Jonathan Colaco Carr
MAIS 202 – Project Deliverable

In the final stages of my DNA splicing project, I realized that my dataset was not large enough to get any sort of reasonable prediction accuracy on my model, as it hovered around 84%. Apologies for any inconvenience this causes, but I hope that my new results will please! I will do my best to describe the entire methodology, let me know if you'd like a more detailed description.

Problem Statement

"So you think you can predict the stock market?"

No, I am not proposing a solution to the million dollar question. But who could resist the allure of such a fascinating ML problem? My goal is to predict the short term changes in stock prices based on historical stock data along with sentiment analysis from relevant tweets and news articles (i.e. what's the price of tomorrow's stock?).

Data Preprocessing

For my data, I began with the S&P500 dataset on Kaggle, which had historical prices for the top 500 stocks. I extended my dataset to include news articles by getting access to Yahoo! Finance through Rapid API, and then got relevant tweets with some Twitter API keys. I then used the polarity score metric from TextBlob to convert the Tweets and News articles to either positive or negative scores. The process is described in the flowchart below:

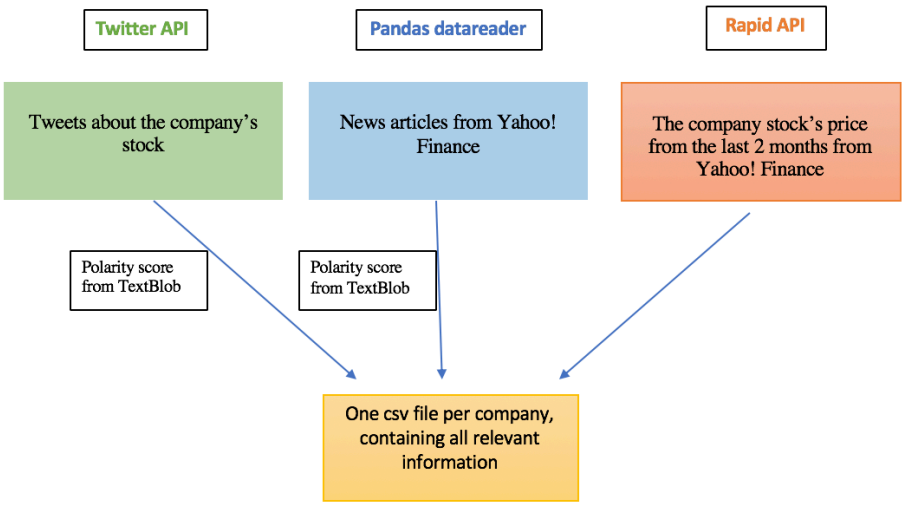

Figure 1: data flow chart

Note: I switched from the Kaggle dataset to pandas datareader in order to receive live data for the project implementation

Here is a peek at Bank of America's data:

Figure 2: sample data

Date	High	Low	Open	Close	Volume	Adj Close	NewsData	TweetScore
2019-03-04	172.35000610351562	169.75	171.99000549316406	170.6300048828125	1663400	170.6300048828125	0	1
2019-03-05	172.69000244140625	156.08999633789062	171.32000732421875	157.25	6699600	157.25	0	-0.5
2019-03-06	166.55999755859375	163.30999755859375	166.52000427246094	164	2534200	164	0	0.6
2019-03-07	163.99000549316406	161.75	163.39999389648438	162.5500030517578	1644300	162.5500030517578	0	-0.375
2019-03-08	162.1699981689453	160.2899932861328	161.58999633789062	162.0399932861328	934700	162.0399932861328	0	-0.05
2019-03-11	164.74000549316406	162.66000366210938	162.7899932861328	164.7100067138672	1064300	164.7100067138672	0	0.3
2019-03-12	167.00999450683594	164.44000244140625	164.8699951171875	166.47999572753906	959500	166.47999572753906	0.5	-0.1
2019-03-13	168.4499969482422	166.66000366210938	166.75	167.08999633789062	816700	167.08999633789062	0.18125	0.1

I created one csv file per company. Before feeding them into my net, I normalized the contents using sklearn's MinMaxScaler, effectively squishing the prices between 0 and 1. This allowed me to combine all the data from the stocks together, and predict changes on general (normalized) scores that could then applied to a particular stock.

Machine Learning Model

From the preprocessing, I prepared 4000 samples, each sample being a 50*6 matrix for the 6 features from the last 50 days. The features for each day was the stock's high, low, open, and closing prices along with its polarity scores for news and tweets. Due to time constraints, I was unfortunately not able to implement my own NLP model for tweet and news article analysis. This remains an area of improvement.

For my model I used a Gated Recurrent Unit. Most amateur stock predictions use and LSTM approach, however I found my best results to be with the GRU. Moreover, the increased efficiency of the GRU allowed me to process more training data. I connected my GRU layer to a fully connected layer that mapped to the output space. My prediction, or labels, were the following day's high, open, and low prices.

Results

I was thrilled with my model's results. Below is a table comparing my error to "the best on Kaggle"[1] from Rohit Verma. Verma trained his network on the four daily prices that I did. With the added polarity scores for news and tweets, my model came out on top:

	Best Kaggle	This model
Train MSE	1.91×10^{-4}	4.02×10^{-7}
Test MSE	3.28×10^{-4}	1.08×10^{-6}

Table 1: Mean Squared Error on normalized data

Daily high price	Daily low price	Daily closing price
8.74×10^{-7}	1.40×10^{-6}	8.5×10^{-7}

Table 2: Mean Squared Error on normalized test data

Verma, Rohit. New York Stock Exchange Predictions RNN LSTM https://github.com/deadskull7/New-York-Stock-Exchange-Predictions-RNN-LSTM

```
(epoch, train_loss, val_loss) = (1934, 3.8701338780811054e-07, 4.700596226333194e-07)
(epoch, train_loss, val_loss) = (1936, 5.047591986340194e-07, 8.160517950273061e-07)
(epoch, train_loss, val_loss) = (1938, 5.184654440881787e-07, 7.893897209972541e-07)
(epoch, train_loss, val_loss) = (1940, 4.2273726876373983e-07, 4.915590778864498e-07)
(epoch, train_loss, val_loss) = (1942, 4.2590826694777205e-07, 4.866284513127539e-07)
(epoch, train_loss, val_loss) = (1944, 3.765192900573311e-07, 4.1894932678587793e-07)
(epoch, train_loss, val_loss) = (1946, 4.67860146500243e-07, 5.576925824849847e-07)
(epoch, train_loss, val_loss) = (1948, 4.4047002631941725e-07, 5.37755686499016e-07)
(epoch, train_loss, val_loss) = (1950, 4.2079149068285917e-07, 4.886970496424207e-07)
(epoch, train_loss, val_loss) = (1952, 3.909472320628993e-07, 4.2366842478713806e-07)
(epoch, train_loss, val_loss) = (1954, 4.723235792880587e-07, 6.02575080203375e-07)
(epoch, train_loss, val_loss) = (1956, 4.047295101372583e-07, 5.510410450900357e-07)
(epoch, train_loss, val_loss) = (1958, 4.7180899741761096e-07, 4.018919668169474e-07)
(epoch, train_loss, val_loss) = (1960, 3.960693689464279e-07, 4.172863251975893e-07)
(epoch, train_loss, val_loss) = (1962, 4.468452931405409e-07, 5.949340315207033e-07)
(epoch, train_loss, val_loss) = (1964, 3.889542335855367e-07, 4.077924747510527e-07)
(epoch, train_loss, val_loss) = (1966, 4.3135560758855716e-07, 4.099195318000663e-07)
(epoch, train_loss, val_loss) = (1968, 4.499726503581769e-07, 7.433859726309796e-07)
(epoch, train_loss, val_loss) = (1970, 5.342656544371493e-07, 5.87522445509118e-07)
(epoch, train_loss, val_loss) = (1972, 7.611474273971908e-07, 7.026621915429132e-07)
(epoch, train_loss, val_loss) = (1974, 4.456182568901568e-07, 5.892482496013448e-07)
(epoch, train_loss, val_loss) = (1976, 5.054431443340945e-07, 5.052203041107836e-07)
(epoch, train_loss, val_loss) = (1978, 4.5166143195274347e-07, 4.146910868030318e-07)
(epoch, train_loss, val_loss) = (1980, 3.992755085846511e-07, 4.1202195196395525e-07)
(epoch, train_loss, val_loss) = (1982, 3.7161537932206555e-07, 3.452191113713828e-07)
(epoch, train_loss, val_loss) = (1984, 4.435101277522335e-07, 4.796970548189469e-07)
(epoch, train_loss, val_loss) = (1986, 4.7538434643001894e-07, 4.475652417568199e-07)
(epoch, train_loss, val_loss) = (1988, 4.659833734876884e-07, 8.294052899297336e-07)
(epoch, train_loss, val_loss) = (1990, 3.959459894531392e-07, 4.0752453382234915e-07)
(epoch, train_loss, val_loss) = (1992, 3.8803649403007514e-07, 4.2968252955688513e-07)
(epoch, train_loss, val_loss) = (1994, 3.492573458174775e-07, 3.5802257987901004e-07)
(epoch, train_loss, val_loss) = (1996, 4.287738397579233e-07, 4.86117277394745e-07)
(epoch, train_loss, val_loss) = (1998, 4.110711347493634e-07, 4.018040774174854e-07)
```

Figure 3: Training and Validation Error during training

Final Demonstration Proposal

I would like to create a simple landing page website that will allow a user to input a stock name, then generate some predictions along with some nice graphs and trends. I know HTML and CSS well enough, and I will attend the Flask workshop next week. The backend does not seem terribly difficult and I am able to automate the procedure from the command line. Below is a simple flow chart of how I have run the program from the command line, all of the referenced files can be found in my repo's source folder: