Stock Price Analysis Using Twitter Sentiment with Lambda

ECE 4813
December 1st, 2018

Peter Fechter
Marites Hendrix
Cameron Pepe
Noah Pilz

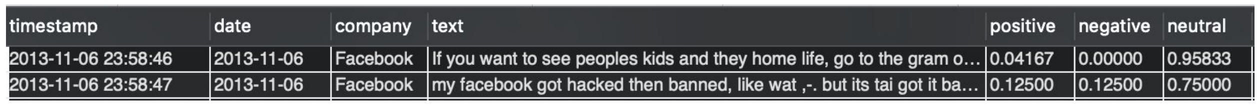
I. Introduction

Widespread data availability and flexibility through social media and user friendly APIs is one of the highlights of living in the information age. In the context of corporate advertising, there exists an age old adage that “Any press is good press”. While quantitative analysis of a financial transactions and balance statements is the primary method of valuing a company’s stock, it is probable that public sentiment towards a company also plays a large role in its current and future success [1]. This project explores this idea.

In an effort to further understand the impact of public opinion and how it affects the stock price of a company, this project uses sentiment analysis of Twitter posts to make stock-purchase suggestions for several consumer facing corporations (e.g. Nike, Facebook, etc.). The service makes suggestions on three different time horizons for the stock purchase: three months, one week, and three days in the future.

II. Data Sources

For this project, the primary method for determining public opinion was through analysis of Twitter data containing the name of a company and/or its respective hashtag. The tweets were collected from Twitter using a 3rd-party Python library, *GetOldTweets-python* [2], which performed the appropriate Twitter web queries and web-scraped the response. This library provided the ability to query tweets programmatically more than seven days in the past, a functionality that the free tier of Twitter Developer did not support. To build the training data set for the machine learning, tweets were gathered and stored from the last five years. Each time a stock purchase recommendation is requested, the database is updated with the new tweets since the previous request. For each company, 500 tweets were pulled per day. Due to the large amount of network bandwidth this operation consumed, AWS throttled the network speed and request frequency for requests of over 500 tweets per day. On average, around 100 tweets per day were ultimately stored and used after filtering of non-English tweets and tweets containing promotional URLs.

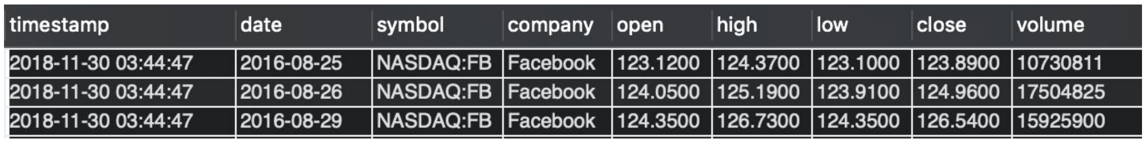


timestamp	date	company	text	positive	negative	neutral
2013-11-06 23:58:46	2013-11-06	Facebook	If you want to see peoples kids and they home life, go to the gram o…	0.04167	0.00000	0.95833
2013-11-06 23:58:47	2013-11-06	Facebook	my facebook got hacked then banned, like wat ,-. but its tai got it ba…	0.12500	0.12500	0.75000

Figure 1. Example of stored tweets and format

The stock data is retrieved through AlphaVantage, a company that provides updated stock information with granular detail. The AlphaVantage API is free, fast, and flexible, offering up to

500 requests per day. Once the stock data for a certain day was pulled for a company, it was stored in the database so no further requests were needed. The trading volume and high, low, open, and closing stock prices were retrieved for each day for each company. Because the stock prices were only needed to train the machine learning model, stock prices were only pulled for the same five year window as the respective tweets for each company.



timestamp	date	symbol	company	open	high	low	close	volume
2018-11-30 03:44:47	2016-08-25	NASDAQ:FB	Facebook	123.1200	124.3700	123.1000	123.8900	10730811
2018-11-30 03:44:47	2016-08-26	NASDAQ:FB	Facebook	124.0500	125.1900	123.9100	124.9600	17504825
2018-11-30 03:44:47	2016-08-29	NASDAQ:FB	Facebook	124.3500	126.7300	124.3500	126.5400	15925900

Figure 2. Example of stored stock data and format

III. Project Architecture

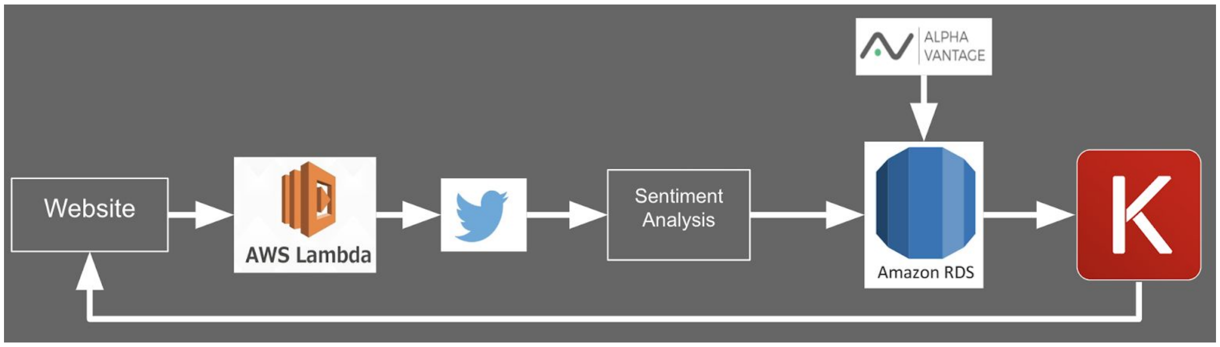


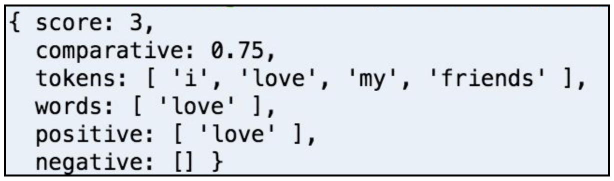
Figure 3. Overview of the connected architecture

When the user requests a stock suggestion for a certain company, it publishes a notification, via AWS SNS, to the tweet-pulling Lambda function. The Lambda function queries Twitter for the tweets containing the company's name within a specified date range, performs sentiment analysis on each tweet, and stores the tweets in the MySQL database hosted on AWS RDS. Once the tweets and their corresponding sentiments are stored, the machine learning model is queried to make a suggestion on whether the user should or should not purchase stock for that company based off of the twitter sentiment. For the machine learning training, the stock prices for each company was pre-loaded into the database.

AWS Lambda is ideal for this project's objective because of its automatic scaling. If ten different users request stock suggestions for ten different companies at the same time, ten different instances of the tweet-pulling Lambda function will be automatically spun up and operate independently.

IV. Sentiment Analysis

The sentiment analysis was performed with the *sentiment* Node.js package, using AFINN-based sentiment analysis [3]. This tool tokenizes each word within a tweet and assigns a numerical rating to each word ranging from -3 to 3. These values were then summed to generate an overall sentiment score and comparative score specifying the degree to which a tweet was positive or negative. The output of the tool is shown in Figure 4 below.



```
{ score: 3,
  comparative: 0.75,
  tokens: [ 'i', 'love', 'my', 'friends' ],
  words: [ 'love' ],
  positive: [ 'love' ],
  negative: [] }
```

Figure 4. Example of sentiment and classification of score words

The output from this tool was then used to create a positive, negative, and neutral probability distribution for the tweet. Using the AFINN labels of the positive and negative words and the total number of words in the tweet, a probability distribution was created in order to develop a more expressive representation of the sentiment.

In order to properly process tweets, the sentiment analysis tool required the text to be English. To filter out the non-English tweets, Google's Python *language-detection* package was used [4].

Although free, this sentiment analysis tool is rudimentary and future work on this project could be focused on improving the accuracy of the analysis. In the original project architecture, sentiment analysis was performed through the use of Amazon Web Services Comprehend, a scalable natural language processing toolkit. Sentiment analysis provided by Amazon continually improves upon itself, which is a key distinguishing factor when selecting an analysis engine [5]. This tool provided more detailed and more accurate sentiment analysis, however was very expensive.

V. Machine Learning Prediction Model

To find the correlation between twitter sentiment and whether the stock is a good purchase was done via a neural network with binary classification. The model predicts whether the stock price *tended* to increase or decrease over the future time horizon and is not intended to predict the stock price. Three neural networks were generated for each company, one for each time horizon.

The number of corresponding preceding days of twitter sentiment used for each time horizon can be found in Table 1 below.

Stock Time Horizon (Future)	**Number of Days of Twitter Sentiment Used (Past)**
60 Trading Days (long-term)	180 days
7 Trading Days(mid-term)	30 days
3 Trading Days (short-term)	7 days

Table 1. Stock Time Horizon and Number of Preceding Days of Twitter Sentiment Used

The input features to the machine learning models are an average of the twitter sentiment for each day. For each day analyzed, the positive, negative, and neutral sentiment confidence values were summed and divided by the number of tweets analyzed for that day. This resulted in an input vector of three times the number of preceding days analyzed (table above).

The corresponding labels of the neural network were a 1 (recommended buy) or a 0 (recommended don't-buy). The labels were determined by finding a "stock quality score" over the future time horizon; if the score is non-negative, then the label is 1, if the score is negative, then the label is 0. The stock quality score is determined via the following equation:

$$\%\Delta_i = \left(\frac{(price_{close,\, i} - price_{high,\, i})}{2} - \frac{(price_{close,\, i-1} - price_{high,\, i-1})}{2} \right) / \frac{(price_{close,\, i-1} - price_{high,\, i-1})}{2}$$

$$score = \sum_{i=0}^{time\ horizon} \sqrt{i+1} * (\%\Delta_i)$$

The percent change was found for the averaged high and closing price for each day and its previous day. The quality score is then calculated as the sum of weighted stock price changes for the days in the time horizon. The weight is the square root of how many days in the future that price change corresponds to. This weight is applied so that the distant future is of higher importance than the near future, assuming that the user of this application is intending to hold stocks for a duration near the length of the time horizon. The quality score is calculated as such in order to smooth out the volatility of stock prices. If the label was only dependent on the change between today and the last day of the time horizon, the stock price performance information between the two days would be lost and the quality score would be dependent on a single day and subject to that day's volatility.

A neural network is uses to solve this binary classification problem. The neural network has two hidden layers and a single output perceptron. The hidden layers and input layer used the ReLU activation function and the output layer used a sigmoid activation function to incorporate probabilities (confidence levels) into the binary classification. The input and hidden layers have $3 * \#_{preceding\ twitter\ days\ analyzed}$ perceptrons: the averaged positive, negative, and neutral sentiment for each day in order from furthest in the past to yesterday. This information is represented visually in Figure 5 below.

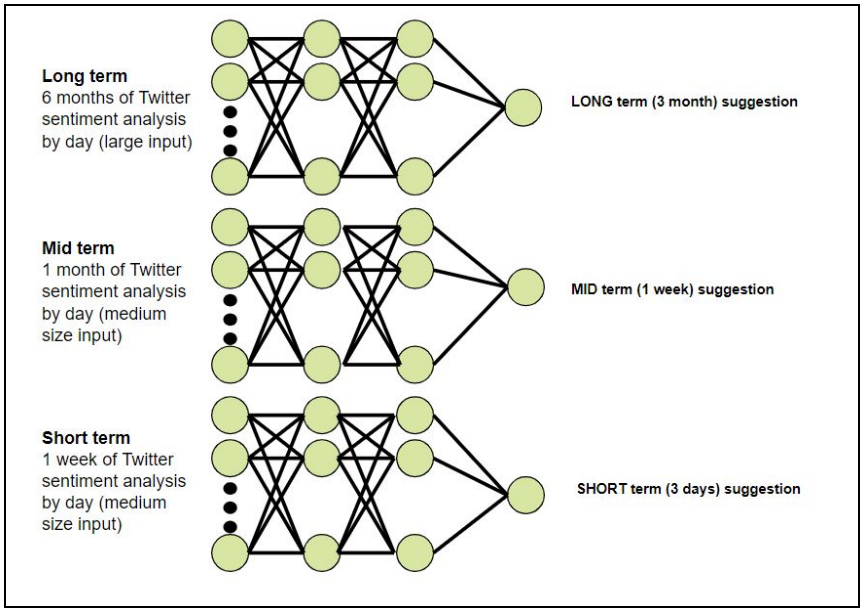

Long term
6 months of Twitter sentiment analysis by day (large input)
LONG term (3 month) suggestion
Mid term
1 month of Twitter sentiment analysis by day (medium size input)
MID term (1 week) suggestion
Short term
1 week of Twitter sentiment analysis by day (medium size input)
SHORT term (3 days) suggestion

Figure 5. Layout of neural network of nodes

Table 2 below shows the testing accuracy of the machine learning model. The model had much higher classification accuracy for the long-term prediction than is does for the short-term prediction. A random guess would have 50% accuracy. These results show that there is a strong correlation between the twitter sentiment from the last six months and the stock price performance of the future three months of trading days, especially for Facebook. For the short term there is little-to-no correlation between twitter sentiment and the stock price performance.

Time Horizon	**Nike**	**Facebook**
Long (60 Days)	69%	79%
Mid (7 Days)	57%	62%
Short (3 Days)	52%	58%

Table 2. Test Accuracy for Nike and Facebook for the Different Time Horizons

Though there are correlations, the high test accuracy may be due to the bullish market of the last five years biasing the training data set. This bias can be assumed true because around 75% of the classification labels were a 1 (“buy”), meaning the stock prices continued to increase over the duration of the samples gathered for the training data set.

VI. Website

The website was created with a Python Flask backend and a frontend that utilized Bootstrap, AngularJS, HTML/CSS and CSS animations. The Flask backend was specifically used to serve up the frontend pages and images, as well as create a point where the machine learning functionality could connect to the user.

The website navigated between two pages main pages: the home page and the results page. Using AngularJS ngRoute functionality, the two pages in which users saw smoothly transitioned from one to another. This is because ngRoute serves up all static objects at the initial startup of the web application, so that when you switch between pages, there is no reloading or waiting on the server for a new page. Instead, index.html serves as the main page and as a user interacts, different HTML is injected into it.

The Angular home page controller, named “HomeController”, played the role of listening to which company the user selected. When the user selects a company, the HomeController would pass on the company name to the result page controller with a POST request, and then send a json object with the company name to the Python Flask backend. The backend would then send the company name to a Lambda function to pull the most recent tweets, if necessary, and then to the machine learning backend for the prediction and wait for a response.

Once the response is received from the machine learning model, the page switches to the result page and the result controller, named appropriately “ResultsController”. The result controller calls the Python Flask backend with a GET request to obtain the machine learning results and then parses through the results, setting scope parameters for the web page to display.

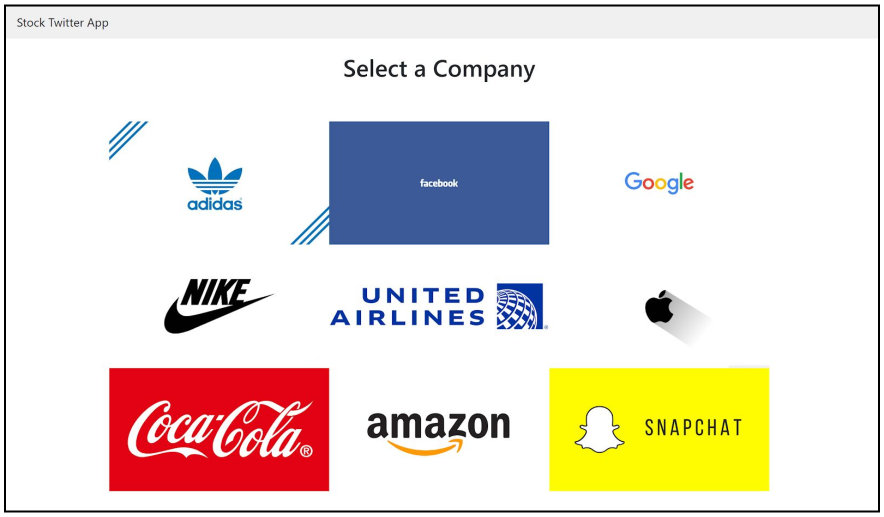
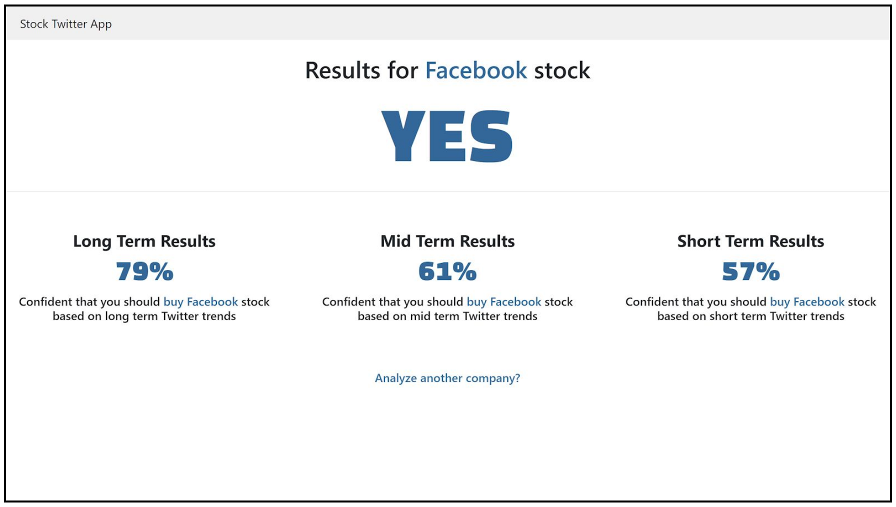


Figure 6. Home page for web application

The index.html page only has the navbar. On load, the home.html page is injected by Angular and displays a 3x3 grid of 9 companies to choose from. On hover, each section zooms in and describes the company. A user can click anywhere on any box to see results.



Results for Facebook stock

YES

Long Term Results	Mid Term Results	Short Term Results
79%	61%	57%
Confident that you should buy Facebook stock based on long term Twitter trends	Confident that you should buy Facebook stock based on mid term Twitter trends	Confident that you should buy Facebook stock based on short term Twitter trends

Figure 7. Result page of web application

Once the machine learning analysis is complete, the results page displays. The title and results change dynamically in real-time based on the company and results returned. First, the title fades in, followed by an animated slam of the results on whether to buy stock or not. The “YES” in this photo is due to the fact that at least two or more of the results have returned a yes to buy stocks. If only one result returned as yes, it would display “Maybe?” and if neither long, mid, or short term results say to buy then it would say “NO”.

Afterwards, the long term, mid term and short term results text fade in cascaded in that order, followed by a cascaded slam of each percentage confidence in those results. At the bottom “Analyze another company?” fades in about 5 seconds after page load, and continues to fade in and out slowly. If the user clicks that or the “Stock Twitter App” on the top left of the page, it will redirect them to the home page to try another company.

VII. Conclusion

This project accomplished the goal of providing the user a suggestion on whether to buy stock in a certain company by analyzing tweets. Our three-tiered suggestion approach also helps the user judge whether or not to invest in a certain company based off their frequency and style of stock trading.

However, our offering could be vastly improved by not needing to rely on free tools. Amazon Comprehend provided the best sentiment analysis, but at a great cost as well. Paying for full Twitter API access would have given more data to train the machine learning models, likely resulting in an increased confidence in purchasing suggestion.

All code for the project can be found in the following GitHub repository:
https://github.com/cameronpepe/TwitterSentimentStockPurchaseSuggestion

VIII. Contributions

Cameron Pepe	AWS Setup & Devops Management AWS Lambda Structure Setup Lambda Code Refactoring Machine Learning and Data Handling
Noah Pilz	Stock Price API Integration Stock Data Pulling
Marites Hendrix	Twitter Sentiment API Integration Tweet Pulling Lambda Function Website
Peter Fechter	AWS Comprehend Component Invocation

	Language Detection Invocation

IX. References

[1] F. Daréna and J. Petrovsky. "Machine Learning-Based Analysis of the Association between Online Texts and Stock Price Movements," *Inteligencia Artificial,* vol. 21, no. 6, June 2018. [Accessed November 30, 2018].

[2] J. Henrique. "Get Old Tweets Programmatically," *GitHub.* Available: https://github.com/Jefferson-Henrique/GetOldTweets-python

[3] Thisandagain. "Sentiment," *NPMjs.* Available: https://www.npmjs.com/package/sentiment

[4] S. Nakatani. "Language Detection," *Google Code Library.* Available: https://code.google.com/archive/p/language-detection/

[5] C. Tao and X. Ruifeng. "Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN," *Expert Systems with Applications,* vol. 72, April 2017. [Accessed November 30, 2018].