Stock Sentiment Using Text Based Analysis

Marius Boda and Subham Das

September 20th, 2023

1 Introduction

The analysis and prediction of the stock market is a tremendously big market. Many banks and financial institutions are continuously looking for ways to help in prediction. Machine learning has been tried time and time again to 'beat' the market, but the staggering truth is that the market cannot be predicted. At its core, no individual nor institution nor machine learning module knows which way the market or a stock will trend. There are ways, however, to aid in the analysis of stocks or the grander market. These methods can provide insight into a stock and can provide investors with more information; thus allowing for many to make a more educated guess. One of these methods is analyzing the public sentiment of certain stocks based on tweets. For example, if the public sentiment of a stock is looking positive, we may see a lot of tweets talking about how individuals are optimistic for the future of stock 'ABC' or 'XYZ'. On the other hand, if sentiment is negative it is expected that certain tickers are associated with words that display a negative sentiment.

1.1 Report Outline

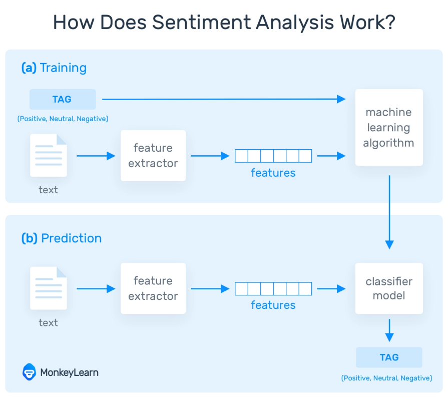
The first section, Section 1, is an introduction to the report. Section 2 goes into the details of the problem formulation. Section 3, discusses the methods used to achieve results. Results, then, are discussed in the following section, Section 4. Finally, Section 5 summarizes the report and provides a conclusion. Section 6 lists sources and Section 7 is the code appendix.

2 Problem Formulation

This machine learning task is a classification problem. Our classifier is inputted an excerpt of text, or tweets, and then outputs a result of either negative, positive, or neutral. We must be able to extract features from our tweets, we will do this by using a method called 'bag-of-words'. By doing feature engineering using this bag-of-words method, our data set will have multidimensional input. The type of data we are dealing with is categorical.

Our data set is a set of 3887 tweets containing @apple or the hashtag for Apple. This ensures that all of our tweets pertain to Apple. Each data point has text, date, judged sentiment, and other miscellaneous info. The labels for our data set will be 1, 3, and 5. 1 is a negative sentiment, 3 is neutral, 5 is for positive sentiment.

The publicly available data set was found on www.crowdflower.com, it is a data set meant for sentiment analysis of Apple tweets. The data set contains prejudged sentiment for each tweet which will make it easy to compare our ml results with what is expected.


How Does Sentiment Analysis Work?
(a) Training
TAG
(Positive, Neutral, Negative)
text
feature extractor
features
machine learning algorithm
(b) Prediction
text
feature extractor
features
classifier model
TAG
(Positive, Neutral, Negative)
MonkeyLearn

Figur 1: Problem Formulation. Source: MonkeyLearn

3 Methods

3.1 Pre-Processing

We have a collection of 3886 tweets relating to Apple. Data pre-processing included creating headers for each column and then extracting only the columns relevant. Relevant columns for this project are only the ’text’ and ’sentiment’ columns. This is because later in the feature engineering process, we will create feature vectors using the ’text’ column.

3.2 Feature Selection

Feature selection for the classification of text-based sentiment is done with a ’Bag-of-Words’ model. In essence, we create vectors for each tweet that contain a binary reading for each column. Each column is a single word in our vocabulary. To do this bag-of-words approach, we can manually write functions in order to achieve our final feature vectors. However, after trying this approach, we found that it is much more convenient to use a built in library called TF-IDF Vectorizeråhich essentially does this bag-of-words approach in only a few lines of code. We will, however, add to the code appendix a subsection that shows the bag-of-words approach written manually by defining a vocabulary function and the document term matrix function. For the code to the manual approach please refer to Section 7.3. Below is a brief explanation of the feature engineering process that our manual approach and the TF-IDF Vectorizer approach does.

The create vocabulary function takes in our filtered data set and returns the set of all words contained in all 3886 tweets. This function additionally filters out words that start with ’http://’ or ’https’ or ’@’. If need be, we can adjust and add more filters to make our ’Bag-of-Words’ model better. In order to best use our data, we have taken from the 3886 tweets a total of 1200 tweets to use in our training. These 1200 tweets are comprised of 400 of each different sentiment. So we have 400 tweets that are labeled as positive sentiment, 400 that are neutral, and 400 that are negative.

The next step in creating our feature vectors is making the document term matrix. To do this we define a new function which goes through each single tweet and converts it into a vector form. This vector form, for each individual tweet, contains ’1’ in every column where the column’s header word appears in the tweet. Similarly, the vector contains ’0’ in every column where the column’s header word does not appear in the tweet. An example is provided after the code below.

As an example, imagine that one of our tweets in our data set is: ’Hello, this is my first tweet’. The vocabulary for this space is Hello, this, is, my, first, tweet. Now, the document term matrix for this tweet, ’Hello, this is my first tweet’, is [1, 1, 1, 1, 1, 1] because every word in the vocabulary appears in the tweet. Now if we add more data to this example we will see how the document term matrices change. For example, our new data set is: 'Hello, this is my first tweet', 'Hello, I hate this'. Our new vocabulary is now, Hello, this, is, my, first, tweet, I, hate. The document term matrix for the first data point is [1, 1, 1, 1, 1, 1]. For the second data point it is [1, 1, 0, 0, 0, 0, 1, 1].

3.3 ML Model

The two chosen ML models for this project are Logistic Regression and MLP Classifier. Logistic Regression is a classification method using linear maps, with a logistic loss function, and is in section 3.6 of the course book. MLP Classifier is a classification method using linear maps + linear/nonlinear activation functions, with a logistic loss function, and is in section 3.11 of the course book.

3.3.1 Logistic Regression

Logistic Regression is a simple machine learning model that is able to take in a large amount of features. It is good for this problem because of the large amount of features that we are feeding into our model. While we do not expect a linear relationship by any means, logistic regression is still a generally good model to start with when building a classification model.

3.3.2 MLP Classifier

MLP Classifier is a good model because it can offer the advantage of capturing complex patterns and representations in data. This makes it a fairly viable option for sentiment analysis, especially when dealing with nuanced and context-dependent language such as is seen in tweets.

3.4 Loss Function

For both models, a logistic loss function is best suited. For this reason, we have used a logistic loss function to calculate the loss in our models. Logistic Loss function is typically used for multi-classification problems, such as the problem in this project. An example of why logistic loss is such a good loss function is because of its logistic nature, meaning that predictions that are far from the true label are more heavily amplified. This essentially makes logistic loss a good loss function to deal with outliers.

3.5 Model Validation

The data set has been split into 80% training set and 20% for test and validation set. We have chosen this split because of the vast amount of data that we have. We want to have a lot of data in our training set, and this is why we have it at a staggering 80%. The validation set is reserved for after the training and testing phase; in order to have an unbiased testing of the model to see how well it worked. We have used this same setup for both the Logistic Regression model as well as the MLP Classifier.

4 Results

4.1 Logistic Regression Results

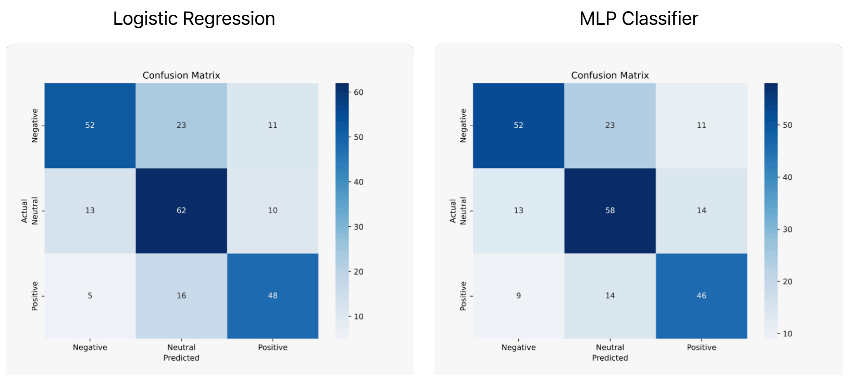
For the first model, Logistic Regression, we achieved a test accuracy of 66% and a logistic loss of 0.786. The confusion matrix for our classification is shown below.

4.2 MLP Classifier Results

For the second model, MLP Classifier, we achieved a test accuracy of 67% and a logistic loss of 1.119. The confusion matrix for our classification is shown below.

4.3 Final Chosen Method

Our final chosen method is the MLP Classifier method. The reasoning behind the selection of the final method MLP Classifier is that we believe that the MLP Classifier model provides more realistic results that seem to be more reliable. To test out our model we wrote a function called analyze_sentiment, this function takes in a written text/tweet and then outputs the predicted sentiment based on the model. For the first model, Logistic Regression, although the test accuracy and the confusion matrix looked decent, the results through this analyze_sentiment model were fairly poor. The MLP Classifier model worked much better in analyzing the sentiment of the inputted text. The test error of the final chosen method is 1.1196981461644262.



Figur 2: Confusion Matrix for Logistic Regression and MLP Classifier

5 Conclusion

To summarize, both models worked relatively well with an accuracy of 66% for Logistic Regression and 67% for MLP Classifier. Although both models only used 1200 data points the accuracy could definitely be increased by using more data. The confusion matrices for both models look pretty decent. The main problem with our model is the occurrence of the model predicting a tweet to be positive when it is in fact negative. For example, "I am going to buy apple stock. I love apple stock!", should definitely be predicted to be positive, yet our MLP Classifier predicts this sentence to be 1.0 or negative sentiment. This is fairly disappointing but it is to be expected because our data set does not include a general data set of english language sentiments.

We are satisfied with the results of the MLP classifier, however, the classifications could be improved with a larger and more equally sentiment distributed data set for training. In the future, we could add a more general text sentiment dataset that would allow our model to predict the sentiment of sentences with a more general ability.

Since our sentiment labels and features of the data set are not necessarily linearly correlated to each other, the Logistic Regression model may not work as effectively as expected. In the MLP method, the lack of a big dataset might result in an over-fitting issue since this method does not necessarily provide the desired sentiment labels when tested on datapoints outside the training data.

6 Sources

Referenser

[1] sentiment analysis https://monkeylearn.com/sentiment-analysis/

[2] crowdflower https://data.world/crowdflower/apple-twitter-sentiment

7 Code Appendix

7.1 Method 1: Logistic Regression

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import log_loss
data = pd.read_csv("Apple-Twitter-Sentiment-DFE.csv", sep=';', encoding='utf-8', header = 0)
headers = data.iloc[0]
data.columns = headers
data = data.iloc[1:]
filtered_data = pd.DataFrame()
filtered_data['selected_text'] = data['text']
filtered_data['sentiment'] = data['sentiment']
data = filtered_data
data.head(10000)

# Check unique values in the 'sentiment' column
unique_sentiments = data['sentiment'].unique()
print(unique_sentiments)
# If 'not_relevant' is present, remove those rows or replace it with a valid integer label
data = data[data['sentiment'] != 'not_relevant']

# Assuming your original DataFrame is named data
data['sentiment'] = data['sentiment'].astype(str).str.extract('(\d+)', expand=False).astype(float)

# Filter rows for each sentiment
sentiment_1 = data.loc[data['sentiment'] == 1]
sentiment_3 = data.loc[data['sentiment'] == 3]
sentiment_5 = data.loc[data['sentiment'] == 5]
# Use the total number of rows for each sentiment as the sample size
sample_size = 400
# Sample rows for each sentiment
sampled_sentiment_1 = sentiment_1.sample(n=sample_size, random_state=42)
sampled_sentiment_3 = sentiment_3.sample(n=sample_size, random_state=42)
sampled_sentiment_5 = sentiment_5.sample(n=sample_size, random_state=42)
# Concatenate the sampled DataFrames to create the new DataFrame
new_df = pd.concat([sampled_sentiment_1, sampled_sentiment_3, sampled_sentiment_5])
# Optional: Reset the index of the new DataFrame
new_df = new_df.reset_index(drop=True)
data = new_df

# Split the data into training (80%), validation (20%)
train_data, test_data, train_labels, test_labels = train_test_split(
    data['selected_text'], data['sentiment'], test_size=0.2, random_state=42)

# Vectorize the training, validation, and test data
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')

train_features = tfidf_vectorizer.fit_transform(train_data)
test_features = tfidf_vectorizer.transform(test_data)

model = LogisticRegression()

# Train the model
```

```python
model.fit(train_features, train_labels)

# Make predictions on the test set
test_predictions = model.predict(test_features)

# Evaluate the model on the test set
test_accuracy = accuracy_score(test_labels, test_predictions)
test_report = classification_report(test_labels, test_predictions)
test_logloss = log_loss(test_labels, model.predict_proba(test_features))

print(f'Test Accuracy: {test_accuracy:.2f}')
print('Test Classification Report:')
print(test_logloss)
print(test_report)

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Create a confusion matrix
conf_matrix = confusion_matrix(test_labels, test_predictions)
# Plot the confusion matrix using seaborn and matplotlib
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.savefig('1_conf.pdf', format='pdf')
plt.show()

def analyze_sentiment(input_text):
    # Preprocess the input text
    input_text = input_text.lower()
    # Transform the input text using the same TF-IDF vectorizer
    input_tfidf = tfidf_vectorizer.transform([input_text])
    # Predict the sentiment for the input text
    predicted_sentiment = model.predict(input_tfidf)[0]
    return predicted_sentiment
```

7.2 Method 2: MLP Classifier

```python
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)

# Train the model
clf.fit(train_features, train_labels)

# Make predictions on the test set
test_predictions = clf.predict(test_features)

# Evaluate the model on the test set
test_accuracy = accuracy_score(test_labels, test_predictions)
test_report = classification_report(test_labels, test_predictions)
test_logloss = log_loss(test_labels, clf.predict_proba(test_features))

print(f'Test Accuracy: {test_accuracy:.2f}')
print('Test Classification Report:')
print(test_logloss)
print(test_report)
```

7.3 Bag-Of-Words Code

```python
def create_vocabulary_list(data):
    vocabulary = set()
    for text in data['text']:
        words = str(text).lower().split()
        vocabulary.update(words)

    vocabulary = {word for word in vocabulary if "http://" not in word}
    vocabulary = {word for word in vocabulary if "https" not in word}
    vocabulary = {word for word in vocabulary if "@" not in word}
    return vocabulary
```

```
def create_document_term_matrix(data, vocabulary_list):

    document_word_counts = []

    for _, row in data.iterrows():
        document = str(row['text']).lower()
        word_counts = {word: document.count(word) for word in vocabulary_list}
        document_word_counts.append(word_counts)

    # Create a DataFrame from the list of dictionaries
    document_term_matrix = pd.DataFrame(document_word_counts)
    document_term_matrix.index = range(1, len(document_term_matrix) + 1)
    document_term_matrix = document_term_matrix.fillna(0)  # Fill NaN values with 0
    return document_term_matrix
```

```
def check_text_with_document_term_matrix(row_index):

    # Check if the specified row contains '1.0' and print the corresponding column names
    value_to_check = 1.0
    columns_with_1 = combined_df.columns[combined_df.iloc[row_index] == value_to_check].tolist()

    print(combined_df.loc[row_index, 'text'])
    print(combined_df.loc[row_index, 'sentiment'])
    print(columns_with_1)
```

7.4 Analyze Sentiment

```
def analyze_sentiment(input_text):
    # Preprocess the input text
    input_text = input_text.lower()

    # Transform the input text using the same TF-IDF vectorizer
    input_tfidf = tfidf_vectorizer.transform([input_text])

    # Predict the sentiment for the input text
    predicted_sentiment = clf.predict(input_tfidf)[0]

    return predicted_sentiment
```