for the job. As I was using the scikit-learn package for Python, I was bound to the classifiers they provided out-of-the-box. After reading through the documentation and considering multiple options, I landed on the following three: a **Support Vector Machine** (sklearn: *SVC*), a **Multi-Layer Perceptron** classifier (sklearn: *MLPClassifier*) and a **Gaussian Naive Bayes** classifier (sklearn: *GaussianNB*).

From the beginning I was quite sure I wanted to use the SVM because of its flexibility and thus its adaptability to different problems. The same goes for the MLP neural network, since these are usually very well fit for recognising patterns in data, which I thought was very important in this specific issue. Naive Bayes wasn't actually highest on my list at first, but I figured that it might pay off to try a completely different approach as compared to the other two to see how it compares to the others.

For both the SVM and the MLP I tweaked their parameters using grid search with cross validation in order for them to perform optimally on both the training and test sets. As mentioned before, there are three

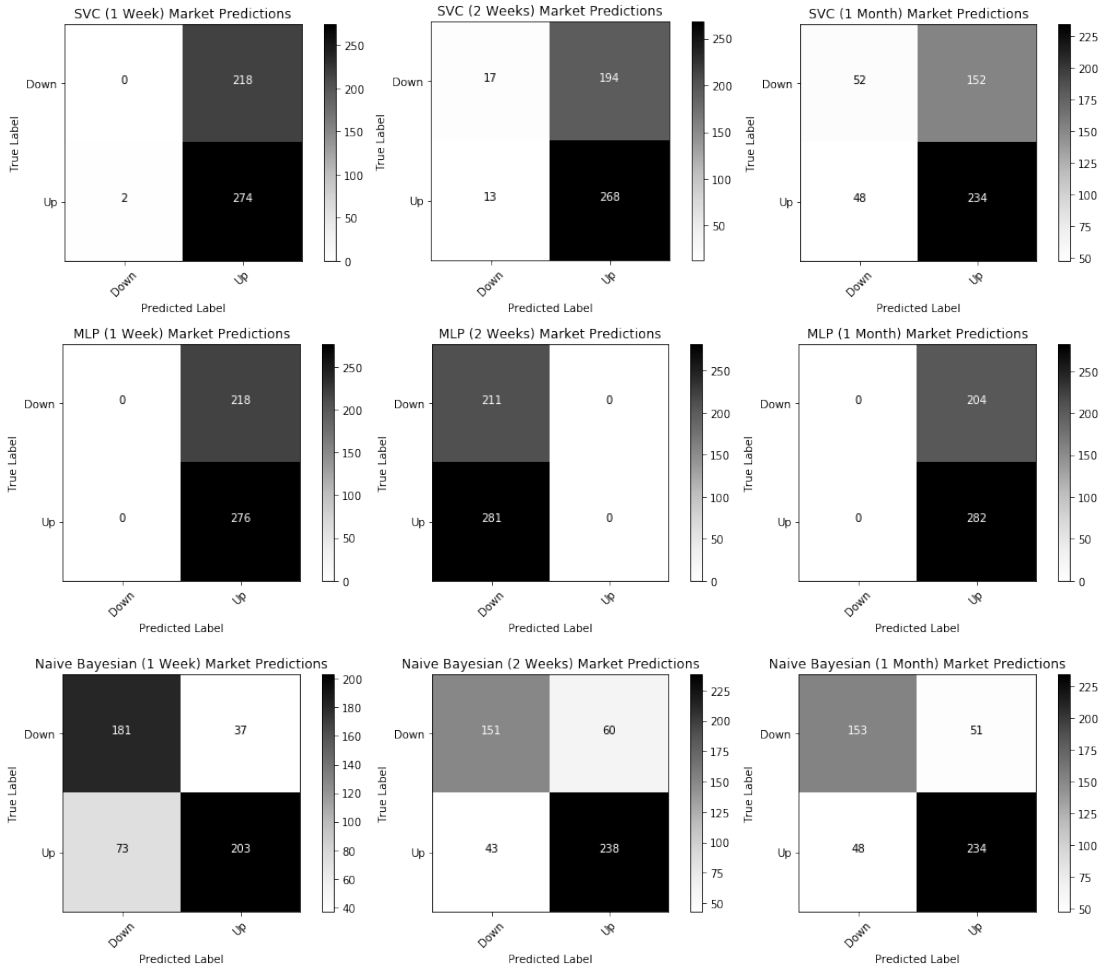
variations on the dataset, all with different values of n. Because of this variation, you end up with three different versions of every classifier, each with a slightly different dataset and parameters. In the end, this results in a total of nine classifiers.

The Results

After fitting and optimising each classifier, I ran a number of tests on them to see how they all performed. First of all, I ran cross validation for each of my classifiers and calculated its precision, which resulted in the following scores:

Classifier	n	Cross Validation Accuracy Score	Precision Score
Support Vector Machine	7	0.575424575425	0.31
	14	0.58734939759	0.57
	30	0.605476673428	0.57
Multi-Layer Perceptron	7	0.579420579421	0.75
	14	0.584337349398	0.33
	30	0.614604462475	0.50
Naive Bayes	7	0.748251748252	0.79
	14	0.761044176707	0.79
	30	0.788032454361	0.80

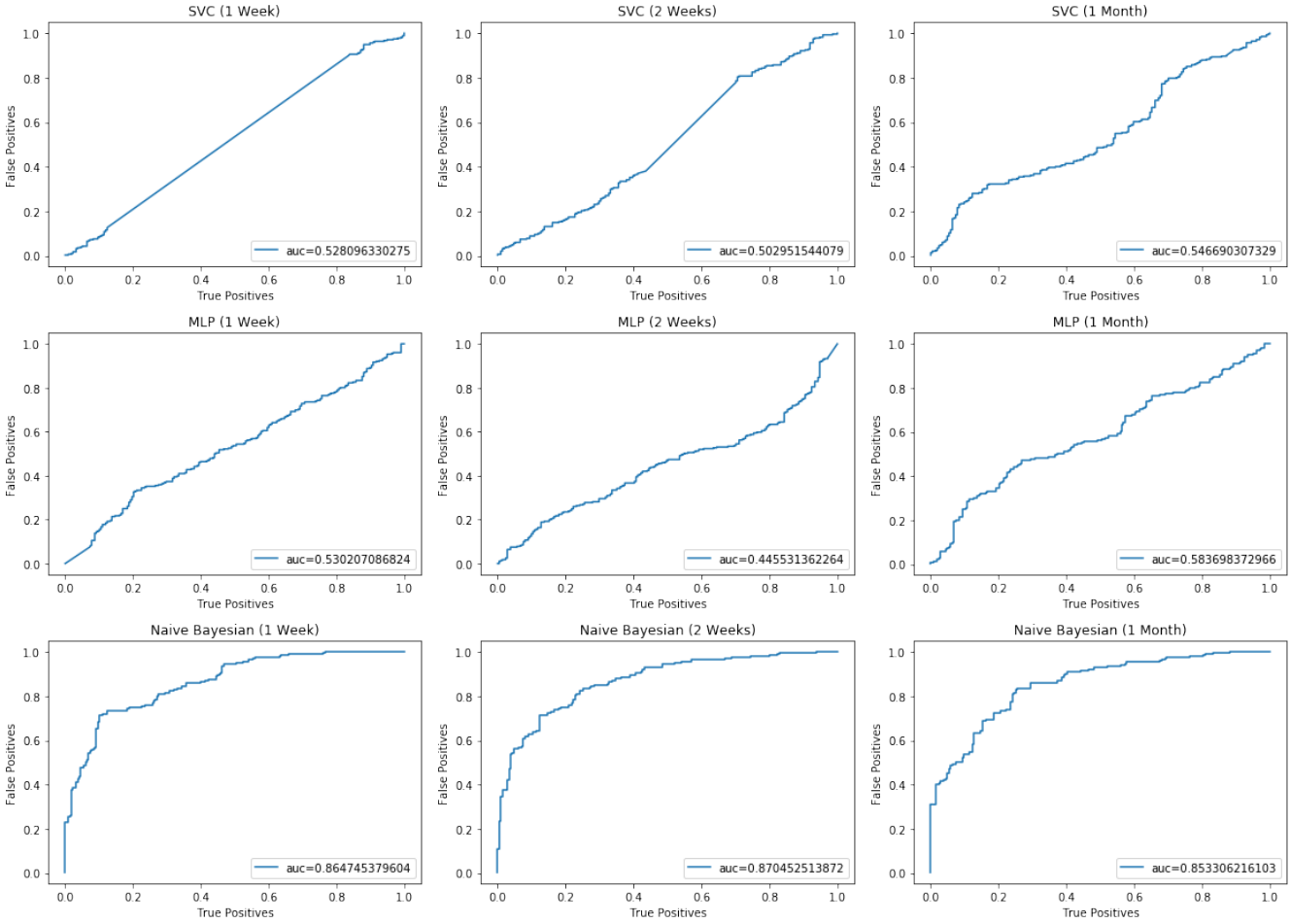
As you can see, the results of the SVM and the MLP aren't too great. If you look at the accuracy score of the SVM with n=7, you get a score of ~57%. For a comparison, basing your choices on a simple coin flip would already result in a 50% accuracy score. Even though the SVM and MLP scores get better with higher values of n, the optimal score they achieve is slightly more than 61%, which still isn't amazing. When you look at the Naive Bayes classifier, however, it gets consistent scores of around 75% accuracy. This means for every arbitrary 4 days, it predicts the correct direction for approximately 3 of them. This is actually a relatively good



how the classifiers predict, though. To get a better image of this, I ran some tests concerning the rates at which the classifiers predicted the correct/false values. Let's check out the confusion matrices of the classifiers:

Confusion matrices for the several classifiers.

Ideally, what you would see in these images is a diagonal black line from the top-left to the bottom-right. Obviously, this is not the case with the SVC and MLP classifiers, which mainly show a bias for one specific class. For example, the MLP with n=30 actually only predicts 'up' for every given set. On the other hand, the Naive Bayes classifiers seem to portray a slightly better matrix, which in most cases almost



resembles a diagonal line. Another way to visualise this difference between SVC and MLP, and Naive Bayes is through a so called ROC-curve, which portrays the ratio of true-positive to false-positive predictions:

ROC-curves for the several classifiers.

What you would want to see in these graphs, is a very high area underneath the line, which is indicated by the AUC-value (area-under-curve). As you can see in the SVC and MLP curves, the value is consistently around 0.50, which indicates a near-random classification, as a straight diagonal line across the graph would be the same as flipping a coin. Again, however, the Naive Bayesian seems to perform better in this field. The graph very nicely goes toward the top-left corner, which indicates a higher AUC-value, and thus a relatively better precision than the other classifiers (less FP's than TP's).

The Conclusion

When considering all the data we gathered, we can conclude that the best way to accurately predict the price direction of Bitcoin is by using the Naive Bayes classifier, using a dataset that uses n=30 days as its features. This classifier has both the best accuracy and precision, and thus generally has the best predictions out of all the classifiers. There are some caveats to consider when implementing this technique in real life, though.

First of all, the Bitcoin market over the last few years has been extremely bearish, meaning it has mainly been going up except for some small crashes. This means that the dataset I used mainly has datapoint that contain upward trends, and not too many downward ones. This could result in the algorithm performing well as long as Bitcoin keeps rising, but failing to predict correctly when the market goes into a downward or crashing trend.

Second, this technique only predicts wether the market goes up or down, and doesn't associate any quantifier with this. This means that the algorithm only tells you "the market will go up tomorrow", instead of "the market will go up 40% tomorrow." Creating an algorithm that could predict this properly would be way harder and would probably require some advanced optimisation and training (if it's even efficiently possible). In the end, this technique definitely isn't ideal or optimal, but it simply shows that it's possible to predict trends in the Bitcoin market, and that the data definitely isn't as random as it might sometimes seem.