Financial Sentiment Analysis
ECE 4424 Final Project

Matthew Rajan, Nihal Mitta, Jack Orr

5/4/2023

Github

Abstract

Financial sentiment analysis is a new and challenging task due to the niche vocabulary special to the field. General-purpose sentiment models are not effective enough because of the specialized language used in a financial context. In this paper, we will discuss 4 possible machine-learning methods and algorithms and test their effectiveness in classifying financial news statements as positive, negative, or neutral. To see if trading based on financial news is a viable strategy, we will back-test our models and see their performance against the S&P 500.

Keywords and Phrases: Sentiment Analysis, Quantitative Analysis, Logistic Regression, Multi-Layer Perceptron, Recurrent Neural Networks

1 Introduction

Within just the past 10 years, technology has been rapidly increasing and has continually been infiltrating all fields and industries. In recent years, the finance industry has welcomed this change in the status quo and has been trying to leverage technology for its benefit. Quantitative Analysis is a new and growing field of research in which financial models are being implemented and paired alongside machine learning methods and algorithms. The use of Quantitative Analysis in financial firms is constantly growing as the opportunity to outperform competitors and generate alpha is incredibly appealing.

When trying to learn how to trade in a market, what is one of the first things you do? For most people, checking the news is absolutely vital to conducting any type of trade or analysis within a certain market. Financial news is one of the most important factors to look at when conducting a trade as that is how information is gained publicly. Although now in modern times many financial institutions seek to get news and information straight from the source, the market still reacts to major news articles and headlines. One prime example are the FOMC meetings where the FED's interest rate targets are announced. After the news hits, major movements can be seen in the gold, silver, bonds, and even stock markets.

Now that we have seen the importance of news on financial markets, how can we leverage this? One way we can do this is through the use of Sentiment Analysis. Sentiment Analysis has been a growing field of research with the large increase in data available as well as the advancements of Natural Language Processing (NLP) techniques. Outside of the financial view, NLP and Sentiment Analysis are extremely useful in finding the connotations behind words and the underlying emotion. Sentiment Analysis is being used for creating chatbots, virtual assistants, and recommendation systems. Most noticeably with the release of ChatGPT, we can see the immense possibilities of NLP and Sentiment Analysis.

Our hypothesis is that we can utilize Sentiment Analysis models trained on financial news statements and headlines, determine the trading day's sentiment, and then trade on whether or not the day's sentiment is positive, negative, or neutral.

2 Data Collection and Pre-Processing

2.1 Datasets

To obtain a sizable dataset of financial news article titles with labeled sentiments for our project, we turned to the Financial Sentiment Analysis[1] and Financial Phrase Bank v1.0[2] datasets. These datasets consisted of approximately 5,000 statements each, with each statement labeled as positive, negative, or neutral. However, the Financial Phrase Bank dataset had multiple labels per statement, resulting in each statement being categorized into one of four categories (100 %, 75 %, 66 %, or 50 %) based on the degree of agreement among the labels. After exploring all possible combinations of datasets, we concluded that the Financial Sentiment Analysis and the 75 % agreement labels from the Financial Phrase Bank were the most robust and suitable for our project.

2.2 Bag of Words Vectorization

In order to process our data and make it usable for training, we need to encode our financial news article titles. One effective method of encoding is through the use of a bag of words vectorization technique. This approach involves counting the frequency of each word in the dataset and representing it in a structured format, making it useful for our models. By utilizing this technique, our models can identify words that are more common in positive, negative, and neutral statements and use them to make accurate predictions.

2.3 Data Cleaning

Now that we have gathered all the necessary data for our sentiment analysis project, the next step is to clean the information before we can effectively use it. As we are utilizing a bag of words vectorization method, it is essential that we remove extraneous words, symbols, and random characters from the data. Doing so will enable us to obtain accurate frequency information for each word in the dataset.

We have also made the decision to convert all text to lowercase and remove any non-English words and numbers from the dataset. This is crucial as we do not want a misrepresentation of the significance of words that are typos or give a sentiment to a numerical value. Numbers inherently do not carry a good or bad sentiment, and so including them in the dataset would only serve to skew the results of our sentiment analysis models.

For example, imagine a sentence that contains the phrases "a loss of 100 dollars" and "a gain of 100 dollars." If we were to include the number "100" in our bag of words frequency count, our models may misinterpret the significance of this number and assign a false sentiment to it in either situation.

Therefore, by cleaning our data and removing extraneous words, symbols, and numbers, we can ensure that our sentiment analysis models will provide accurate and meaningful results. This crucial step will allow us to effectively analyze the sentiment of financial news and make informed trading decisions based on the sentiment of the day.

2.4 Pre-Processing

Now that our data has been cleaned and prepared for our models, the next step is to pre-process it in a way that our models can handle. One crucial step is to create a mapping for our sentiments. In the original dataset, the sentiments were labeled as

positive, negative, or neutral, and so we can convert them to 2, 0, and 1, respectively. This allows our models to work with numerical values rather than string labels.

The next step in pre-processing our data is to convert our sentences into a bag of words model. This can be done quite easily using sklearn's CountVectorizer function, which converts text into a sparse matrix of token counts. This process involves extracting the individual words from the sentences and representing them as features in our dataset.

Once we have our bag of words model, we need to split the data into training and testing sets. We used sklearn's train_test_split function to perform an 80/20 split of our data, separating the data and their labels into their respective sets.

To further improve the performance of our models, we normalized our bag of words vectorization. This helps in cases where our models use gradient descent, as normalization ensures that the values are in a range that allows for efficient optimization.

2.5 Data Loader

In order to ensure that our financial sentiment analysis models can be effectively implemented using PyTorch, one crucial step is the creation of data loaders. These data loaders play a pivotal role in loading and batching data samples in an efficient manner, allowing us to feed these samples into our neural networks.

Data loaders are responsible for grouping data samples into batches of specified sizes, which can then be used to perform matrix operations on multiple samples in parallel. This is essential for ensuring that our models can handle large datasets with ease and are able to process the data quickly and accurately.

To create a data loader, we first need to convert our training and testing dataframes into the PyTorch Dataset class. This class provides an interface for loading and manipulating data samples, which we can then use to create our data loaders. By utilizing the PyTorch Dataset class, we can easily manipulate our data and prepare it for use in our sentiment analysis models. Once we have converted our data into the PyTorch Dataset class, we can create our data loaders by specifying the batch size and any other relevant parameters. This allows us to control how the data is loaded and how it is processed by our neural networks.

Overall, the creation of data loaders is a critical step in preparing our financial sentiment analysis models for use in PyTorch. By effectively batching our data and performing matrix operations in parallel, we can ensure that our models are able to process large datasets quickly and accurately.

3 Models

3.1 Logistic Regression

Now that we have pre-processed our data, it's time to move onto the modeling stage. Our first model of choice is logistic regression, a simple yet powerful algorithm that is commonly used for binary classification tasks. In our case, we will modify the algorithm to perform multi-class classification. The logistic regression model works by minimizing the cross-entropy loss, which measures the difference between the actual probability distribution and its predicted probability distribution. The model accomplishes this by using gradient descent to iteratively adjust the weights and biases of the input features. To calculate the predicted probabilities, the model employs the use of the softmax activation function. This function ensures that the predicted probabilities sum to 1, which is necessary for multi-class classification. The output of the model is a probability distribution over the three possible sentiments: positive, neutral, and

negative. By selecting the class with the highest probability, we can make our final sentiment prediction.

3.2 Multi-Layer Perceptron Algorithm

Our upcoming model will be utilizing the Multi-Layer Perceptron (MLP) Algorithm. This particular algorithm operates using the MLP update rule, which involves the following steps:

1. Utilize a linear predictor to forecast the classification using the current weights.
2. If the prediction is accurate, no further action is necessary, and your model is considered correct!
3. If the prediction is incorrect, then the features of the data should be subtracted from the weights for the predicted class, and the features of the data should be added to the weights for the correct class.

This iterative process continues until the model reaches convergence, which happens when the loss function is minimized.

3.3 Multi-Layer Perceptron Network

We'll be implementing another MLP process called the Multi-Layer Perceptron Network. This model is also known as a "Feedforward Neural Network" or an "Artificial Neural Network," and is the foundation of modern neural and deep neural networks. The MLP network computes predictions by creating hidden layers, connected through activation functions. Our implementation uses just one hidden layer for simplicity. The model employs backpropagation with gradient descent to update the weights for every epoch until convergence.

3.4 Recurrent Neural Network

For our final model, we drew inspiration from the MLP Network and decided to implement the Recurrent Neural Network (RNN). Like the MLP Network, the RNN is a neural network, but it has loops that allow previous information to be stored within the network. This unique feature enables RNNs to use past experiences to inform decision-making for new events. RNNs have shown exceptional performance on sequential data, making them ideal for NLP applications. During training, the same set of weights is used for all timesteps, and the model is updated accordingly. This approach allows current inputs to depend on previous states, enabling the model to capture dependencies.

3.5 Model Results

After training and testing each of the models on our data, we were able to produce pretty great results. The model accuracies are listed below:

- ◇ Logistic Regression: 82.6789 %
- ◇ MLP Algorithm: 78.0527 %
- ◇ MLP Network: 82.3815 %
- ◇ RNN: 81.0884 %

We were happy with these results, especially considering the fact that random guessing would only result in a 33 % accuracy. Overall we can see that the Logistic performed the best, with the MLP Algorithm lagging in the back by only around 4 %. With these results, we are confident to try testing our models with new data and implementing our trading strategy!

4 Back Testing

Now that we have our trained models, let's see how we can use them in a trading strategy. For our backtesting we will be comparing our model's performance to a naive strategy of just holding one stock of S&P 500.

4.1 News Collection

To conduct a backtest on our strategy, we needed to gather news articles over a period of time to provide input for our model to classify. We opted for using an RSS Feed Aggregator, which is a straightforward method to obtain news articles. We located an RSS Feed for the S&P 500 and saved the news articles to a csv file. By using this single feed, we acquired roughly 100 news articles that spanned from December 2019 to the present time.

4.2 Model Predictions

Having obtained our data, the subsequent stage is to classify the news articles utilizing our model. Similar to the process followed during the model training phase, the data must be vectorized and scaled. After transforming the data, we then proceeded to pad it with zeroes so that the feature size of our new input corresponds to the feature size utilized during the model training phase. With all of that done we have our predictions and it's time to test!

4.3 Trading Strategy

For our strategy we will make it simple with just a few rules:

- ◇ We will start off holding 1 share of S&P 500 in our portfolio
- ◇ Our portfolio can hold a maximum of 1 share and a minimum of 0 shares
- ◇ If the predicted label is positive and the portfolio is empty, buy 1 share
- ◇ If the predicted label is negative and the portfolio has 1 share, sell 1 share
- ◇ If the predicted label is neutral or the above conditions do not apply, do nothing

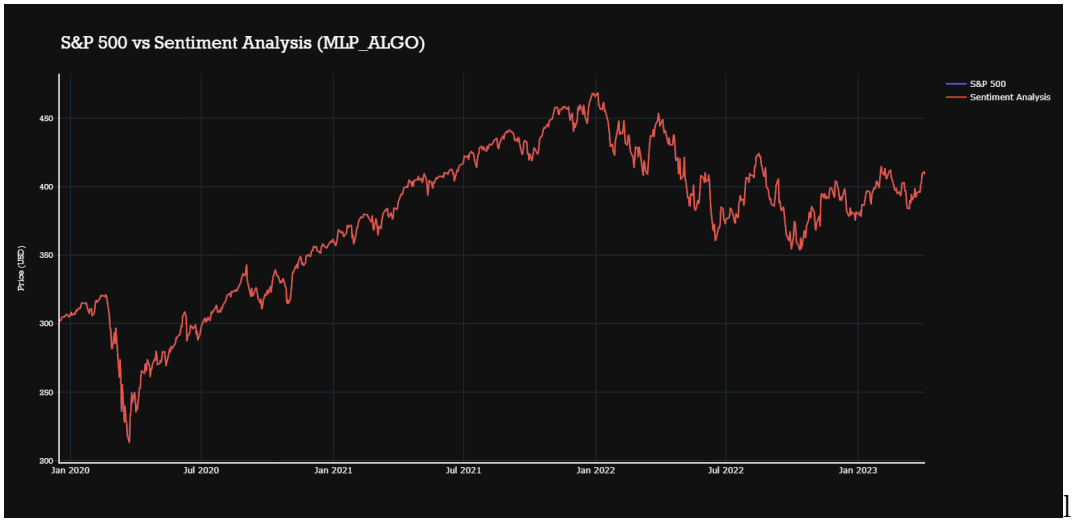
With this strategy, the goal is to hold a cash balance during bad periods and minimize our portfolio loss, and capitalize on the uptrend in good periods.

4.4 Back Testing Results

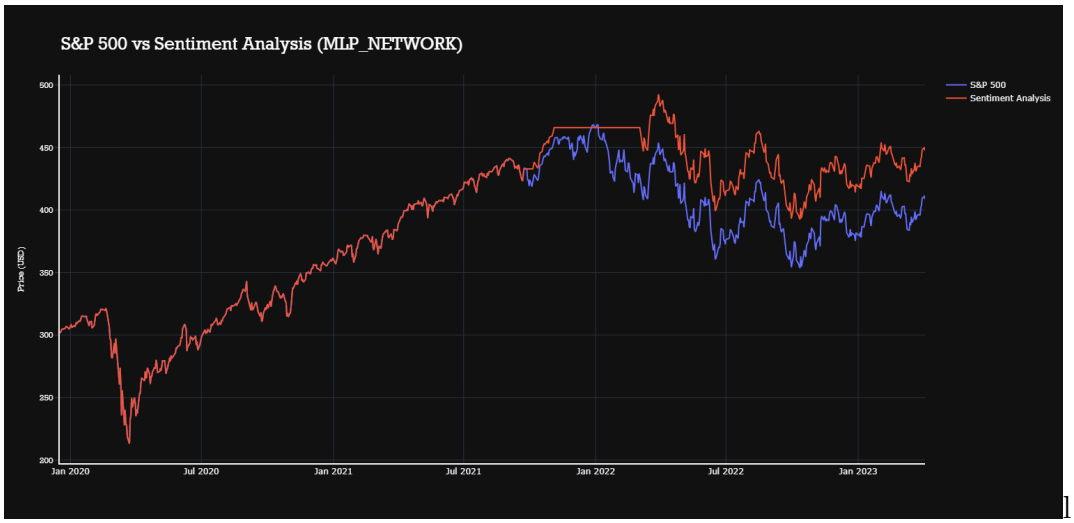
After implementing our strategy using our predicted labels, we were able to generate the following results:

◇ Logistic Regression:

◇ MLP Algorithm:



◇ MLP Network:



◇ RNN:

After analyzing the charts above, we can observe that the models performed exceptionally well. Among the models, the RNN showed the highest performance, while the MLP Algorithm had the lowest performance and held onto one share throughout the entire period.

However, we were surprised by the fact that the MLP Algorithm did not make any trades during its time. After examining the predictions generated by the model on the news articles, we discovered that no negative classifications were predicted. Further investigation revealed that the MLP Algorithm had only a 48 % accuracy rate on negative samples, while it had an accuracy rate of 83 % and 82 % on neutral and positive classifications, respectively. This poor performance on negative data is likely due to the lack of negative samples in our training and testing data, which resulted in the model's inability to sell.

Despite this limitation, we are delighted with the models' performance and believe that their success in this limited scope justifies further research and exploration!

5 Conclusion and Future Work

This project was a great success! Despite the time constraints, we set ourselves a lofty goal, and the hard work paid off with exciting results. However, there is still room for improvement. One area for improvement is hyper-parameter optimization. We did not conduct much experimentation with batch sizes, layers, epochs, and other parameters, so exploring different options could potentially improve the accuracy of the models.

Another way to extend this project is to try different types of encodings for the sentences. While the Bag of Words approach worked well, using embeddings such as Word2Vec or graph neural network encodings could yield better results. Additionally, creating other types of models could be worthwhile. For instance, we found that LSTM models could be effective for sentiment analysis, so experimenting with other models may lead to even better performance.

References

[1] Financial Sentiment Analysis, https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

[2] Financial PhraseBank v1.0, https://www.researchgate.net/publication/251231364_FinancialPhraseBank-v10