CSE472 Project 2 Report

Social Media Data Analysis

Rui Heng Foo (1226940821) and Michael Liu (1225287050)

Step 1: Collect data

Instructions:

Start by setting up the prawscraper.py script, which will be used in order to scrape Reddit for sentiment training & testing data used in the following steps. There's 2 variables you can configure called "subreddits" and "tickers" which are used to select which subreddits to scrape from and stock symbols to look for respectively. The scraper will then look for 100 relevant posts from each subreddit and send them to their own csv file. Then, you will need to run the Alpha Vantage scraper script to get the stock prices for each ticker, which will be used in training the sentiment decision trading module in part 3. By default all the scripts should have NVDA and should scrape r/stocks, r/wallstreetbets, r/investing, and r/daytrading.

Implementation Details:

We initially wanted to use multiple platforms such as twitter and yahoo finance, but after seeing that twitter charges a hefty price for their API with no usable free tier, and that yahoo finance didn't have an actual API for their news portion of their site, we resorted to using only Reddit. We figured this would be fine since the different communities in Reddit were generally isolated enough such that they shouldn't impact the results. As for interfacing with Reddit, we tried PSAW as recommended by the project outline but it turns out that PSAW was restricted so we switched to PRAW at the end.

Step 2: Sentiment analysis

Instructions:

Make sure that all and only stock data files are located in part1\out\ folder and closing price data files are in part1\ itself. Then, run the sentiment_analysis.py to produce the daily sentiment score data file and the combined sentiment analysis results data file in the out\ folder. By default it will look for all stock data in out\ file.

Implementation Details:

Since the stock data files and closing price data files are conveniently separated. We can search for all files in part1\out\ (which should only contain stock data files), read them, and merge them into a large data object for training. Unnecessary words such as urls are filtered out from all content and updated to the data object as data preprocessing. We use a trained model, TextBlob to determine the sentiment of the content. The results of that are then outputted as the combined sentiment results data file consisting of the content and its sentiment, and the daily sentiment score consisting of daily aggregated sentiment scores.

Step 3: Trading strategy implementation

Instructions: (Optional, this is automatically set up in step 4)

Run processdatasets.py in order to combine the results from step 1 and 2 for use in step 3. Then run brokerbot if you just want an example you can uncomment the sample driver code we have). Otherwise, you can continue to step 4, since all this step really does is make a tool for step 4 to use.

Implementation Details:

Our trading strategy implementation involved taking the sentiment data from the BERT sentiment analysis step and the changes in stock prices to train a linear regression model for use in step 4. We also need to encode the predicted sentiment values as trading decisions using the defined thresholds, so we had to implement that as well. To normalize the outputs I subtracted 9 from the predicted sentiment and multiplied by 100. This was because the generated sentiment distribution didn't really fit well with threshold values that were recommended, as it mostly averaged around 9 with a very weak slope, leading to the benchmark values of 50/ -50 constantly making the decision to hold.

Step 4: Backtesting Framework

Instructions:

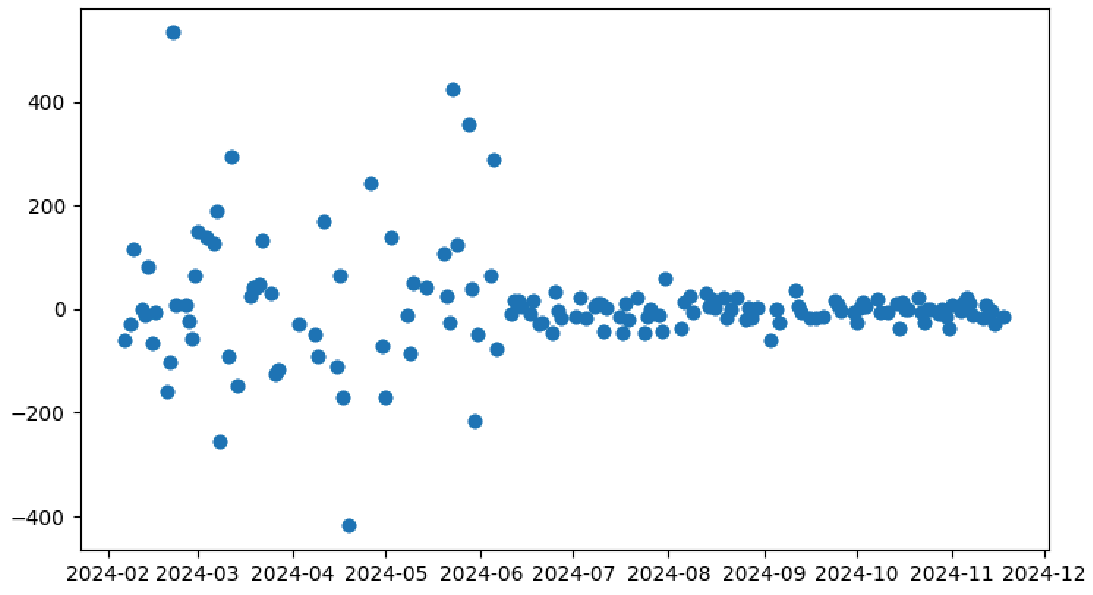
Make sure that all files are not moved from the original locations. Then, run the backtesting_program.py as is. The commission value is initially set to 0.002 and can be altered. As for threshold, you can try 10/ -10, 30/ -30, and 50/ -50 and compare their results.

Implementation Details:

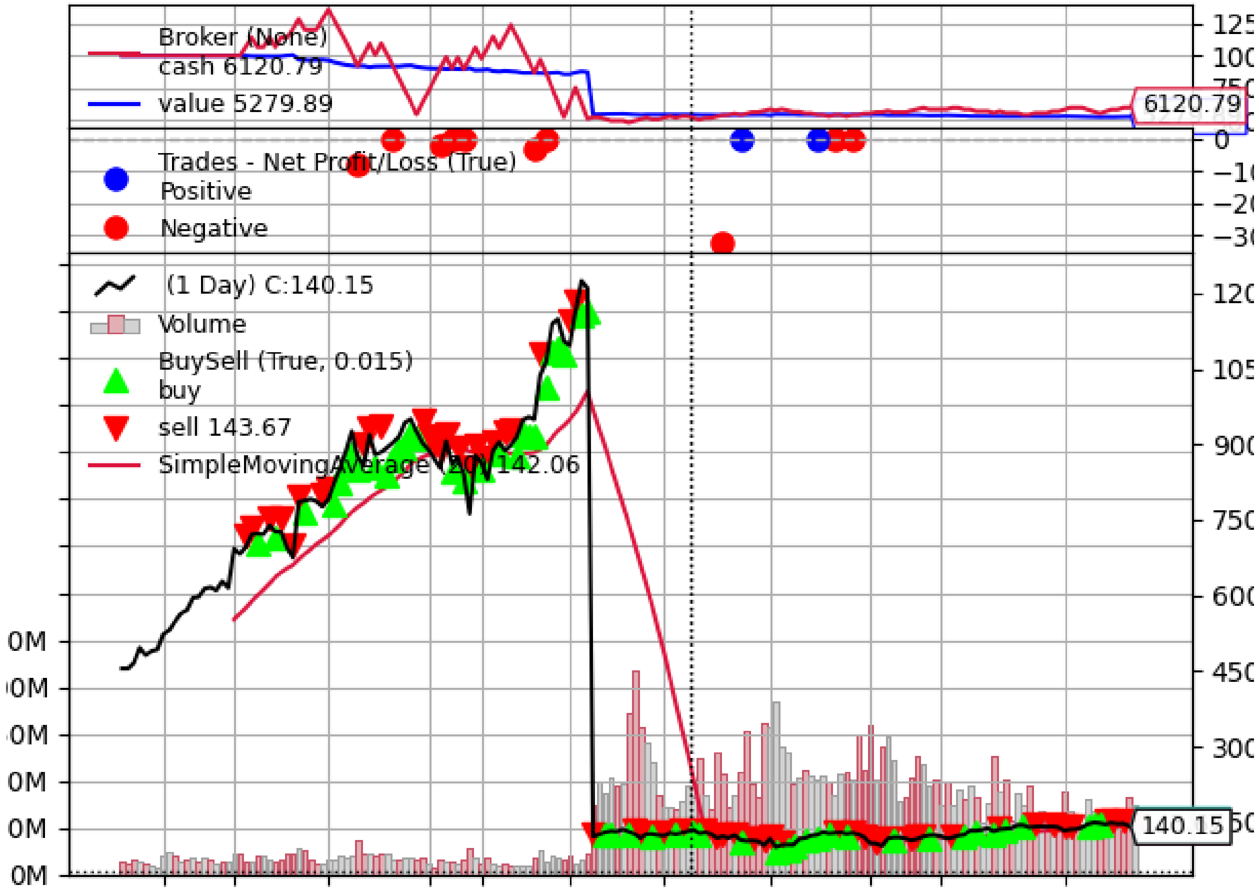
First the sentiment thresholds and initial cash amount for backtesting are prompted. Next, the daily stock prices data file is fetched, read, and renamed for later use. The SentimentDecision module, that is, the code from part3 is imported and trained inside this file. We will use Backtrader as our option of the backtesting framework. It consists of all the necessary functionalities and results for our system. We first define the class SentimentStrategy. It has the Cerebro engine which will take the prices data, class defined, the cash amount, and the commission which is set to 0.002. The Cerebro engine, after taking in parameters, will be set to run and give out the results.

Step 5: Reporting & Visualization

Predicted sentiment scores from Feb 2024 → Nov 2024

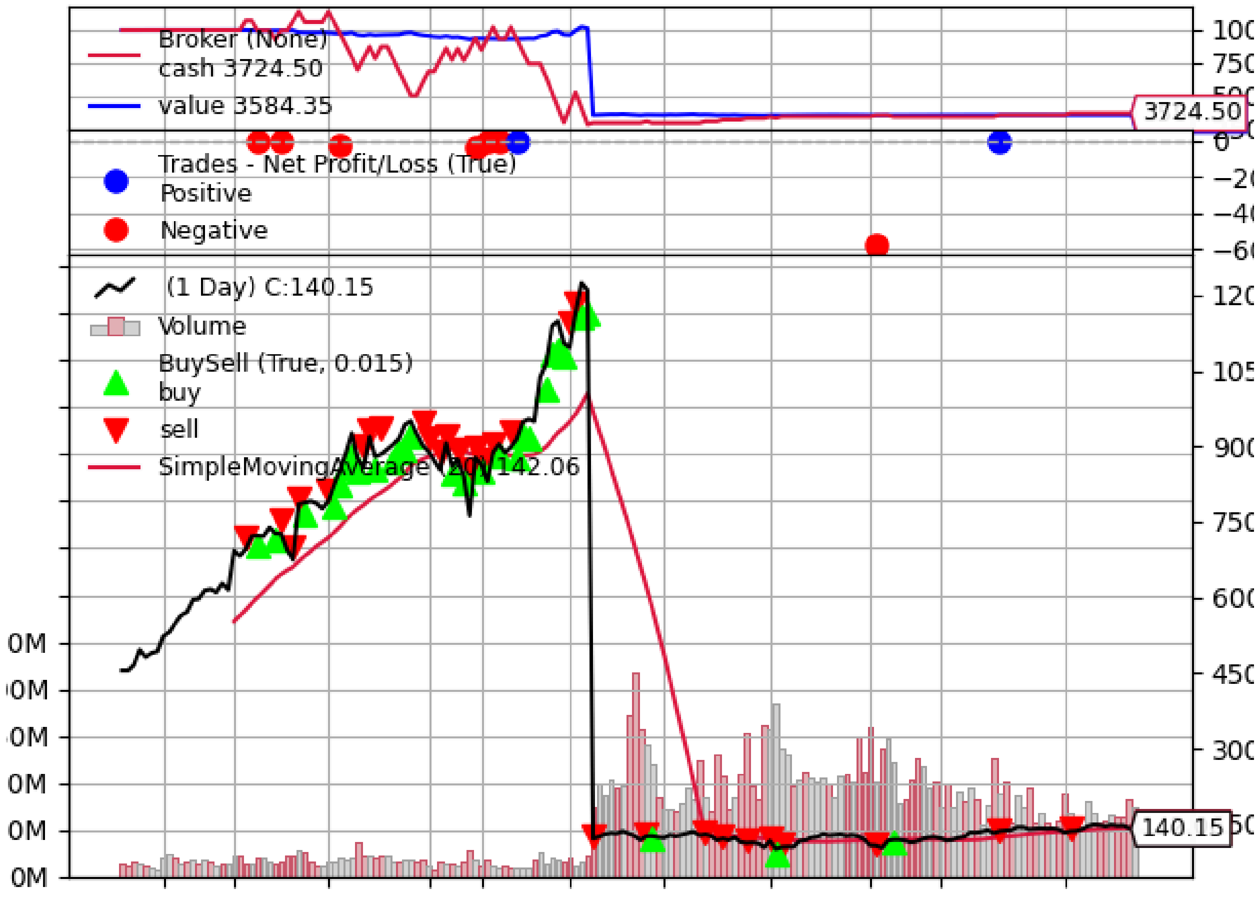


Buy threshold: 10 , Sell threshold: -10, Starting Portfolio Value: $9999.00



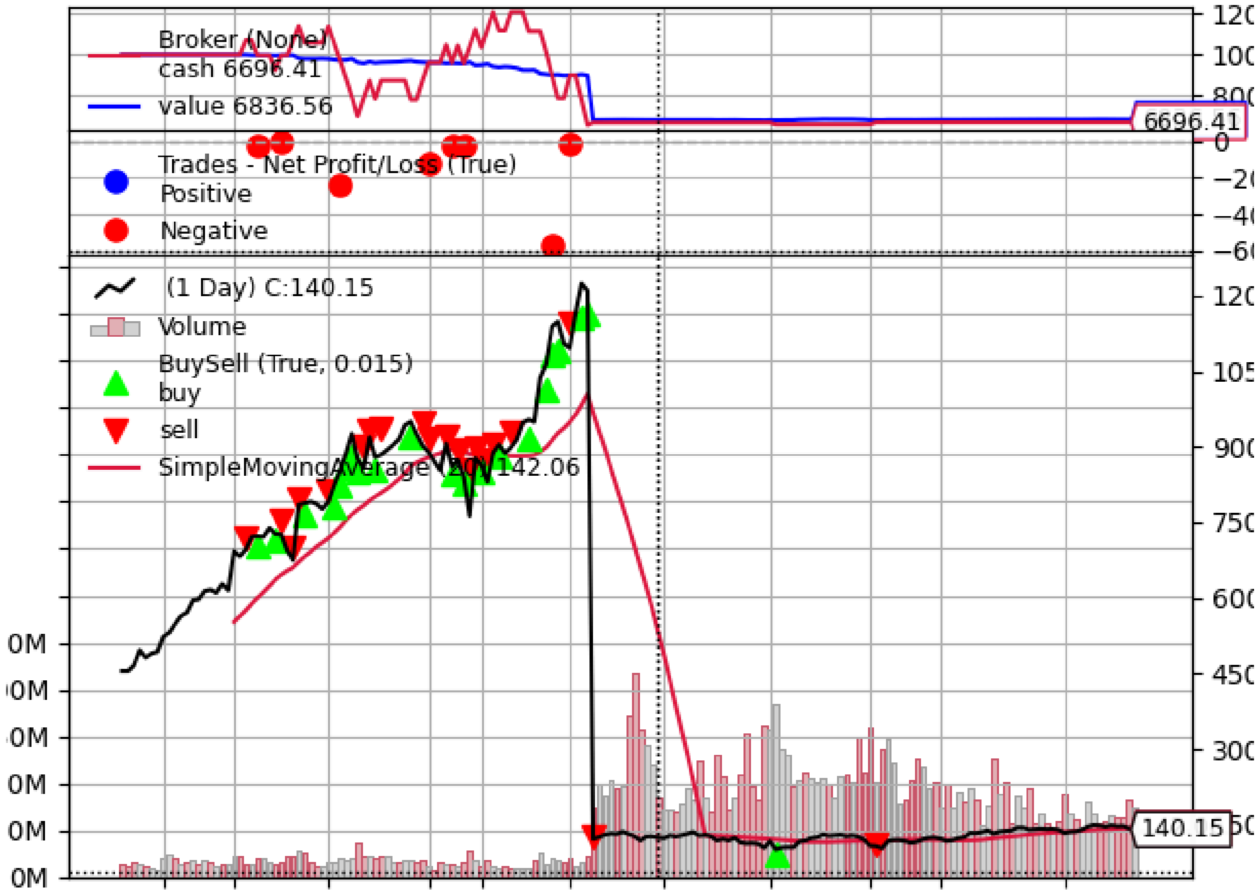
Final Portfolio Value: $5279.89
Return on Investment (ROI): -47.20%
Sharpe Ratio: -1.04
Maximum Drawdown: 47.90%
Win/Loss Ratio: 0.20

Buy threshold: 30 , Sell threshold: -30, Starting Portfolio Value: $9999.00



Final Portfolio Value: $3584.35
Return on Investment (ROI): -64.15%
Sharpe Ratio: -1.03
Maximum Drawdown: 65.28%
Win/Loss Ratio: 0.29

50/-50 threshold, Starting Portfolio Value: $9999.00



Final value: $6836.56
Return on Investment (ROI): -31.63%
Sharpe Ratio: -1.06
Maximum Drawdown: 32.14%
Win/Loss Ratio: 0.00

Conclusions

First off, there were a couple problems with the actual model in that while it did make decisions, it wasn't able to determine how much of a stock to buy exactly. By default our model just buys/sells 1 stock at a time, even though it could potentially have scaled a number of stocks to buy/sell based on the sentiment thresholds. Another issue came with the fact that our model didn't really understand stock splits, and the symbol we chose to investigate just so happened to have a major stock split midway through the dataset. This had a couple implications, one of which was that the model seems to have lost money even though it should have definitely profited. Overall however, it seems that our model does good enough. In general, it buys when the graph starts going up and sells when the graphs start trending down.

Looking at the data, it seems that it was heavily impacted by the nvidia stock split. This includes the predicted sentiment scores from the model. This directly impacted the decision making of the model, 30% and 50% bought a lot more before the dip than it did afterwards.

The actual metrics themselves lean deep in the red due to the stock split concern mentioned above. A negative sharpe ratio indicates that our model severely underperformed its capabilities, and this is corroborated by all the other metrics such as the win loss ratio. The maximum drawdown being incredibly high is not surprising given the stock split tanking the value of the stock all at once.

As for the difference in impact in terms of sentiment thresholds, it seems that having 10% or 50% seems to be optimal, since they had the best returns on investment. This does make sense since 10% seemed more willing to run investments after the split where the line was a lot flatter, and 50% just saved everything after the split occurred instead of mass selling.

References

Packages used: PRAW, Alpha Advantage API, pandas, scikit, matplotlib, dotenv, numpy, TextBlob, mplfinance

https://praw.readthedocs.io/en/stable/getting_started/installation.html
https://github.com/praw-dev/praw
https://www.alphavantage.co/documentation/
https://pandas.pydata.org/docs/
https://scikit-learn.org/stable/
https://matplotlib.org/stable/index.html
https://www.dotenv.org/docs/
https://numpy.org/doc/
https://textblob.readthedocs.io/en/dev/quickstart.html