Eli Jaghab and Keerthi Seetha

Dr. DeBenedetto

CSC 9010: Natural Language Processing

May 2022

Stock Price Prediction using Sentiment Analysis

Abstract

While the stock market is a volatile place, many changes can be attributed to public textual data sources. These sources range from newspaper articles, to articles online, to speeches by eminent people, to quarterly earnings reports, to social media sources such as Tweets from Twitter to discussions on Reddit.

Summary

Goal

The goal of our project is to be able to inform an investor on whether or not they should invest in Apple or Tesla stock based on the previous sentiment of text data from a variety of sources. We ingested textual and price data from a variety of sources to come up with a model to decide our predictions.

Inputs

The inputs of our project involve articles from *The Motley Fool* and *Market Watch*, Tweets from Twitter, comments from Reddit's thread, *r/wallstreetbets*, and price data from Alpaca. We decided to cast a wide net regarding sources ranging from social to financial news to identify which would contribute the most towards an accurate prediction.

Outputs

We built multiple classification and regression models. The classification model will be able to predict the movement of the stock price (predict 0 for price going down, 1 for price going up or staying stable). Regression models will predict the next day's price.

How can it be used?

A user can use this model by providing it with text data from the previous day from *The Motley Fool*, *Market Watch*, Reddit and/or Twitter to come up with a prediction for the following day. Not all sources are required, but providing the model with more data can potentially increase the accuracy of it.

Implementation

After extracting the data, we performed additional transformations to prepare the data for the models. For the news articles, the data frame consisted of text, url, title and timestamp for each article. The text, url and title columns were combined into a new column to ensure we did not miss any important words. The combined data is then cleaned to remove numbers (0-9 to POSNUM, <0 to NEGNUM), remove shortcuts (they're to they are), and remove punctuations. Then, the data is tokenized to split entire articles into words. Upon tokenization, stopwords (words that frequently occur across all documents eg. articles, pronouns) are removed.

The next step involved generating the sentiment scores for each article which determines the emotion of the text. We have used VADER (Valance Aware Dictionary and sEntiment Reasoner) to get the sentiment scores with SentimentIntensityAnalyzer() to identify positive, negative, neutral and compound scores. Next, we used compound scores which are obtained by normalizing the sum of positive, negative and neutral scores. Compound scores range from -1 (extreme negative) to 1 (extreme positive).

Once the scores for all the articles were generated, we calculated a sentiment score for a single day by calculating the mean of sentiments of all articles published on a single day. A similar process was implemented for the Twitter dataset.

The stock price dataset had timestamp, open, close, low, high, volume, tradecount, and vwap columns. This dataset had records for every hour of a particular stock. We consolidated all the records of a day into a single record. This was achieved by calculating the mean value for the open, close, high, and low columns individually. A new column ‘Label’ was created by checking if the closing price of one day was greater than the price of the previous day. Label 0 indicates the price went down, 1 indicates stable or increase in price.

The stock dataset and news article dataset were joined on timestamp.The twitter dataset was joined on timestamp with the stock dataset. A new column C-O was generated to store if the price of the stock decreased or not in a day. The target column is Label, the factors are open, close, high, low, sentiment scores, and C-O columns.

These datasets were then split into training and test sets on a 70:30 ratio. The training sets were trained on the classification models, which were later used to predict the movement of the price on the testing set. The datasets were stored as pickle and CSV files for easy retrieval in the future.

For regression models, the dataset had to be scaled and a new dataset was created.The new dataset was created combining all the columns taking window size into consideration (X) and the target column was loaded into y. Scaling was important to make sure that all the values were on a single plane. We used a window value in creating a dataset for regression models.

Baseline Implementation

Our baseline involved using historical stock prices to predict future prices using a regression model. The implementation of the baseline was relatively easy since it involved only one column (i.e closing price). It was a univariate model and relatively easy to design and implement.

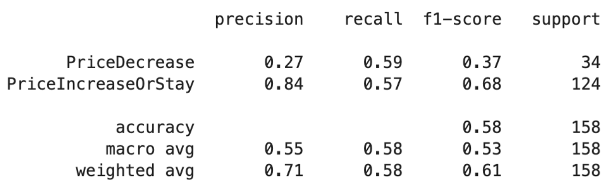
Experiments

Classification Experiments

We implemented classification models to predict if the stock price would go up (1) or down (0) given that the model has the sentiments of the previous day.

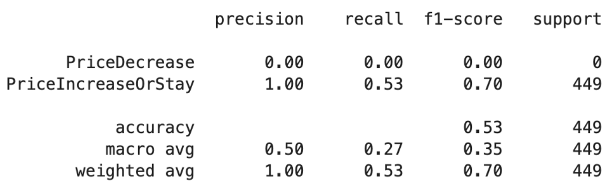
Support Vector Machine (SVM)

The first experiment was done using an SVM Classifier to determine the movement of stock price using *Market Watch* articles' sentiments. Below are its metrics:



```
                     precision    recall  f1-score   support

      PriceDecrease       0.27      0.59      0.37        34
PriceIncreaseOrStay       0.84      0.57      0.68       124

           accuracy                           0.58       158
          macro avg       0.55      0.58      0.53       158
       weighted avg       0.71      0.58      0.61       158
```

Then the SVM was used on *The Motley Fool* articles' sentiments, which gave us the below metrics:



```
                     precision    recall  f1-score   support

      PriceDecrease       0.00      0.00      0.00         0
PriceIncreaseOrStay       1.00      0.53      0.70       449

           accuracy                           0.53       449
          macro avg       0.50      0.27      0.35       449
       weighted avg       1.00      0.53      0.70       449
```

Then, the sentiments of *The Motley Fool* and *Market Watch* articles were combined. The mean of the sentiments were taken for each day and then the sentiment was then used to predict the stock movement using SVM. Below are its metrics:

```
                     precision    recall  f1-score   support

      PriceDecrease       0.03      0.67      0.05         3
PriceIncreaseorStay       0.99      0.54      0.70       155

           accuracy                           0.54       158
          macro avg       0.51      0.60      0.38       158
       weighted avg       0.97      0.54      0.69       158
```

Based on the metrics above, it was clear that the models were doing well in predicting if the price was going to increase or stay stable. The same cannot be said about the price decrease prediction.

We then moved to predicting the stock movement using Twitter. The sentiment scores of Tweets each day were used to predict if the price was going to decrease or not. The metrics of this model are below:

```
                     precision    recall  f1-score   support

      PriceDecrease       0.67      0.64      0.65       214
PriceIncreaseorStay       0.66      0.69      0.67       216

           accuracy                           0.66       430
          macro avg       0.66      0.66      0.66       430
       weighted avg       0.66      0.66      0.66       430
```

This model proved to be a better one compared to the others. The accuracy increased and the precision of the price decrease prediction also increased.

Random Forest Classifier

The next experiment was conducted using the Random Forest Classifier. Random Forest classification involves building numerous decision trees from the samples of training data. The output is based on the class that receives major votes. We used the twitter dataset combined with stock prices for this model. The RF model metrics came out as below:

```
              precision    recall  f1-score   support

           0       0.73      0.64      0.68       203
           1       0.71      0.79      0.75       227

    accuracy                           0.72       430
   macro avg       0.72      0.72      0.72       430
weighted avg       0.72      0.72      0.72       430
```

Gradient Boosting Classifier

Gradient Boosting Classifier is a model which when combined with the previous model would minimize the loss/error rate in prediction. We tried to tune the GB by using the following various learning rates:

$$learning\_rates = [0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1]$$

We used the best learning rate to predict the stock price based on the sentiment scores of tweets.This is the model's metrics:

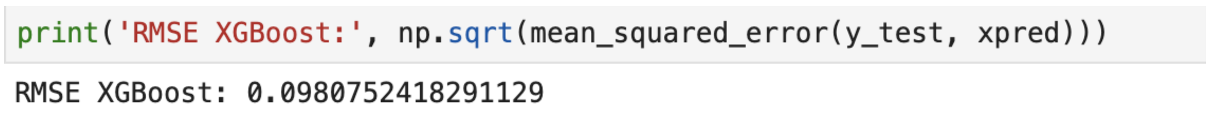
```
              precision    recall  f1-score   support

           0       0.77      0.67      0.72       203
           1       0.74      0.81      0.77       227

    accuracy                           0.75       430
   macro avg       0.75      0.74      0.75       430
weighted avg       0.75      0.75      0.75       430
```

Regression experiments

Regression Models were used to determine the next possible price of the stock given the previous prices and the sentiments of the previous day.

XGBoosting

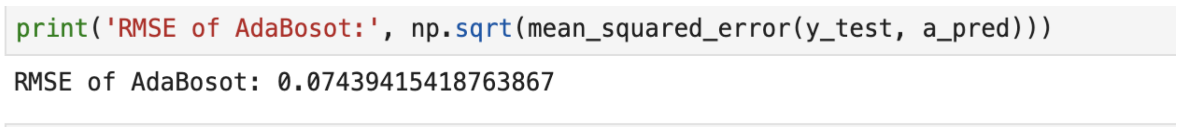
Extreme Gradient Boosting is an efficient implementation of Gradient Boosting. XGBoost is a specific implementation of the Gradient Boosting Model which uses more accurate approximations to find the best tree model. We have used closing price, closing-open difference, and Twitter sentiment score for predictions. Below are the metrics:



```
print('RMSE XGBoost:', np.sqrt(mean_squared_error(y_test, xpred)))
RMSE XGBoost: 0.0980752418291129
```

AdaBoosting

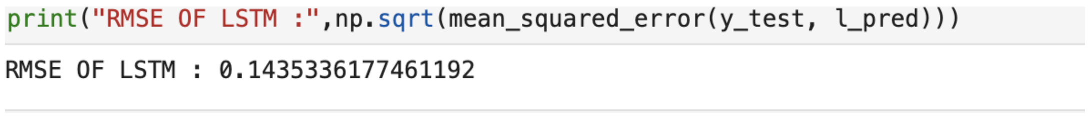
AdaBoost algorithm works on changing the sample distribution by modifying weight data points for each iteration. The dataset used for XGBoost is used for AdaBoost as well.The metrics are:



```
print('RMSE of AdaBosot:', np.sqrt(mean_squared_error(y_test, a_pred)))
RMSE of AdaBosot: 0.07439415418763867
```

Long-Short Term Memory (LSTM)

LSTMs are considered the best approach for predictions using time series data. LSTM consists of cells that remember values and gates that regulate flow of information in and out of cells.We used twitter sentiment and closing prices for this model. It produced the following metrics:



```
print("RMSE OF LSTM :",np.sqrt(mean_squared_error(y_test, l_pred)))
RMSE OF LSTM : 0.1435336177461192
```

Data Extraction

Webscraping *The Motley Fool* and *Market Watch*

The Motley Fool and *Market Watch*, like many news sources, do not provide APIs for the general public to interface with to extract their articles. Therefore, to get the text from their articles, we used Selenium, a Python web driver and Beautiful Soup, a Python package that interprets HTML, to download the data.

For both sources, we used the webdriver to automate clicking a button that loads more articles on their respective web pages that aggregated links to articles about Apple and

Tesla. Once the page reached enough articles, we exported the source code of the page, which included a list of articles that we would iterate through.

The next step involved sending a request to each of the links and then extracting the HTML from the web pages retrieved. This included using a Chrome Extension called SelectorGadget that identifies the CSS path of elements of interest from the UI of the webpage. This extension was most useful for identifying the articles' text elements of interest by summarizing them into one element. Additionally, this loop caught potential exceptions the parsing could have when trying to get elements of interest such as the title, date or text from the articles. The final step in preparing this data required casting the string time columns to UTC timestamps which would enable us to join the article data with the price data timestamps we retrieved from Alpaca. We had 2655 articles from Market Watch and 2599 articles from Motley Fool.

Getting Tweets from Twitter

Another source we were interested in analyzing included Tweets from Twitter. Tweets could include an ever broader range of sources for article headlines, as well as a general sentiment of a large population of people influencing the stock market. Additionally, we were able to get a much larger data set from Twitter by activating 2 Academic Research API Tokens allowing us to get up to 20M tweets combined per month.

After several failed attempts of retrieving Tweets and losing them, we finally came across a video that described the exact process of getting the Tweets to demonstrate how to iterate the structure of the raw data. Other tutorials and videos we stumbled across were not specific to the Academic Research Tier which was important due to the amount of data we were requesting. A small misstep in iterating through the Tweet objects could cause the Python kernel to crash along with the Tweets gathered which was troublesome because not only did some of these scrapes take over 5+ hours, but each token was capped at 10M Tweets per month, regardless if you were able to successfully retain them.

Getting the Tweets involved using Tweepy and its paginator along with a custom query that we created for Tesla and Apple respectively. We did this process in batches to minimize the damage of an unsuccessful request which would result in manually identifying the last successful Tweet retrieved and then changing the start date accordingly. It was important to reduce any unnecessary iteration of these objects as it could crash the kernel at any moment due to the size of the objects.

Extracting Comments from Reddit

We did not want to miss out on any data sources, especially the infamous Reddit thread, *r/wallstreetbets*, solely responsible for Gamestop’s 1700% spike in 2021. After doing some research on Reddit’s native API, we realized that it was limited in its capabilities of capturing past data. Instead, we came across PushShift, a third-party API which routinely scrapes and stores Reddit in its own database. Furthermore, PMAW (Pushshift Multithread API Wrapper) is a library that interfaces with Pushshifts database allowing for sequential calls magnifying the data you can scrape while minimizing the time it takes to run.

We came across this article which details how you can use Matt Podolak's PMAW library to scrape 1M comments from a Reddit thread. We tried to extract more items and ran into errors and then did a little more digging in his article which mentioned he had an upcoming article that would detail scraping items ranging from 500k to 10m; however, he never got around to writing it. We reached out to Matt on LinkedIn and he mentioned that we could use the memory safety feature to cache documents to disk and to write to CSV in batches to achieve the goal. Maybe we could test the limits of his library in a future project, but for now we were able to gather 2M Reddit comments.

Getting Price Data from Alpaca

Initially, we gathered price data from Yahoo for Apple and Tesla stock. We had gone back to adjust the criteria of this API call to account for articles that we encountered from more than 5 years ago and learned that the Yahoo API did not support this range capability. We did some more exploring and came across another API, Alpaca, which allowed us to get hourly data from more than 5 years ago. We were able to convert the raw response to a dataframe.

Baseline vs Best

Our baseline model was using a univariate LSTM model to predict closing prices.The RMSE score was 114.8829350010486. The best Regression model we found was using

ADABoost with sentiment scores. The RMSE for this model was 0.07439415418763867.

RMSE(Root Mean Squared Error) tells us how far are the predictions from the actual values.The formula for RMSE

$$RMSE = \sqrt{\overline{(f - o)}^2} = \sqrt{\sum_{i=1}^{N} (z_f - z_o)^2 / N}$$

f= predicted values

o=actual value

bar above the difference square is mean

N=sample size

The best classification model we found was Gradient Boosting Classifier which gave the accuracy of 74.883%.

Conclusions:

From this process, we learned how to web scrape data from a variety of sources such ranging from financial to social sources. Webscraping *The Motley Fool* and *Market Watch* taught us methodologies that could translate to web scraping data from any news source to theoretically power any future NLP project. Similarly, interfacing with PMAW, Tweepy and Alpaca taught us how we could connect to APIs which could be useful in future projects that require data gathering from similar sources.

This work has emphasized the importance of preprocessing the data before using it to build the models. Trying out multiple models made us understand that the analysis on the data should be done before deciding which model to be used. Classification models were new to us and we learnt how to implement and tune them.The regression models gave insight into how to deal with time series data and that regression is a better approach for such data.

We implemented all of the methods mentioned for Apple.But were not able to finish them for Tesla. For tesla, the tweets processing kept going for a long time and crashed the system.So, for the classification model we used, motley fool articles and reddit comments for prediction

References

https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

https://medium.com/swlh/how-to-scrape-large-amounts-of-reddit-data-using-pushshift-1d33bde9286

https://www.youtube.com/watch?v=rQEsIs9LERM