Predicting the movement of the stock price using Sentiment Analysis

Naveen Mallemala (nm3937)
Jash Unadkat (jru5640)
YiChen Lin (yl9396)
Jiazhao Shi (js12624)

1 Problem Statement

The objective of this project was to predict the movement of the stock price using Sentiment Analysis. The first step in hypothesizing the model was to ask ourselves a few questions. These have been listed below:

1. Is it possible to implement a model that uses Sentiment Analysis to predict whether the trend in the stock price of the SP500 will increase or decrease?

2. Why must we use techniques from Big Data to predict the stock price from Reddit news?

3. What kind of accuracies can you get by implementing multiple different Big Data models on the data that contains the date, the news from that date and a label where 1 or 0 indicates an increase or decrease in the price respectively?

The first question was to learn about the relationship between Sentiment Analysis and making predictions on Big Data. We concluded from previous papers [[1], [2], [3]] that this method has the potential to extract useful information and can be attributed to a label of 1 for positive sentiments and a label of 0 for negative sentiments.

The next step was to ascertain the techniques in Big Data that we could use to preprocess the data into the correct format which could then be implemented in Big Data platforms such as PySpark and MongoDB.

This data would then have to be run on models to achieve the highest accuracies. We decided to implement three different models. This would allow us to show how numerous models compared and provide insight for future work.

This set the foundations of how we would tackle this problem and to create the following architecture for the project.

2 Architecture

Figure 1 demonstrates an overview of how we completed the project.

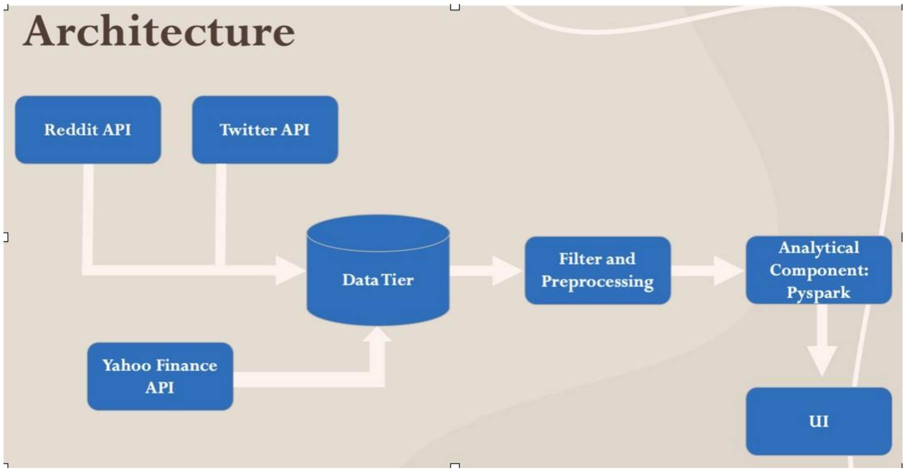


Figure 1: Architecture

We first needed to collect the data. The data was obtained using APIs for Reddit, Twitter and Yahoo Finance. The data fetched from Reddit was collected using PushshiftAPI and Praw. The data fetched from Twitter was collected using Tweepy. The Reddit and Twitter data contained comments and posts from a particular date that we would use as the text variable. In addition, we also collected data from Kaggle that was used to test our model initially on data that was already in a format similar to what we wanted to achieve Figure 2. The data from Yahoo Finance was the stock price change of the S&P500 for a particular date Figure 3. This was displayed as a positive or negative percentage. The hope was that the text data showed a correlation and reasoning for why the S&P500 increased or decreased. We were able to collect over a million data points. All this data was collected into a csv file and stored for preprocessing.

We discovered that not only were we going to have to manage the volume of data but also how unstructured the data was formatted. This meant a lot of preprocessing. The first stage was to simply convert the percentage change of the S&P500 to a label. A negative percent indicated a drop in price of the S&P500 and vice versa. Therefore, we decided to convert any negative percent change to label 0 and any positive percent change a label 1. For example, if the number was -0.002738, we labelled that date 0 and this this indicated that the stock price fell on that particular date. The next stage was the preprocessing of the text data which was more complicated. We used multiple techniques from Big Data such as a Bloom Filter, PySpark N-grams and Broadcast Variable. The details of this are described in the Data and Preprocessing section below.

After the data had been preprocessed the task was to pass it through our analytical component. The tool we used to implement our models was PySpark in which we conducted Count-Based approach, a Jaccard Similarity Search and Machine Learning classifiers. These models required further Big Data techniques. These included a PySpark window, Cartesian Join, MongoDB, Spark ML libraries in addition to those aforementioned.

3 Data and Preprocessing

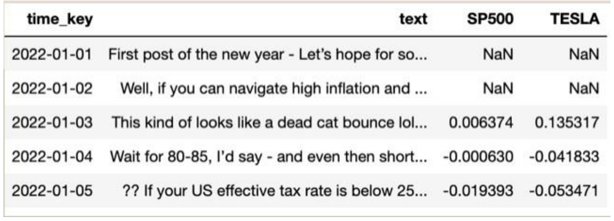
In this section, we describe how data was fetched and how it was preprocessed before being used to train our models.

3.1 Crawl

We fetched social media posts from Reddit and Twitter. To crawl data from Reddit, we made use of the PushshiftAPI and the Praw API. For Twitter, we made use of Tweepy API. For both Reddit and Twitter data, we fetched the ID of the post, title of the post, comments related to a post, and timestamp of when the post or comment was created.

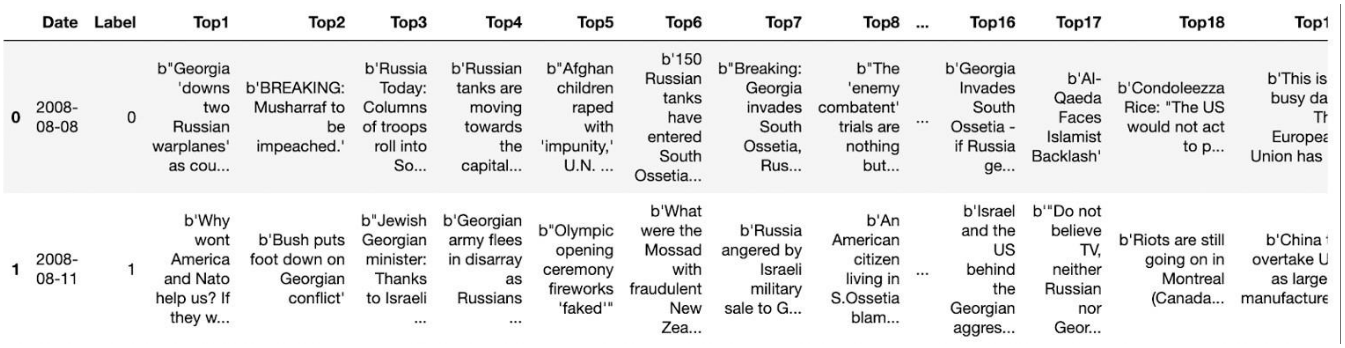
Further, we fetched stock data from Yahoo Finance using the Yahoo Finance API. For the stock data, we fetched the stock price for each stock at a particular date, then we calculated the difference from the previous date to get the percentage increase or decrease.

Afterwards, we merged the social media posts and the stock data and the result is shown below:



time_key	text	SP500	TESLA
2022-01-01	First post of the new year - Let's hope for so...	NaN	NaN
2022-01-02	Well, if you can navigate high inflation and ...	NaN	NaN
2022-01-03	This kind of looks like a dead cat bounce lol...	0.006374	0.135317
2022-01-04	Wait for 80-85, I'd say - and even then short...	-0.000630	-0.041833
2022-01-05	?? If your US effective tax rate is below 25...	-0.019393	-0.053471

Figure 2: Sample data



	Date	Label	Top1	Top2	Top3	Top4	Top5	Top6	Top7	Top8	...	Top16	Top17	Top18	Top1
0	2008-08-08	0	b"Georgia 'downs two Russian warplanes' as cou...	b'BREAKING: Musharraf to be impeached.'	b'Russia Today: Columns of troops roll into So...	b'Russian tanks are moving towards the capital...	b"Afghan children raped with 'impunity,' U.N. ...	b'150 Russian tanks have entered South Ossetia...	b"Breaking: Georgia invades South Ossetia, Rus...	b"The 'enemy combatent' trials are nothing but...	...	b'Georgia Invades South Ossetia - if Russia ge...	b'Al-Qaeda Faces Islamist Backlash'	b'Condoleezza Rice: "The US would not act to p...	b'This is busy da Th Europea Union has
1	2008-08-11	1	b'Why wont America and Nato help us? If they w...	b'Bush puts foot down on Georgian conflict'	b"Jewish Georgian minister: Thanks to Israeli ...	b'Georgian army flees in disarray as Russians ...	b"Olympic opening ceremony fireworks 'faked'"	b'What were the Mossad with fraudulent New Zea...	b'Russia angered by Israeli military sale to G...	b'An American citizen living in S.Ossetia blam...	...	b'Israel and the US behind the Georgian aggres...	b'"Do not believe TV, neither Russian nor Geor...	b'Riots are still going on in Montreal (Canada...	b'China overtake U as large manufacture

Figure 3: Sample data from Kaggle

3.2 Preprocessing

Before passing the data to the models, it needs to be preprocessed. We have made use of Bloom Filter to quickly and efficiently filter unrelated posts, Pyspark ML to create n-grams and to remove unnecessary words and characters.

3.2.1 Bloom filter

A bloom filter is a key-value storage system, which is space efficient and provides extremely fast filtering to determine if an element is part of a set. It can allow false positive but not false negative. We used it in our project to determine if a post is related to a stock or not. We did so by first going through all post titles and we would add the IDs of the titles containing certain keywords related to the stock in the bloom filter. Then, we will go through all of the social media data and filter out all the comments and posts based on the IDs in the bloom filter.

3.2.2 Pyspark ML

Before passing the data to the model, all the sentences in a post needed to be converted to n-grams and all unnecessary words needed to be removed. To do so, we made use of Pyspark ML's NGram and StopWordsRemover modules.

4 Methods

4.1 Count-Based Approach

We first implemented a model using word counting. This is done on *Pyspark*. We considered a word positive if it appears when there has been an increase in the stock market, and negative otherwise. Then, we used a ratio of positive to negative to determine the sentiment for the market and predict if the stock return is positive or negative. The best accuracy we got was around 51%.

4.1.1 Methodology

We first performed a word counting on training data. Specifically, we counted the number of words when the market rose and dropped separately. We denote them by positive counts (pos_{word}) and negative counts (neg_{word}), respectively. Since we have seen more positive days (a day the market rises), our word count was biased to have larger values in positive counts. To fix this issue, we first normalize them as

$$\overline{pos}_{word} = \frac{pos_{word}}{\sum_{word} pos_{word}}, \quad \overline{neg}_{word} = \frac{neg_{word}}{\sum_{word} neg_{word}}$$

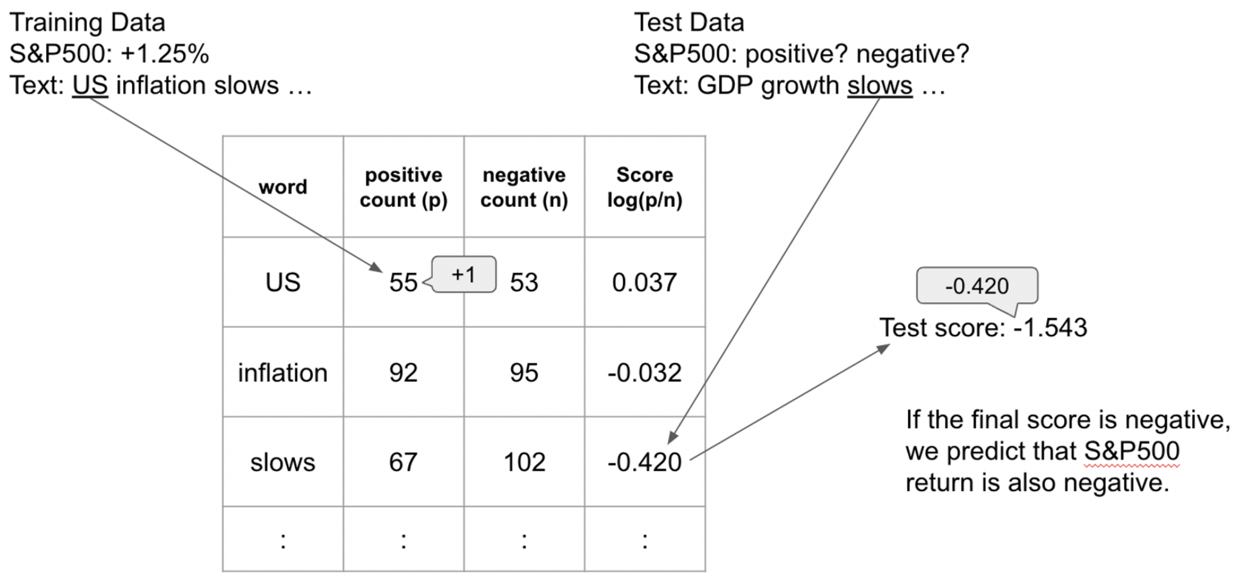
Then, we compute a score for each word as

$$s_{word} = \log \frac{\max(\epsilon, \overline{pos}_{word})}{\max(\epsilon, \overline{neg}_{word})}$$

Note that we set $\epsilon = 1e - 7$. ϵ is used to avoid computing log 0, which is undifined. When giving a sentiment analysis to text in the test set, we only used frequent words. This is because s_{word} can be seen as an estimation of true score $\mathsf{E}[s_{word}]$. Thus, if the number of samples is small, our estimation has a high variance, and this makes our analysis noisy. To determine the sentiment for texts in the test set, we compute the sentiment score (S) as

$$S = \sum_{word\in text} s_{word}$$

If a word is not in our vocabulary, we just ignored it by considering it as neutral, namely, $s_{word} = 0$. Finally, if this sentiment score is positive, we conclude that the given text indicates the market rose. Otherwise, we predict that the market dropped. As an extension, we also tried the same approach using bigram. However, we couldn't see an improvement in accuracy.


Training Data
S&P500: +1.25%
Text: US inflation slows …
Test Data
S&P500: positive? negative?
Text: GDP growth slows …
+1
-0.420
Test score: -1.543
If the final score is negative, we predict that S&P500 return is also negative.

word	positive count (p)	negative count (n)	Score log(p/n)
US	55	53	0.037
inflation	92	95	-0.032
slows	67	102	-0.420
:	:	:	:

Figure 4: The Count-Based Approach Sketch (Unigram Case)

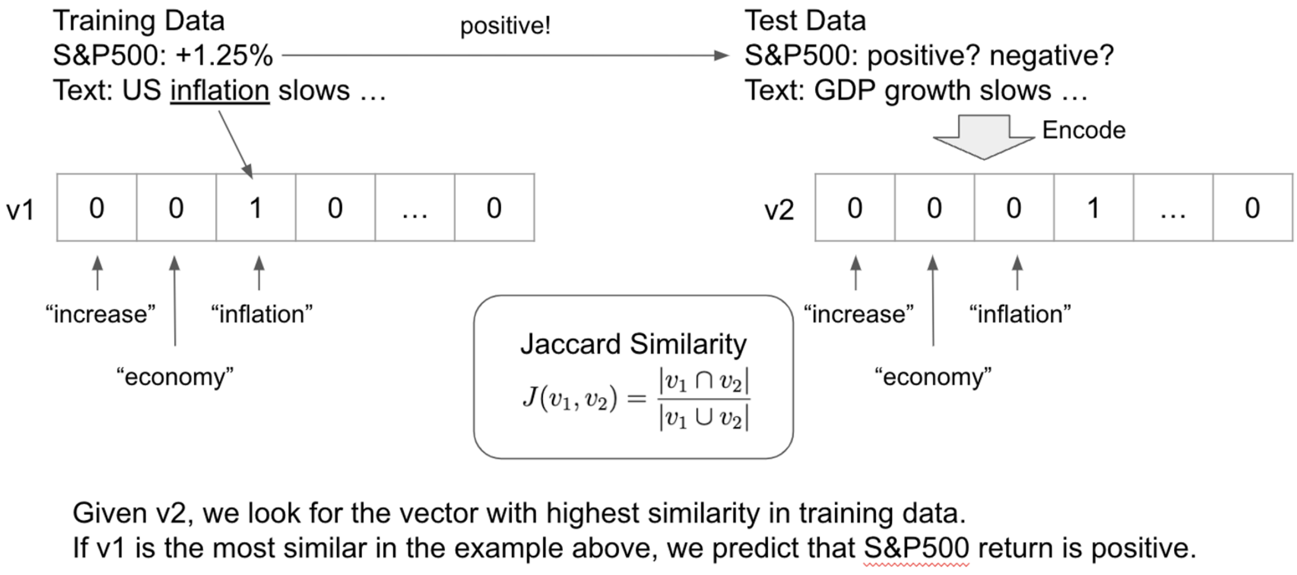
4.2 Jaccard Similarity Search

As an alternative way of encoding texts and making a prediction, we consider the use of binary vector representation and the use of Jaccard similarity.

4.2.1 Methodology

For the binary vector encoding, given a text, we set the corresponding entry in the vector to 1 if a word appears in the text, and set it to 0 otherwise. Since this is not a bag of words, all the entries in the vector are either 0 or 1. In the implementation, we used only frequent words to ease the computational complexity. Then, the Jaccard similarity is computed as

$$J(u,v) = \frac{|u\cap v|}{|u\cup v|} = \frac{\sum_{i=1}^{d} \mathbb{1}[u_i = 1 \wedge v_i = 1]}{\sum_{i=1}^{d} \mathbb{1}[u_i = 1 \vee v_i = 1]}$$



Given v2, we look for the vector with highest similarity in training data.
If v1 is the most similar in the example above, we predict that S&P500 return is positive.

Figure 5: The Jaccard Similarity Approach Sketch

where u and v are the binary vectors and d denotes the length of them. Using Jaccard similarity, given a text query from the test set, we can choose a day from the training set that has the highest similarity. We assume that if the texts are similar, what happens in the stock market are also similar. Thus, our prediction is the same as the label of the highest-similarity day. With this model, we got 56% accuracy. We implemented this model in *PySpark*.

4.2.2 MinHash

One disadvantage of the model in 4.2.1 is that we might want to have the vector length d very large if we have access to the huge text data. In this case, computing the Jaccard similarity for a pair of vectors increases linearly with a naive implementation. To address this issue, we came up with the use of MinHash as a prepossessing. First, we choose k random hash function $h_1, \cdots, h_k : \{1, \cdots, d\} \to [0, 1]$. Then, given a binary vector $u \in \{0, 1\}^d$ form a vector $c^{(u)} \in [0, 1]^k$ such that each entry is computed as

$$c_i^{(u)} = \min_{j \in [d], u_j = 1} h_i(j)$$

Using this, we can estimate the Jaccard similarity of binary vectors u, v by

$$\tilde{J}(u, v) = \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}[c_i^{(u)} = c_i^{(v)}]$$

There is a theoretical guarantee from Chebyshev's inequality that this $\tilde{J}(u, v)$ is an estimation of $J(u, v)$ up to a constant additive error ϵ such that

$$J(u, v) - \epsilon \leq \tilde{J}(u, v) \leq J(u, v) + \epsilon$$

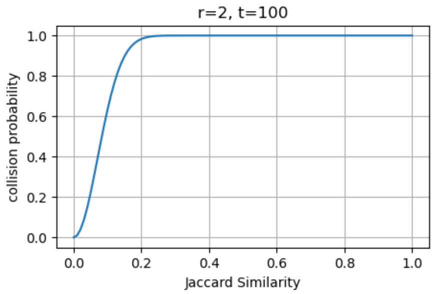


Figure 6: Collision Probability in LSH

This inequality holds with high probability if we choose $k = O(\frac{1}{\epsilon^2})$ random hash function, which is independent of the length of binary vector d. In our implementation, we set $k = 120$. The implementation was done in *PyMongo*. This choice is because MongoDB is useful when our data is unstructured. The posts on Reddit, for example, we have a significant number of posts when the market crashes, while we see a moderate number of posts on normal days. As the number of posts is different day by day, we didn't choose the tabular format here.

4.3 Machine Learning Models

The outline for the steps involved in running our data through ML models in *Spark* is detailed below:

1. load the data from Reddit
2. preprocess the text as described in section 3
3. apply transformations
4. run ML classifiers
5. evaluation

We created a *Spark ML* Pipeline to provide a more detailed structure of the steps taken is displayed in Figure 7.

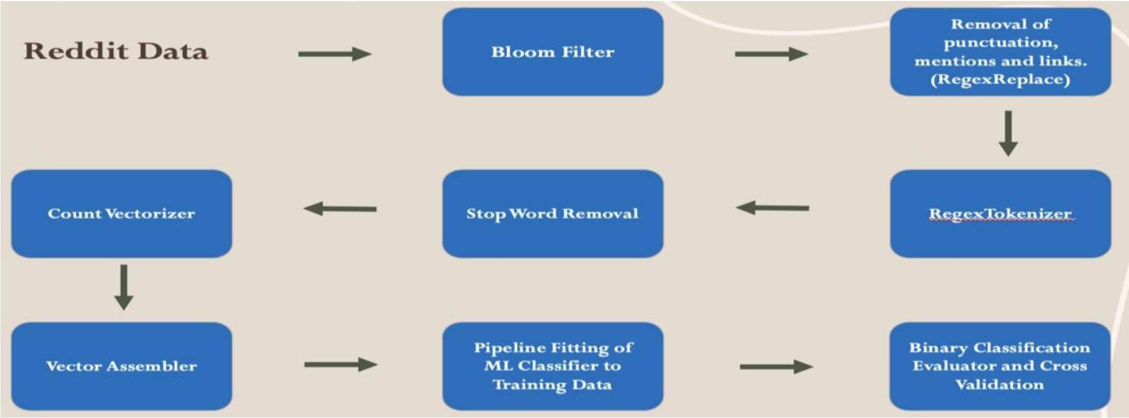

Reddit Data
Bloom Filter
Removal of punctuation, mentions and links. (RegexReplace)
RegexTokenizer
Stop Word Removal
Count Vectorizer
Vector Assembler
Pipeline Fitting of ML Classifier to Training Data
Binary Classification Evaluator and Cross Validation

Figure 7: Spark ML Pipeline

4.3.1 Transformations

The third step in the ML process was to apply transformations to the data that would help increase the accuracy. We applied the following transformations:
Regex Tokenizer splits the text of the news into individual words or phrases using a regular expression. We specified the pattern to be ¥¥w+ . The ¥¥w character class represents any ”word” character, bawhich includes letters, numbers, and the underscore character. The + character is a quantifier that specifies that the preceding character class (¥¥w) should be matched one or more times.
Count Vectorizer then takes the tokens and helped us identity the most frequently used and significant words and convert them into sparse vector representation of the input text. Each element of the vector represents the number of times a particular token appears in the text.
Vector Assembler was used to take the sparse vectors and combine them into a single dense vector column to help reduce the dimensionality of the data.
StopWordsRemover removes the words from the Reddit text that have little or no meaning on its own such as ‘the’, ‘a’, or ‘and’. This allows the ML model to focus on the more meaningful words and phrases in the text.

4.3.2 Models

After we applied these transformations, we were able to run the correctly processed data on numerous ML classifiers. The benefit of training the data on multiple classifiers is that we can find how strengths and weaknesses of different models. The following classifiers were used:
Random Forest is able to handle complex and non-linear relationships in the data in addition to the fact that they are relatively robust to overfitting.
Logistic Regression is well suited to binary prediction tasks and can provide estimates of the likelihood of different outcomes.
Naïve Bayes helps makes predictions based on the individual probabilities of the individual words from the Reddit data and is appropriate when handling large numbers of features.
Linear Support Vector Classifier uses the feature words to learn a decision boundary and find the hyperplane that best separates the data into the two classes (e.g., increasing or decreasing stock price) to make predictions on new data.
Decision Trees learns the rules by splitting the data into smaller and smaller subsets based on the values of the features, and then uses these splits to make predictions on new data.

4.3.3 Evaluation

We used three techniques to evaluate the performance of our ML models:
Binary Classification Evaluator: we provided the label as the parameter and computed performance metrics such as the Area Under the Receiving Operating Characteristic. AU-ROC is a robust metric that is not affected by the choice of classification threshold, which can be important in situations where the cost of false negatives (missing a buying opportunity) and false positives (incurring unnecessary trading costs) is not equal.

Cross Validation: this technique can help to reduce overfitting, which is important when predicting stock prices, as overfitted models may not generalize well to new data and may produce poor predictions. CV also provides a more accurate estimate of a model's performance than using a single train-test split, as it involves training and evaluating the model multiple times using different subsets of the data.
Comparison Plots: helped visualize how the different models performed relative to each other. The results for this are outlined in section 5.

5 Insights and Reports

The ML classifiers showed different accuracies and the comparison plots below represents this. We ran the model multiple times and depending on the randomized split of test train data the results varied and so the plot is an example of one of the times we ran the models. As shown in Figure 8, some classifiers ran significantly better than others.
Figure 9 helps understand better how all three methods performed in comparison to each other. The objective of the project was to see if we could get accuracies better than random chance or flipping a coin. The main outcome we noticed was that all the models were not significantly strong on a consistent basis however we were able to achieve the objective. The variation within each model did surprise us.

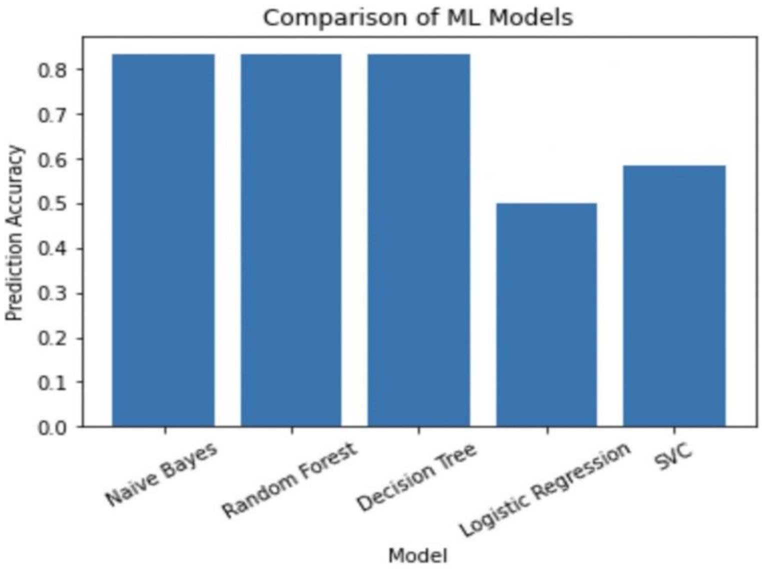
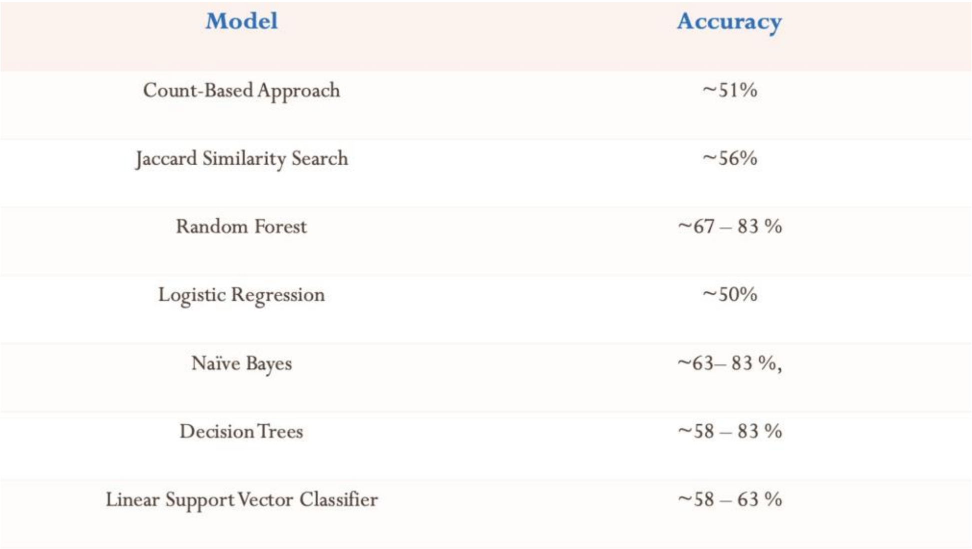


Figure 8: Comparisons Plot for the Machine Learning Models

We were able to achieve higher than 65% in some instances such as Random Forest, Na¨ıve Bayes and Decision Trees however we concluded that only Random Forest would be somewhat suitable for future improvement.



Model	Accuracy
Count-Based Approach	~51%
Jaccard Similarity Search	~56%
Random Forest	~67 – 83 %
Logistic Regression	~50%
Naïve Bayes	~63– 83 %,
Decision Trees	~58 – 83 %
Linear Support Vector Classifier	~58 – 63 %

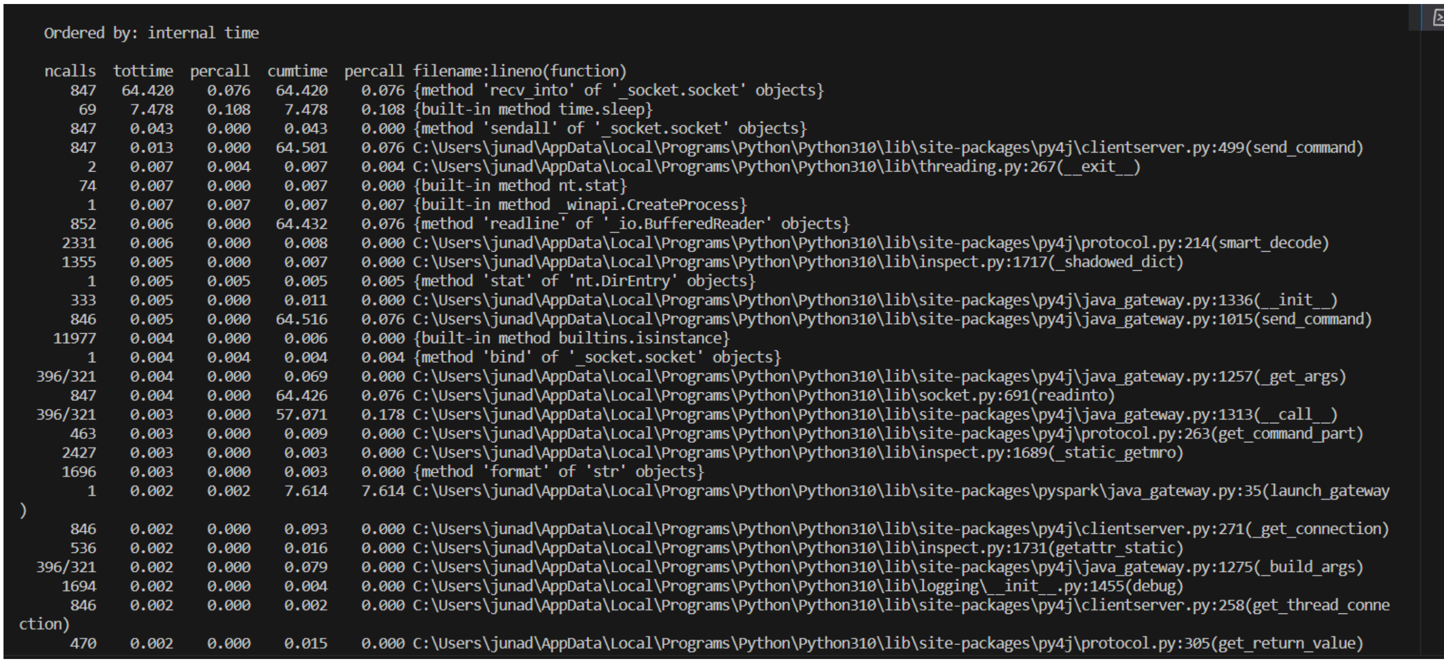
Figure 9: Insights and Reports of All Models

There are multiple reasons why the ML classifiers did not show consistent high accuracies.

Features: different classifiers might be more or less sensitive to certain features, which could impact their accuracy. For example, the fact that we did not have a large number of features may have not optimized the use of a classifier that is better at handling high-dimensional data.

Nature of the Data: the accuracy of a classifier can also be influenced by the characteristics of the data it is being applied to. For example, if the data is highly unbalanced e.g. if there are significantly more positive sentiments than negative or more positive increases in stock price. Some classifiers might be better suited to handle this type of data than others.

Jaccard Similarity Profiling results:



```
   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      847   64.420    0.076   64.420    0.076 {method 'recv_into' of '_socket.socket' objects}
       69    7.478    0.108    7.478    0.108 {built-in method time.sleep}
      847    0.043    0.000    0.043    0.000 {method 'sendall' of '_socket.socket' objects}
      847    0.013    0.000   64.501    0.076 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\clientserver.py:499(send_command)
        2    0.007    0.004    0.007    0.004 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\threading.py:267(__exit__)
       74    0.007    0.000    0.007    0.000 {built-in method nt.stat}
        1    0.007    0.007    0.007    0.007 {built-in method _winapi.CreateProcess}
      852    0.006    0.000   64.432    0.076 {method 'readline' of '_io.BufferedReader' objects}
     2331    0.006    0.000    0.008    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\protocol.py:214(smart_decode)
     1355    0.005    0.000    0.007    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\inspect.py:1717(_shadowed_dict)
        1    0.005    0.005    0.005    0.005 {method 'stat' of 'nt.DirEntry' objects}
      333    0.005    0.000    0.011    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\java_gateway.py:1336(__init__)
      846    0.005    0.000   64.516    0.076 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\java_gateway.py:1015(send_command)
    11977    0.004    0.000    0.006    0.000 {built-in method builtins.isinstance}
        1    0.004    0.004    0.004    0.004 {method 'bind' of '_socket.socket' objects}
  396/321    0.004    0.000    0.069    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\java_gateway.py:1257(_get_args)
      847    0.004    0.000   64.426    0.076 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\socket.py:691(readinto)
  396/321    0.003    0.000   57.071    0.178 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\java_gateway.py:1313(__call__)
      463    0.003    0.000    0.009    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\protocol.py:263(get_command_part)
     2427    0.003    0.000    0.003    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\inspect.py:1689(_static_getmro)
     1696    0.003    0.000    0.003    0.000 {method 'format' of 'str' objects}
        1    0.002    0.002    7.614    7.614 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\pyspark\java_gateway.py:35(launch_gateway
)
      846    0.002    0.000    0.093    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\clientserver.py:271(_get_connection)
      536    0.002    0.000    0.016    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\inspect.py:1731(getattr_static)
  396/321    0.002    0.000    0.079    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\java_gateway.py:1275(_build_args)
     1694    0.002    0.000    0.004    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\logging\__init__.py:1455(debug)
      846    0.002    0.000    0.002    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\clientserver.py:258(get_thread_conne
ction)
      470    0.002    0.000    0.015    0.000 C:\Users\junad\AppData\Local\Programs\Python\Python310\lib\site-packages\py4j\protocol.py:305(get_return_value)
```

Here,
ncalls: The number of calls made to the function
tottime: The total time spent in the function excluding time spent in calls to other functions
percall: The average time spent in the function per call
cumtime: The cumulative time spent in the function including time spent in calls to other functions
percall: The average cumulative time spent in the function per call
filename:lineno(function): The file name, line number, and function name where the function is defined

For Jaccard Similarity Code Results click on the link below:

https://drive.google.com/file/d/1jzptegfx7u8TbwUPxOkpF1TTJLZUXxxh/view?usp=share_link

The data was also large in volume but low in quality which may suit certain classifiers.
Underlying Assumptions: different classifiers make different assumptions about the data they are being applied to, and these assumptions can impact their performance. For example, the difference between the linear and non-linear classifier might affect performance with highly non-linear data.
Hyperparameter Settings: the accuracy of a classifier can also be influenced by the specific hyperparameter settings that are used. For example, if the learning rate for a gradient boosting classifier is set too high, it might overfit the training data and perform poorly on new data.

The Count-Based approach also has limitations.
Scaling of the Data: word count approaches are sensitive to the scaling of the data and the presence of noise. If the data is not properly scaled or contaminated with irrelevant of misleading information, the word count approach may distribute the weight to certain features that are not as important.
Complex Relationships: if there is a non-linear relationship between two features, a word count approach might not be able to capture this more complicated relationship.

The low accuracy from the Jaccard Similarity Search may have the following limitations.
Importance of Features: Jaccard similarity is based on the presence or absence of individual features, rather than their importance or relevance. This means that it may not be able to capture the most important features for predicting stock prices.
Complex Relationships: Jaccard similarity is a relatively simple measure of similarity, and it may not be able to capture the ambiguous relationships between Reddit news and stock price.
Noise: Jaccard similarity may be sensitive to the presence of noise in the data. If the data is contaminated with irrelevant or misleading information, Jaccard similarity might produce low accuracies.

6 Lessons Learned and Future Work

This project highlighted some advantages and disadvantages with our approach.
Reddit News has a lot of jargon and irrelevant information, so it is very hard to extract useful information. Therefore, Reddit News and Sentiment/Label alone are not sufficient in most cases to predict whether the stock price will increase or decrease.
Pyspark and MongoDB are both useful as we don't need to worry about what is happening behind the scenes.
Bloom Filters also extremely efficient at preprocessing the dataset. The techniques and method we chose showed potential however after reviewing and researching we have suggested the following approaches that may produce higher accuracies.
There is a need to use more data such as high low close open prices of multiple stocks. This gives a better indicator on exactly which parts of the stock price are affected the most and potentially could show if Reddit data has effects on one type of price or another. We also conducted a preliminary out of scope analysis on the Kaggle dataset using the Vader

Sentiment Analyzer tool in PySpark just for interest. This showed accuracies of over 80% when including all stock price information.

The Reddit posts and comments we used were filled with jargon and irrelevant information. We used this data as we found it more interesting to see if there was a relationship with this data than news articles. Moreover, the use of more robust news data such as data from Bloomberg or Financial Times would most likely yield higher accuracies as the stock price notably does correlate more with this kind of information.

There are some techniques that we researched that may show better results. For example, despite being outside the scope we found extremely interesting results when Long Short-Term Memory RNN were implemented. They are able to model long term dependencies and find patterns in data which could not only include news but also historical stock price data and market indicators.

References

[1] Darji M.D.N, Parikh S.M., and Patel H.R. “Sentiment Analysis of Unstructured Data Using Spark for Predicting Stock Market Price Movement”. In: (2022).

[2] C. Lee and I. Paik. “Stock market analysis from Twitter and news based on streaming big data infrastructure”. In: (2017).

[3] Kim R. “Sentiment analysis with pyspark, Medium”. In: (2020). url: https://towardsdatascience.com/sentiment-analysis-with-pyspark%20bc8e83f80c35.