MAIS202 Deliverable 2: Stock Market Predictor

Deepak Singh

1. Problem Statement

I want to build a model that predicts stock market trends according to sets of news articles with a web application that will allow users to make their own predictions.

2. Data Processing

My dataset is composed of 1989 days of the top 25 news headlines with its corresponding stock trend. Thus, there are a total of 49,725 headlines. Since, my dependent (response) variable has already been preprocessed with labels for indicating if the DIJA increased or decreased (1, 0), not much additional work was needed. For the news articles, I merged all top 25 news headlines into one column and used regex to only keep individual words. After exploring the actual daily news headlines, I noticed that there were several wording errors and unnecessary characters. I used sklearn's CountVectorizer to vectorize my data by count with a vocabulary of the 30,000 most frequent words. I took advantage of the CountVectorizer parameters stop_words to select all English words and used its default token_pattern parameter to remove all one to two lettered words.

The chosen method made it easy and computationally inexpensive to vectorize the data. Subsequently, this allowed me to continuously fine-tune hyperparameters (i.e. max_features). However, this came at the cost of losing the sentimental value of each sentence. Especially since I merged the top 25 articles into one column, using sentimental analysis would prove as ineffective.

3. Machine Learning Model

For this project, I focused on the Gaussian Naïve Bayes model from the sklearn package due to its simplicity and low computational costs. I decided to additionally try out sklearn's Random Forest and Logistic Regression models to see if they could yield better results. Furthermore, after some discussion with Cheng from the MAIS exec team, I decided implementing a deep learning model, namely the Long Short-Term Memory model from the Keras framework in the later stage of the project to see if it could yield any improvements from my main model. For the preliminary stage, I only decided to optimize the first three models and chose the best one.

4. Preliminary Results

For this stage, as mentioned, I tested three models: Logistics Regression, Gaussian Naïve Bayes and Random forests. With some tweaking, all received strong training accuracies in the range of 95% and 100%. However, all three testing accuracies for each model ranged around 45%- 52% which shows strong signs of overfitting. The Gaussian Naïve Bayes model was able to get the highest testing sample accuracy of ~52% while having a 96% training accuracy. Both Logistics Regression and the Random Forest model had 100% training accuracies and testing accuracy of around 46-47%, showing heavy overfitting (Figure 1). Thus, we can observe that the Gaussian Naïve Bayes model yields the least amount of overfitting compared to the other models.

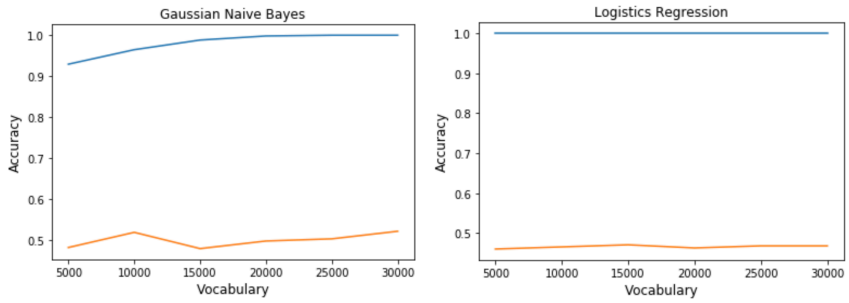
Figure 1: Optimal Accuracies per Model.

	Logistics Regression	Gaussian Bayes	Random Forest
Training	1.0	0.9646182495344506	1.0
Testing	0.4708994708994709	0.5185185185185185	0.46296296296296297

The only adjustable parameter was the max_features parameter in the CountVectorizer object which determines the maximum vocabulary size for vectorization. While there were adjustable hyperparameters in the Random Forest model, after 10 fine-tuning attempts, the model yielded no significant improvements compared to its optimal result in Figure 1. Thus, I decided to disregard it in this analysis because it was too complex and time consuming to fine tune each RF parameter.

Increasing the vocabulary size from 5,000 to 30,000 shows improvements within the Gaussian Naïve Bayes model with high increases in the training accuracy and a small increase in the testing accuracy. However, for logistics regression, we did not notice much improvement for both sets (Figure 2). Thus, I will choose the Gaussian Naïve Bayes model with a vocabulary size of 30,000 words as my main model of choice.

Figure 2



5. Next steps

I will continue with the Naïve Bayes model because it has the least signs of overfitting, and subsequently implement forms of L1 and L2 regularization to minimize overfitting. If testing accuracies do not further increase, I will attempt to increase my sample size with more recent news articles from the past year via web scrapping with the goal of adding 1,000 news headlines. Gathering this new set of data would be time consuming and difficult but could prove as beneficial to my model.

Additionally, I will simultaneously start implementing deep learning with the Sequential model from Keras and implement the built-in Long Short-Term Memory model. My goal is to achieve a testing accuracy of 60%. However, this will be computationally expensive which is why I would need to run this model on google Collab for long periods of time.

At the end, I will implement a web application with React and Flask which will allow users to input recent news article headlines The web app will output its prediction towards the stock market (and possibly allow us to become millionaires!).