CS 11 Bitcoin Data Project Summary

Grant Messner

June 1, 2020

Overview:

The main goal of this project was to use machine learning techniques to make predictions about future Bitcoin prices using historical data. The dataset I used is https://www.kaggle.com/mczielinski/bit historical-data, containing minute-by-minute price data (high, low, open, and close prices), as well as trading volume. I implemented several different approaches to try and solve this problem, from simple (linear regression), to complex (neural networks trained on the residuals of the linear fit, including long and short term weighted averages). I got inspiration for some of these implementations from a couple of articles I read online: https://towardsdatascience.com/ml-approaches-for-time-series-4d44722e48fe, https://towardsdatascience.com time-series-analysis-and-trading-strategy-with-bitcoin-price-data-1a8f1a30f11. I'll go into more depth about each major approach that I tried below, but on the whole the simpler approaches produced very good results, and the more complex approaches were used for minor corrections.

Linear and ridge regression:

The first approach that I tried was simple linear regression (after I tried a "zero order" approach of just using the last close price to predict the new open price, which actually performed decently well). I played around with different window sizes (i.e., looking at farther periods into the past, as opposed to just the most recent period), and surprisingly found that using only one period (window size of 1) was the best. It is possible there was some overfitting going on with larger window sizes, since they produced lower training errors but higher test errors. I went with the window size of 1 for the rest of the project after I discovered this. Next, I tried working with ridge regressor from sklearn. I had to really jack up the α parameter (which controls regularization, i.e., higher α is more regularization) in order to get the error rates to change at all from the linear fit, but surprisingly these all had worse e_{out} values than the linear fit with no regularization (even for the larger window sizes).

Other sklearn models:

From here, I tried playing around with various other types of regression models that sklearn had to offer. The first one I tried was lasso regression. This produced pretty terrible results no matter how I varied α, as you can see from the notebook, so I scrapped this and moved on. It was a very similar story with the histogram gradient boost regressor, as well as the random forest regressor. These models are more powerful than simple linear regression, but did not produce as solid results as the linear model did for this specific dataset. Finally, I tried a Gaussian process regressor and ARD regressor, which both might have generated decent results but for whatever reason my computer couldn't really handle them (both ended up crashing my computer and forcing me to either restart Jupyter or restart my entire computer)! So, I stuck with the linear model with window size 1 and

moved on.

Neural net:

From here, I began using PyTorch's neural net package to make predictions. The first approach I tried was just feeding the data directly into a neural net to train, and this did OK but not nearly as well as the linear fit, even after I tried several different numbers of hidden layers, nodes, hyperparameters, etc. However, I didn't scrap the neural net, because I figured I might be able to train it on the residuals of the original fit to produce better results, and this is exactly what happened. The improvement wasn't much and didn't occur on every training run, although even a small edge like this could be extremely profitable in the trading world. This neural net residual approach would be useful later as well.

Symbolic regressor:

The next approach I tried was a symbolic regressor (from gplearn), which basically constructs random functions out of the input variables using whatever operations you want to pass in ($+, -, \times$,etc.) and tests each to see how well it actually predicts. This is a pretty powerful approach as well, although it pretty consistently learned only the most recent close price as a prediction for the next open price (i.e., the "zero order") prediction I discussed at the beginning. If I had a bit more time to spend on the project, I could have tried to fit this to the residuals as well.

Moving average + neural net:

The final improvement I was able to make to the original predictions came in the form of a neural net that I trained on the residuals of the linear fit and appended a moving average to the training matrix X. I wrote a function that allowed for an arbitrary moving average to be computed (arbitrary weights and arbitrary number of periods into the past). I experimented with this a bit but the best result came from an equally-weighted average 10 periods into the past being the best. This produced a small improvement over the linear fit as well.

Conclusion:

Overall, I think this project taught me that financial data is complicated, and generating small improvements is extremely difficult! It also showed me that sometimes the simple approach can produce very good results, as is evident from how good the linear regressor performed relative to the more powerful models. I enjoyed working on this project, and feel like I spent an appropriate amount of time on it for how the class was united (30 hours over the course of the term).