EE P 596 Mini Project 1
Q1 Report

Ryan Maroney
Blake Downey

2/27/2022

Introduction

This part of the mini-project, Q1, tasked us with creating at least two baseline models and at least one ML or supervised model for predicting whether it is a good time to buy or sell in the stock market for a list of 5 NASDAQ tech stocks of our liking. For this assignment we chose to run backtesting on the stocks: AMZN, GOOGL, AAPL, NVDA, and AMD.

For our baseline models we implemented Simple Moving Average (SMA) and Exponential Moving Average (EMA). For our ML approaches we implemented Seasonal and Trend decomposition using Loss (STL), a Ridge Regression (LR) classifier and a Keras Sequential DNN (DL) that we built.

Code Implementation

Base Trading Strategy Class

The base trading strategy class is used to define the API for buying and selling with Alpaca, and to define a common API for running backtests and live trading across our different strategies.

Baseline Crossover Strategy

The SMA/EMA cross strategy is a simple strategy that implements a “fast” and “slow” moving average. The “fast” moving average more closely follows the data (e.g. shorter period for SMA, and larger factor for EMA), and the “slow” moving average less closely follows the data and more follows the general trend (e.g. longer period for SMA, and smaller factor for EMA).

If this fast moving average crosses above the slow moving average this is a sign of an increasing trend and indicates that we should buy. If the fast moving average crosses below the slow moving average this is a sign of a decreasing trend and indicates that we should sell.

STL Strategy

For one of our ML approaches we used an STL (Seasonal and Trend decomposition using Loss) strategy which is implemented using Facebook’s Prophet API. For this strategy we fit the data and make predictions using the Prophet STL model, then determine whether to buy or sell if the actual price is too far above or below the predicted price and bounds.

Prophet allows for the ability to forecast prices for a specified number of days into the future. The forecast comes with a y_hat prediction, and y_hat_lower and y_hat_upper bound values relating to confidence in the prediction. For this strategy we if the current

price dips below some factor of the lower bound the algorithm suggests to buy. If the current price rises above some factor of the upper bound the algorithm suggests to sell. This algorithm is based on detecting anomalies and making the assumption that the trend will return to something closer to the predicted state. One key hyperparameter for this strategy is to adjust the bound factor, a large value means that only large anomalies will be detected, and a small value will also detect smaller anomalies.

LR/DL Strategies

The Data

To begin each of these, LR and DL, we needed to construct a set of labeled data. To accomplish this efficiently we used a few methods: create_data(), concat_dfs() and feature_stack(). Using these three methods we could create different variations of the stock data to test on the LR and DL architectures with varying hyper-parameters.

When calling the create_data method, it creates a dataframe (with shape (n,8) where n is the number of days requested - 17 days. This is a result of generating data for sma/ema and calculating the slopes for each point) that has the instantaneous values of the stock used to generate the data. These values include closing price, sma, ema, std, upper and lower bound std calculated from the sma plus or minus the actual closing price, the point slope from last closing price to current closing price and lastly the labels. We also used feature stacking for training and testing the LR model as well as the DNN model. This will take the last 4 days of information (closing price, sma, ema and slope) and append it to the 5th day for each sample, making the data frame of shape (n-4,24). We found that when we feature stack we have a more generalized solution.

The Approaches

For our other ML approach, we used the Ridge Classifier from sklearn. We trained a new model each time we wanted to backtest a new stock with a custom alpha for each stock chosen. For each model, we trained on GOOG, MSFT and FB stock over the course of 1 year with a varying alpha hyper-parameter. We did this because the model performed much better when seeing multiple stock's data over a longer period of time. We made sure to choose different stocks for training then we did for backtesting. For the purpose of back-testing, we used the last 5 months from the last year of closing prices for each stock that we chose.

For the DL approach, we created a new model for each new stock to backtest. (Each model is created and trained upon execution time so results will vary). The model uses 1 year of data and is trained on the first 7/12ths of the data and back tested on the last 5/12ths of the data. The architecture, hyper-parameters and all extra functions used to generate it can be found in the code. We chose to use a Keras sequential model to

build it, however if done again we would want to implement either an LSTM or RNN model and see if the results would be better. Fluctuating the hyper-parameters and the dataset heavily influences the output as the stock market is a volatile place and a hard one to predict.

Backtesting Results

As can be seen from the tables below, the Logistic Regression model performed the best, followed by the DNN model, and STL strategy. The baseline crossover strategies were not very successful and resulted in similar performance.

The baseline strategies are heavily dependent on the hyperparameters and vary greatly from one stock to the other depending on which parameters are used.

STL performed better for stocks that had less of a general trend, like AMZN and GOOGL which had many ups and downs.

Looking at the results from the DNN strategy vs the LR strategy, we can see discrepancies in the performance. Specifically the LR strategy performed better across the stocks APPL, NVDA and AMD, whereas the DNN performed better across AMZN and GOOGL. As noted in the discussion section below, the DNN performs either really well, poorly or doesn't work. In Table 2 below we can see that the DNN strategy followed this behavior as it performed well on AMZN (in comparison to the LR strategy), performed poorly on GOOGL, and didn't work on APPL at all. The graph in the notebook corresponding to the APPL stock also shows this as there are no buy and sell points marked.

Another potential metric for comparing the performance of these strategies could be to compare the potential gain with the actual gain. For example if the strategy was LR or the DNN, we had to generate labeled data for these. We could use the labeled data as the 'potential gain' since the labeled data buys on all dips and sells on all spikes which almost always results in net profit. We then could divide the actual gains by the potential and get a ratio of the two as the metric.

	AMZN	GOOGL	AAPL	NVDA	AMD	Total
SmaCross	$133.17	$-270.36	$13.05	$54.13	$12.87	$-57.14
EmaSmaCross	$2.36	$-293.50	$8.51	$49.59	$26.87	$-206.17
EmaCross	$124.62	$-38.25	$2.50	$-4.69	$3.22	$87.40
STL	$642.38	$155.60	$-1.78	$44.95	$-26.51	$814.64
ML-LR	$172.95	$-403.65	$601.41	$1138.99	$187.24	$1696.96
DNN	$333.40	$-23.39	$0.00	$605.66	$176.52	$1092.18

Table 1. Total Earnings of Strategies across the 5 stocks

	AMZN	GOOGL	AAPL	NVDA	AMD	Total
SmaCross	1.33%	-2.70%	0.13%	0.54%	0.13%	-0.11%
EmaSmaCross	0.02%	-2.94%	0.09%	0.50%	0.27%	-0.41%
EmaCross	1.25%	-0.38%	0.03%	-0.05%	0.03%	0.17%
STL	6.42%	1.56%	-0.02%	0.45%	-0.27%	1.63%
ML-LR	1.73%	-4.04%	6.01%	11.39%	1.87%	3.39%
DNN	3.33%	-0.23%	0.00%	6.06%	1.77%	2.18%

Table 2. Average of Percentages of Strategies across the 5 stocks

Discussion / Insights / Takeaways

Overall we found that each strategy seemed to perform well for some stocks and then poorly for others. And depending on which hyper-parameters were used the stocks that performed well changed, suggesting that the strategies are highly susceptible to overfitting. We go into more detail about the takeaways for each strategy below.

Baseline Crossover Strategy

We tried three different baseline crossover strategies. One using SMA for both the fast and slow averages, another using SMA for the slow average and EMA for the fast average, and another using EMA for both the fast and slow average. For the SMA only crossover the fast average used a 13 day average and the slow average used a 25 day average. For the SMA/EMA combination the fast EMA used a beta value of 0.3 and the

slow SMA used an 8 day average. For EMA only the fast average used a beta value of 0.8 and the slow average used a beta value of 0.6.

We found that none of the baseline strategies performed nearly as well as the ML models, but the strategy that used EMA was the most effective. The main trouble with these baseline algorithms is that they are always delayed (more or less so depending on which hyper parameters are used). This works well for long term upwards or downwards trends, but is ineffective for more volatile stocks that have a less consistent long term trend. For more volatile use cases the buys often happen at local maxima and sells often at local minima.

Decreasing the period will allow for the algorithm to be more responsive to volatile stocks, but even at this point the crossover will always be a step behind and will not be able to effectively manage day to day ups and downs.

STL Forecasting Strategy

Our STL forecasting strategy seems to show an interesting contrast to the baseline crossover strategy. Overall the strategy produced better results, but it was not effective in maximizing the potential profits of longer term trends. Some tweaking of the strategy may improve this, but the strategy often buys before reaching the local minima and sells before reaching the local maxima. However, STL does seem to perform better for time periods that are more volatile and have a less clear long term trend.

Given more time it would have been interesting to use some combination of the crossover strategy and STL to make use of each of their strengths. In particular this could be useful to incorporate both as attributes for our deep learning or logistic regression models.

Logistic Regression and Deep Learning Strategies

Our Logistic Regression strategy proved to be the most successful. The model was pretty successful in predicting when to buy and sell. From the predictions we throttled the amount of buys it could do to 4 as it tended to like to buy on downslopes heavily. Because of this we implemented the ML strategy as a dollar cost averaging strategy. It will buy as it is going down incrementally more shares, and when it sells, it sells all the shares and it must be greater than the total weighted sum of the shares bought in order to sell or it will hold. We found that this approach was more flexible than a greedy approach.

The Deep Learning strategy was the second most successful in terms of net gains. By observing the graphs it does seem to buy and sell at relatively appropriate times,

however it is much sparser than the LR strategy. We also implemented dollar cost averaging for this strategy in the same manner. A really interesting takeaway from this DNN strategy was that some stocks performed really well, some had net losses, and some the DNN didn't predict any buy/sell points. Since the stock market prediction problem is so complex, modeling it with a simple dense keras network is not enough to provide consistent gains when only training on 7/12th of a year.

One interesting take away from both the LR and DL strategies are the results of flash selling before earnings dates. Prior to implementing the 'sell before earnings' factor to the code, both the LR and DL models were performing very well with the DL model almost always performing better. After implementing the 'sell before earnings' however, the LR model always outperforms the DL model. I believe this is because the LR buy/sell predictions and actions taken are less sparse so when the time comes to sell before an earnings date the total losses are kept at a relative minimum.