A green and grey logo

Description automatically generated

Stock Sentiment

Predicting market behavior from tweets

Post-graduation in Data Science for Finance

Text Mining – 2023

Authors:

Luís Ribeiro (nº 20231536)

Fernando Tiago (nº 20231535)

# Introduction and Data Exploration

This project intends to perform sentiment analysis driven by Machine Learning, being fully automated. We were given a dataset of tweets and their respective “label” which correspond to the market sentiment expressed by the tweet: bullish; bearish; or neutral.

We will leverage “Pandas” and Sci-kit Learn to achieve this, leading us to the first step which is loading the dataset into Pandas.

A screenshot of a computer

Description automatically generated

Opening the training dataset, we can see that it’s composed of 9543 tweets and their respective, manually flagged, labels (0 for bearish, 1 for bullish, and 2 for neutral). We immediately spot some issues in the corpus that have to be tackled, such as the presence of URLs that carry no meaning and only serve as noise; stock tickers which while carrying meaning to a human reader, serve no purpose for a machine; special characters, etc.

Plotting a word cloud before the data preprocessing, we can easily see the first issue very clearly:

A close-up of words

Description automatically generated

The word “https”, which belongs to URLs, is cluttering the data. One of the first steps to achieving a proper, clean dataset is eliminating URLs.

Further, performing a frequency plot for each word lets us see the prevalence of stop-words (bridging words) and even a special character. Naturally, these will have to be removed too.

A graph of a number of words

Description automatically generated

In any case, performing a word count and word count analysis doesn’t allow us to make many conclusions on each tweet per sentiment, but allows us to conclude that on average tweets in the dataset were around 12 words long, some going as far as 31 words and as little as a single word.

A screenshot of a computer code

Description automatically generatedA white background with black text

Description automatically generatedA white screen with numbers and letters

Description automatically generated

Finally, analyzing the sentiment for all tweets, we can notice a much bigger presence of neutral tweets in comparison to other sentiments. This, in practice, means our model will predict neutral tweets better. To fix this, a possible step would be to use a balancing technique on the data.

A graph of a bar chart

Description automatically generated with medium confidence

# Data Preprocessing

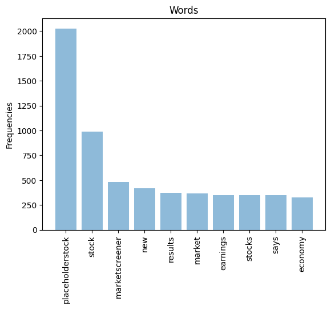
As previously seen in the previous chapter, there is a prevalence of URLs within the corpus of the project. The first step which must be undertaken is removing URLs from the corpus. We can easily achieve it using Regular Expressions (Regex) which will improve our data for model intake.

A close-up of words

Description automatically generatedA graph of a number of words

Description automatically generated

The figures above show the effect it had on our data, severely improving it. However, we now see that the most frequent words are “stop-words” (words that bridge other words) and carry no meaning. Stop-words vary on a language basis, however using the NLTK library we can remove English stop-words. We can also take the opportunity to remove other words that have no meaning like stock tickers, punctuation, and single characters.



Now it’s noticeable that the most frequent words, which were stop-words, have been replaced by more meaningful words. To gain perspective, an example is that the tweet ***“$BYND - JPMorgan reels in expectations on Beyond Meat*** [***https://t.co/bd0xbFGjkT***](https://t.co/bd0xbFGjkT)***”*** has become ***“placeholderstock jpmorgan reels expectations beyond meat”***.

This is still not enough, since more steps can be taken to further reduce and simplify our corpus. Looking attentively at the above word cloud, we can spot words that have a conjugation but could be reduced to their root form. For example, instead of “goes” we can use “go” which would still make the phrase understandable. This is called **Lemmatization** and is a good way to reduce our word quantity (which sits at 13962 unique words).

Still using the NLTK library, we can use “***WordNetLemmatizer***” to achieve this. Doing this, we have decreased from 13962 to 12756 unique words, yielding the following word cloud and word frequency plot:

A close-up of words

Description automatically generatedA graph with blue bars

Description automatically generated

We now see that the frequency plot has changed, rendering more meaningful words. However, we can (optionally) and will further try to reduce the data. We’ll now use **Stemming**, which is the process of removing prefixes and suffixes from words to cut them down to a single root. To do it, we can also use the NLTK library’s ***“SnowballStemmer”*** which has reduced the data to 10774 unique words.

Before moving on to corpus splitting, we must account for an edge case that has happened after the pre-processing and makes models like Word2Vector impossible: empty sentences. Some sentences were filtered out so it’s necessary to remove them from the data:



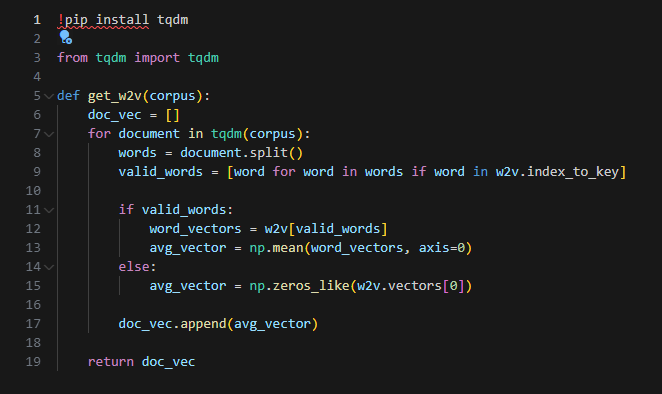
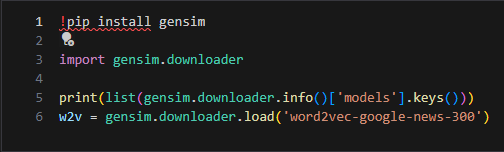
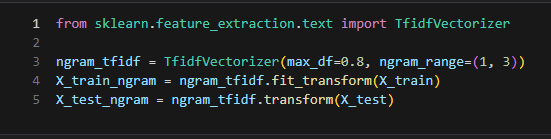
# Corpus Split

To facilitate model training and evaluation, we performed a corpus split on our dataset. The 'clean\_text' and 'label' columns were extracted from the training data, and the train-test split was executed using the “***train\_test\_split***” function. This resulted in two sets, 'X\_train' and 'X\_test', representing the text data, and 'y\_train' and 'y\_test', representing the corresponding labels. With 80% of the data allocated for training and 20% for testing, this division ensures a robust evaluation of our models' performance.

# Feature Engineering

In our feature engineering process, we opted to utilize Bag of Words, TF-IDF (Term Frequency-Inverse Document Frequency), and Word2Vec individually. Through rigorous testing and comparison, we evaluated the efficacy of each method independently, aiming to discern their respective strengths and weaknesses. This meticulous approach ensures a thorough understanding of the impact and performance of each technique in our specific context

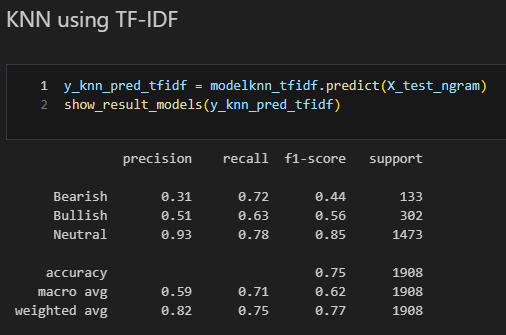
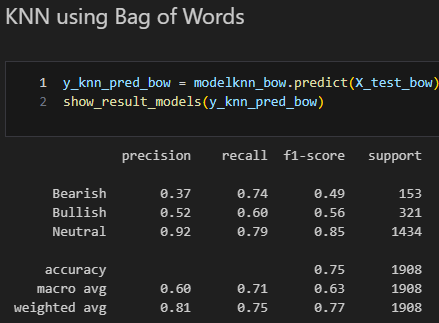
Applied code for Bag of Words, TF-IDF, and Word2Vec, respectively, follows here:

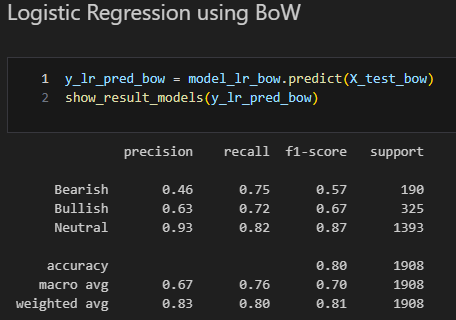
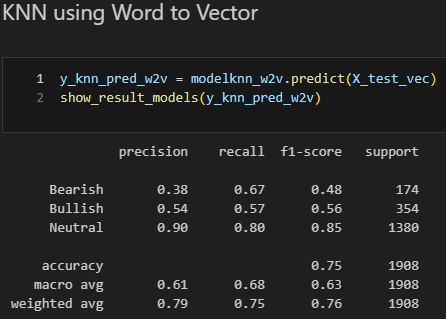


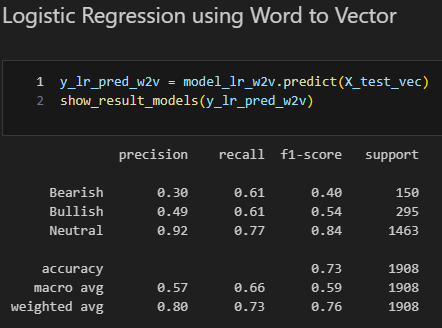
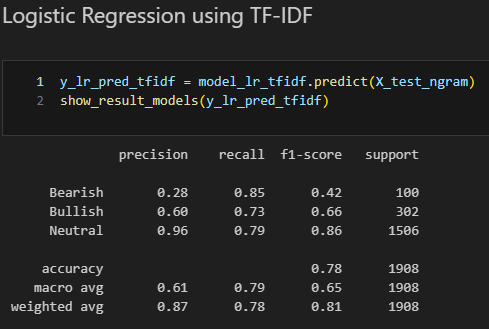
For Word2Vec we used word2vec-google-news-300 as it seemed the one most suited for our data.

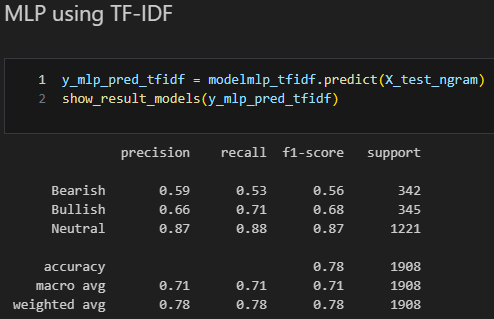
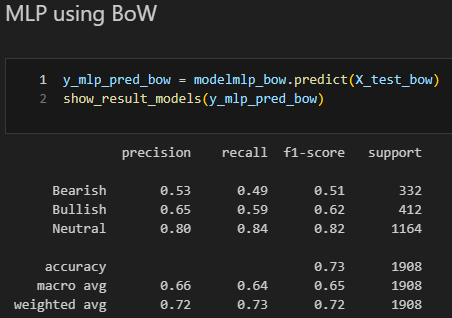
# Classification Models

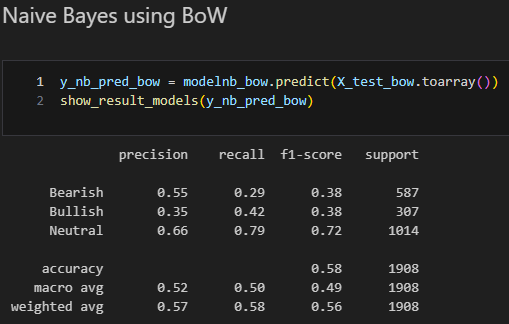
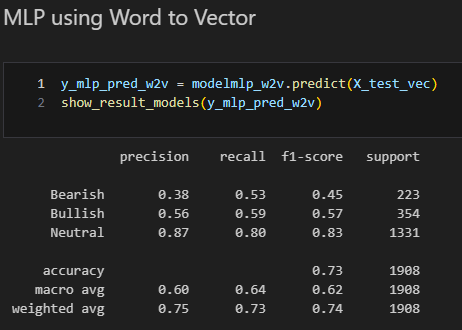
In our classification modeling strategy, we have chosen to implement a diverse set of algorithms. Specifically, we plan to leverage K-Nearest Neighbors (KNN), Logistic Regression, Multilayer Perceptron (MLP), and Naive Bayes. As an additional exploration, we will also incorporate Random Forest as a bonus, aiming to assess its performance in comparison to the other selected models.

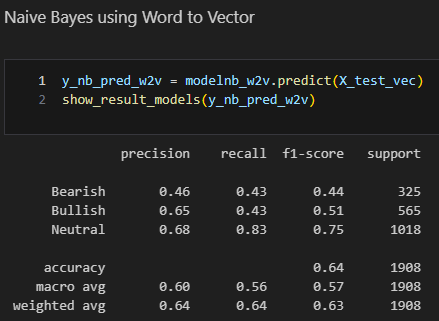
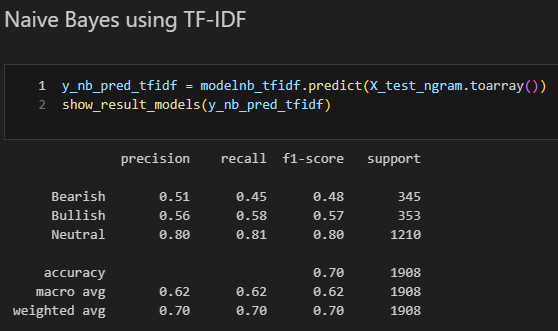


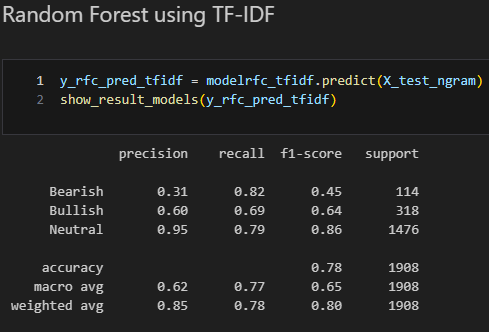
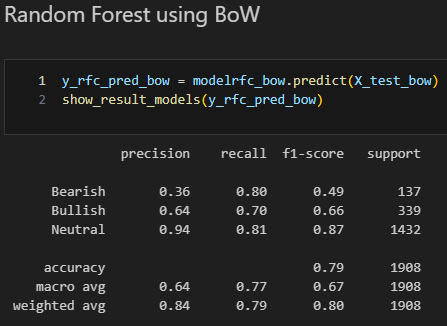


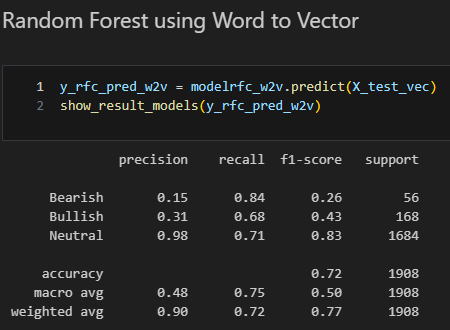












# Evaluation (Conclusion)

When comparing the performance of all models we can see that on average the Naive Bayes model obtained the lowest results this is possibly due to its assumption of feature independence, and the model's struggles with out-of-vocabulary words commonly used in tweets.

Next, we have the MLP in our second to last spot, this derives from the fact that the MLP may struggle with its limited interpretability and difficulty in capturing intricate patterns in tweets.

In the middle of the pack, we have the KNN which faces some challenges with a potential bias toward majority classes, and computational complexity, especially with large datasets. However, KNN's simplicity and ability to capture local patterns in the data could be beneficial for tweet classification, particularly in scenarios with well-defined clusters or when local relationships are essential.

In second place we have the RF, which excels in tweet classification due to its ability to handle high-dimensional and diverse features, providing robustness against overfitting and capturing complex relationships in data. However, its lack of transparency and interpretability, along with potential issues in handling extremely imbalanced datasets, could pose challenges.

In our top spot, we have the LR, which is well-suited for tweet classification tasks due to its simplicity, interpretability, and efficiency, making it computationally lightweight for large datasets. It excels in scenarios where the relationships between features and the target variable are linear. However, logistic regression may struggle with highly non-linear relationships and complex patterns found in tweets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | KNN | LR | MLP | NB | RF |
| BoW | 0.79 | 0.81 | 0.72 | 0.55 | 0.8 |
| TF-IDF | 0.79 | 0.81 | 0.76 | 0.71 | 0.8 |
| W2V | 0.77 | 0.77 | 0.76 | 0.61 | 0.78 |

As seen in the table of weighted averages of F1 scores, models using TF-IDF performed better than their counterparts, because TF-IDF considers the importance of words not just based on their frequency but also by considering their rarity across the entire dataset.

Unlike Bag of Words, TF-IDF captures the essence of term importance, making it more suitable for tasks where certain words carry significant meaning.

Additionally, compared to Word2Vec, TF-IDF does not rely on the context of neighboring words, which is beneficial for tweets with limited context.