Project Report

Stock Price Prediction By Tweet Sentiment Analysis

Background and Description

A company's market values are variable and affected by different factors, the most important of which could be the thoughts of the public. We try to analyze the effect of public opinion about a company on that company's market values.
We are expecting a correlation rate that clarifies the public opinion vs. market values. Sentiment analysis of the Tweets related with certain companies with a time series in a graph might help us with reasoning the possible declines and rises in their stock prices, which also enables making predictions about changes in stock prices as well as providing investors with buy or sell suggestions.

To address the non-linearity and highly complex nature of the underlying process in stock price modeling, we trained a Long-Short-Term-Memory (LSTM) model as our recurrent neural network, which helps to mitigate the issue of exploding or vanishing gradients. To gain better results for tweets' sentiment classification, we also employed BERT (Bidirectional Encoder Representations from Transformers), a transformer-architecture language model for performing various downstream natural language processing tasks. It is pre-trained from an enormous unlabeled text corpus that encompasses Wikipedia and a comprehensive book corpus and has a deeper sense of language context because of its ability to learn the contextual information bidirectionally.

Data

We use a dataset available on Kaggle which was originally a part of a paper[1] published in the 2020 IEEE International Conference on Big Data and Tweets are collected from Twitter by a parsing script that is based on Selenium.
This dataset contains over 3 million unique tweets with their information such as tweet id, author of the tweet, post date, the text body of the tweet, and the number of comments, likes, and retweets of tweets matched with the related company.

For historical stock prices with granularity to 30-minute or 1-hour intervals, because Yahoo's yfinance API limits such queries with finer granularity to the past six months, we found a free tool without paying for other commercial APIs, from which we downloaded NASDAQ values containing OPEN, CLOSE, VOLUME, HIGH, and LOW values for given time intervals of Amazon, Apple, Google, Microsoft, and Tesla.

Architecture and Components

[1] M. Doğan, Ö. Metin, E. Tek, S. Yumuşak and K. Öztoprak, "Speculator and Influencer Evaluation in Stock Market by Using Social Media," 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 4559-4566, doi: 10.1109/BigData50022.2020.9378170.

Software and Libraries

1. Dash as the web UI rendering framework
2. Natural Language Processing analysis:
    a. NLTK library for text preprocessing (e.g. word lemmatization)
    b. Vader/TextBlob/Flair libraries for general-purpose sentiment analysis
    c. Using BERT model for unsupervised sentiment analysis:
        i. Python psentimentio package
3. Tensorflow to train LSTM recurrent neural networks for time series prediction

Machine Learning

In our dataset from Kaggle, the scraped tweets are primarily those with respective cashtag ticker symbols. The tweets are assigned to the stock prices via a moving window approach. After every 30 minutes or 60 minutes we receive a new close price. The tweets of this time interval are used for the calculation of the Twitter features. As such, we lose tweets on the weekend or during the night.

Trimming down the range of dates from 2019-10-01 to 2019-12-31 helps to speed up processes with finer granularity of time such as 30-minute or 1-hour window (rendering around 40,000 tweets for each company).

Sentimental analysis of Tweets (unsupervised labeling)

We started out using TextBlob's sentiment analysis module, which is based on naive Bayes and calculates average polarity over each word using a dictionary of adjectives and their hand-tagged scores. This and Vader use the same kind of approach and achieve an accuracy of ~60%.

Better accuracy (around 70%) of sentiment classification was reached when we used BERT model specifically fine-tuned on Twitter data, and we substituted the original TextBlob's scores. The implementation is available as a Python package pysentimiento, which also features a tweet preprocessor specially suited for tweet classification that handles URLs, hashtags and emojis. Nevertheless, the BERT approach takes much longer than dictionary-based methods to classify the sentimental tendencies (over an hour was consumed to label all tweets for a given company). Given the nature of pre-trained BERT models, even the smallest ones among them have millions of parameters to fine-tune.

Features extracted from tweets

Apart from the sentimental labeling, we also added content unrelated variables such as frequency and the average amount of characters of released tweets within a 30-minute or 1-hour window.

LSTM

At first we put all five years of single-day stock prices into the baseline LSTM model with a lookback window of 60 days. This already took too much computing resource for training even on daily prices due to the fact that LSTMs are inherently linear and unable to fully utilize parallel computation. For 30-minute or 60-minute window the situation would be much worse. We hence tried to limit the size of our training sample and reduced the lookback window size to 5.

We used random search for optimizing models' hyperparameters. The first LSTM cell can have recurrent units from 17 to 500, relu, tanh, or sigmoid as activation function. For the number of hidden LSTM layers we defined a range between 0–3 where each layer has the possibility of having between 17 to 250 recurrent units. If the random search specifies a model with hidden layers, there is also a hidden dropout per layer with a minimum dropout of 5% and maximum of 95%. Early stopping in training was also added to stop the training once there's no more improvement in fitting. 50 maximal trials and 3 executions per trial were carried out.
The best performing hyperparameters for our LSTM models are as follows with a batch size of 32 and Adam as the optimizer.

Hyper-parameters for one of the best-performing models

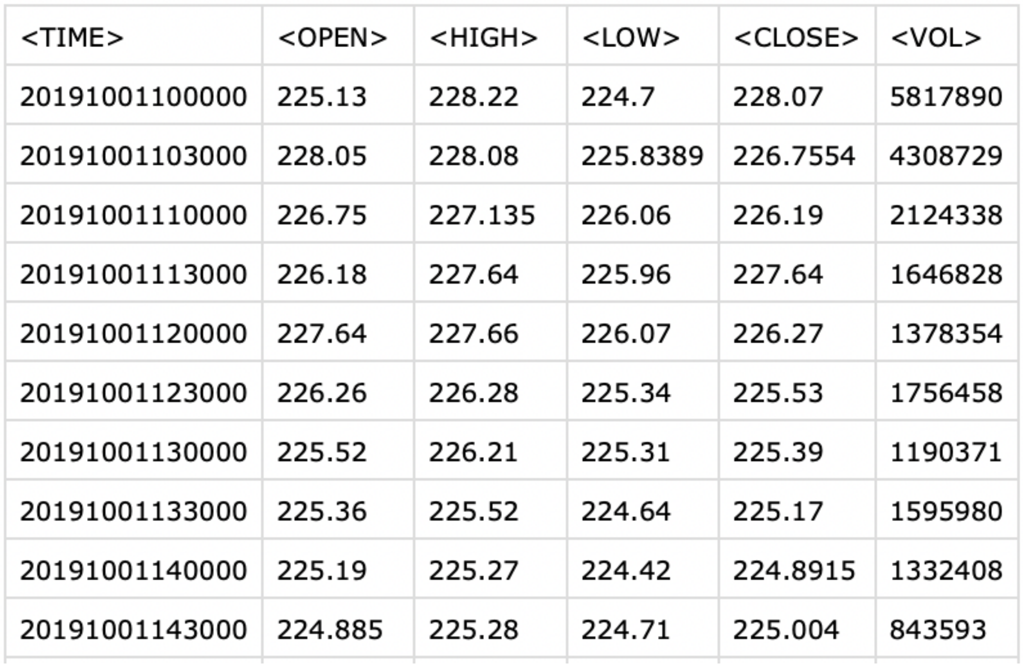
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 217)	192696
dropout (Dropout)	(None, 217)	0
dense (Dense)	(None, 1)	218

Total params: 192,914 Trainable params: 192,914 Non-trainable params: 0

We started with our baseline model which is an LSTM model with lagged close price, open price, high price, and low price of past five time points in the time window fed into the system to predict the next stock value. Proceeding, we combine these baseline features with different combinations of features derived from sentimental analysis to improve accuracy of our system. After adding in frequency and average amount of characters of released tweets, as well as ratio of positive and negative tweets, we managed to approximately half the Mean Squared Error.

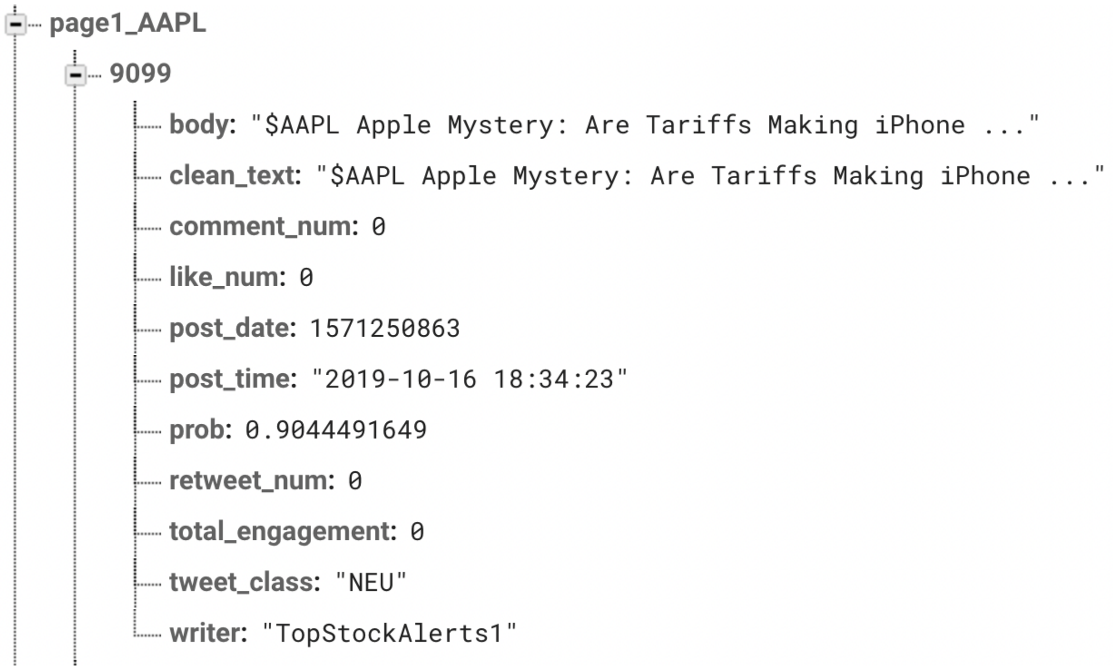
Database

We use Google Firebase Real-Time Database for our application. Each time users try to access or save inference results, we request or write data to Firebase. (We tried using DynamoDB NoSQL database but the API calls consumed too much time to upload the entire dataset.)



<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>	<VOL>
20191001100000	225.13	228.22	224.7	228.07	5817890
20191001103000	228.05	228.08	225.8389	226.7554	4308729
20191001110000	226.75	227.135	226.06	226.19	2124338
20191001113000	226.18	227.64	225.96	227.64	1646828
20191001120000	227.64	227.66	226.07	226.27	1378354
20191001123000	226.26	226.28	225.34	225.53	1756458
20191001130000	225.52	226.21	225.31	225.39	1190371
20191001133000	225.36	225.52	224.64	225.17	1595980
20191001140000	225.19	225.27	224.42	224.8915	1332408
20191001143000	224.885	225.28	224.71	225.004	843593

An sample dataset for $AAPL’s stock prices with 30m intervals



```
page1_AAPL
    9099
        body: "$AAPL Apple Mystery: Are Tariffs Making iPhone ..."
        clean_text: "$AAPL Apple Mystery: Are Tariffs Making iPhone ..."
        comment_num: 0
        like_num: 0
        post_date: 1571250863
        post_time: "2019-10-16 18:34:23"
        prob: 0.9044491649
        retweet_num: 0
        total_engagement: 0
        tweet_class: "NEU"
        writer: "TopStockAlerts1"
```

Sample data for $AAPL’s tweets with tweets’ sentiments classified uploaded to Firebase

Functions and User Interface

The web application’s UI consists of two tabs, one for exploration of historical records, and one for displaying and storing inference results of stock prices and buy/sell suggestions.

Exploration

The system has an intuitive interface for users to explore all the tweets stored in the database as well as relevant tweets to certain companies which users can choose from. Users are able to see the total number of tweets given a certain date, and for each tweet its body, author, post date and time, as well as the number of total engagement, retweets, likes, and comments. Users can also select positive or negative tweets on each company, and filter the tweets by time range.

On the second tab, after selecting a time horizon (30-minute or 60-minute), users are able to see frequency and average amount of characters of released tweets within selected time horizons for a company, as well as number and share of positive and negative tweets.

Furthermore, users are able to upload their own Tweets data and stock price data in .csv format and such data with the above-mentioned metadata and features be presented in the same way for users to explore, which would also be stored in the database.

Prediction/Inferences

On the second tab, the system is able to:

- predict the fluctuation in stock price within 30 minutes or 60 minutes for a selected company;
  (presenting the alignment of historical data with past predictions)
- give a buy or sell suggestion for a given time point;
- store the inference results of a given time in the database

Learning experiences

Lessons and Challenges

Problems encountered with data collection

Primarily the biggest roadblock for this project is the difficulty with data acquisition of Tweets, which is out of our expectation.
Among all the official and third-party APIs for getting Tweet data, we tried various ones including but not limited to Tweepy, GetOldTweets, Twint, and TwitterAPI. What we have found is that Twitter has recently decided to restrict its advanced search to tweets posted in the latest six months. Web-scraper based ones such as Scweet use BeautiSoup on frontend web pages and currently do not support pagination for keyword queries, so we only get three to four tweets per day, which is far not enough to serve our goal. Accordingly, in order to utilize a larger amount of data for model training, we use the dataset available on Kaggle which contains tweets from the year 2015 to 2020 collected from Twitter by a parsing script based on Selenium. As a result, our analysis and application is limited to the six major tech companies included in the dataset, but this should be a good starting point for our explorational project. The restrictions imposed by

Twitter have disabled us from letting users check out historical data that is outside the range of our prefetched dataset.
We also managed to overcome the difficulties of getting historical stock prices with granularity to 30-minute or 1-hour intervals. Yahoo's yfinance API limits such queries with finer granularity to the past six months, so we found a free tool without paying for other commercial APIs.

Experience and skills gained

The most notable takeaway for me implementing this project was that I learned to modularize and organize my project code files in a more coherent and compact way. Previously when I was working on web applications developed by Python-based frameworks, I was always reluctant to split up my code or reuse utility functions. While going through this relatively complex single-page application that has multiple tabs, I had to restructure my code and divide up different components into reusable chunks for easier debugging.

Link to Github repository:
https://github.com/siwei-li/tweet_stock
(.csv files are omitted in .gitignore due to file size limitation of Github upload)