IDS 703 Final Project Report

Alisa Tian, Jenny Shen

I. Introduction

The movement in the stock market is affected by lots of factors including inflation and interest rates, international events, and financial reports directly and indirectly. The latest revealed news and information could impact investors' demand for stocks and thus become influential to the stock price. Therefore, in this project, we focus on a document classification NLP problem. More specifically, we conducted sentiment analysis by using daily news headlines to predict stock market movement. Two stock market prediction models were established and compared as a result.

II. Data

2.1. Data Collection

We obtained the "Daily News for Stock Prediction" dataset from Kaggle to conduct our analysis. The top 25 news headlines for each day were crawled from Reddit WorldNews Channel as the columns of the dataset. The data ranges from 2008-06-08 to 2016-07-01, and the Dow Jones Industrial Average (DJIA) stock index was obtained from Yahoo Finance to assign labels for news headlines on a specific date. When the DJIA adjusted close value increases or stays the same, label "1" is assigned. In comparison, the label "0" is assigned when the DJIA adjusted close value decreases on that day. From Figure 1 below, we can find the distribution of sentiments in the dataset is relatively balanced (approximately 900 examples with label "0" and 1100 examples with label "1") which is suitable for algorithm training.

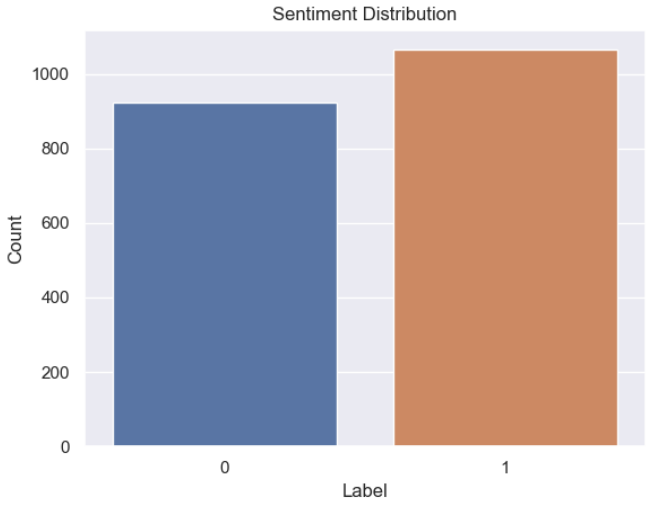

Figure 1. *Sentiment Distribution in the Dataset*

2.2. Data Preprocessing

The non-alphabetic characters were replaced with space by using regular expressions (regex) to remove punctuations and special characters. In addition, we converted the capital letters in news headlines to lower cases. Stopwords were not removed in this case since new headlines tend to be condensed and every word may carry an important meaning. Since we hope to use all 25 news headlines in the analysis, we combined all the news headlines on each day into a paragraph. The data from 2008-08-08 to 2014-12-31 was selected as the training set, and the data from 2015-01-02 to 2016-07-01 was selected as the testing set. Therefore, this train test split is roughly an 80%/20% split.

III. Modeling

3.1. Generative Model

For the generative probabilistic model, we chose the Naive Bayes method. Naive Bayes is a supervised method based on the assumption that all pairs of features are conditionally independent given the class label.

According to Bayes' theorem, the probability given class variable *y* and dependent feature vector *x*1 through *x*n

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

The relation could be simplified to:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

regarding the assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

Since the denominator is a normalization constant,

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

The best class in the Naive Bayes classification is the maximum a posteriori (MAP) class:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y).$$

- The estimation of the prior P(*y*) is the relative frequency of class *y* in this training set.
- For Multinomial Naive Bayes, the conditional probability *P* (*xi*|*y*) is the relative frequency of term *xi* in documents belonging to class *y* with smoothing priors *a*.

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Therefore, we implemented Bag-of-Words for the training set using CountVectorizer to convert a collection of text documents to a matrix of token counts by making each unique word a column and counting the number of times each word appears (the row values). Please find an example of such matrix below:

	aa pakistan	aaa credit	aaa rating	aaa seal	aaa to	...	zuyevo rice	zwanziger has	zweimal hitler	zygi naval	zyklon the
0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
1858	0	0	0	0	0	...	0	0	0	0	0
1859	0	0	0	0	0	...	0	0	0	0	0
1860	0	0	0	0	0	...	0	0	0	0	0
1861	0	0	0	0	0	...	0	0	0	0	0
1862	0	0	0	0	0	...	0	0	0	0	0

Figure 2. Example of the matrixes converted by Bag-of-Words

We then apply the Multinomial Naive Bayes classifier (with smoothing parameter defaulted as 1) by initializing the model, fitting the train data, making predictions of the test data, and computing evaluation metrics to measure the performance of this classification task.

3.2. Generating the Synthetic Data from Generative Model

We could use the generative model Naive Bayes to generate synthetic data since Naive Bayes tends to figure out the distribution that produced the data as the input. We first get all pairs of features from the **vectorizer.get_feature_names_out()** and save them into an array named vocab.

In training, the Naive Bayes model learns estimates for the conditional probabilities with which each value of each feature variable occurs given a class label. These conditional probabilities can be denoted $P(xi|y)$. Then, we could obtain the empirical log probability of pairs of features given a class by using the **feature_log_prob_**. By exponentiating the log probabilities, we can get the actual probability score of a feature in a specific class, which is the same as the below calculation:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Then, we could generate data by randomly selecting feature values according to the conditional probability distributions learned by the Naive Bayes model. In this case, we used **random.choices** to randomly select features from the previously defined "vocab" array given the probability of pairs of features of a given class, with an expected total length of the returning pairs of features specified (the length would be similar to the length of the combined news headlines in the training set). All the returning features are combined together to form a piece of synthetic news headline data. This process was repeated 994 times for each class respectively to achieve a synthetic data size that is similar to the "real" data. By the Law of Large Numbers, the synthetic data generated by this process will approximately mirror the probability distributions learned by the Naive Bayes model.

However, although the synthetic data mimic the real-world data pattern and approximate real-world data, it is worth noting that the generated data could be hardly understandable text with bias.

3.3. Discriminative Model

We chose the Long Short Term Memory Model for the discriminative model and used the Keras library in the Tensorflow package. LSTM is one of the most popular models for sentiment analysis, which performs well when analyzing time series data.

The steps are:

1. Splitting the training and testing dataset, Tokenizing, and Padding
2. Training the Network using the LSTM model
3. Fitting the model and Batching
4. Testing (on real-world test data and synthetic test data) and Evaluating the Performance

For the initial steps, we tokenize the sentences and perform padding. The Keras Library is used for preprocessing. The maximum number of words is set to be 4000 based on word frequency. The document is padded with a length of 10. The vocabulary size is set to be the length of the "vocabulary to integer" mapping dictionary +1. Specifically, "+1" is for the 0 embeddings.

The architecture of the LSTM model:

1. Embedding Layer
2. Spatial Dropout Layer
3. LSTM Layer
4. Sigmoid Layer

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 10, 128)           4147712

 spatial_dropout1d_7 (Spatia  (None, 10, 128)          0
 lDropout1D)

 lstm_7 (LSTM)               (None, 128)               131584

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 4,279,425
Trainable params: 4,279,425
Non-trainable params: 0
_________________________________________________________________
```

Figure 2. LSTM Model Summary

Embedding Layer: Word embedding is used where words can be encoded to be vectors. The distance in the vector space indicates the similarity of different words. The embedding vector length is 128 and the size of the vocabulary(for real-world data) is 29120. In the dataset, the lengths of sequences are different, and we need to vectorize all the input sequences to be the same length. The input sequences are padded with a length of 10.

Spatial Dropout Layer: We used the spatial Dropout layer here to avoid overfitting the model. The inputs of this layer and output from the embedding layer we built are removed by certain probability. Here, we set the dropout value as 0.3, which means that 0.3 of them are dropped randomly. By doing so, our networks' nodes tend to be robust and decrease the chances of overfitting.

LSTM Layer: There are 128 units in this layer. We avoided choosing large output space dimensionality to prevent overfitting and increasing the training time. On the one hand, the dropout rate is used for the input and output of the model. On the other hand, the recurrent dropout rate is for the recurrent state, which influences the model's "memory". Both the dropout and recurrent dropout rates are set to be 0.3 here.

Sigmoid Layer: For the last layer, we added the densely connected layer for the activation function, which is sigmoid. When compiling the mode, we used the Adam optimizer as well as the binary cross entropy loss function. The binary cross entropy loss function is a widely used loss function for these kinds of classification issues.

IV. Results

4.1. Real-World Data

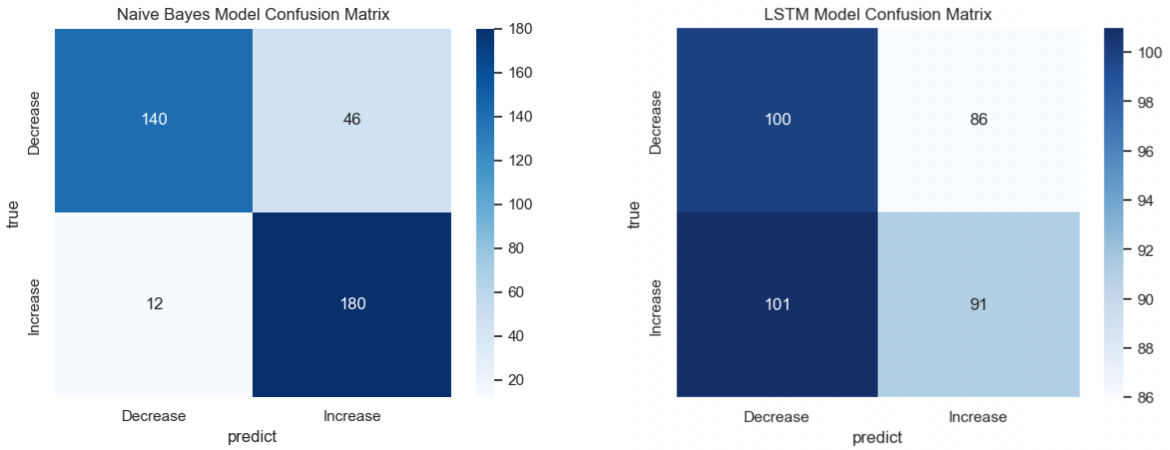

Figure 3. *Confusion Matrix of Naive Bayes Model and LSTM Model for Real-world Data*

The train/test split for the real world was to select the data from 2008-08-08 to 2014-12-31 as the training set, and the data from 2015-01-02 to 2016-07-01 as the testing set to obtain roughly an 80%/20% split. From the two confusion matrices above, we can calculate that the accuracy for the Naive Bayes model is 0.8466, whereas the LSTM model has an accuracy of 0.5053. The AUC of the Naive Bayes model is 0.8451 while the AUC for the LSTM is 0.5058. Therefore, the naive Bayes model outperformed the LSTM model for "real data".

Since the Naive Bayes classifier assumes the presence of a particular feature in a class is not related to the presence of any other feature, when the assumption holds well, the classifier has decent performance and requires relatively little data to work. In this case, the Naive Bayes classifier performed relatively poorly when there were positive and negative sentiments involved in the same new headline text. For example, the Naive Bayes classifier wrongly classified "Rich Getting Richer at the Expense of the Poor, Oxfam Warns" as positive news, and wrongly classified "Australia's gun laws stopped mass shootings and reduced homicides, study finds." as negative news. Although there were words "shootings" and "homicide" in the news headline, there were also "stopped" and "reduced" indicating positive sentiment. Therefore, it is crucial to consider the implication and context of the text in a classification task.

We then examined the situations when LSTM performs badly and checked the situations when the LSTM model labels the headlines as "decreasing" when the actual stock price is increasing. Some cases are: "Ontario is putting an end to coal-burning power plants", "Charlie Hebdo now has 25 times as many subscribers as before deadly attack". The LSTM model might capture specific negative words such as "end" and "deadly attack", and label

them as “decreasing”. This ambiguity and the difficulty to predict the stock price based on news headlines make LSTM models hard to label the headlines correctly.

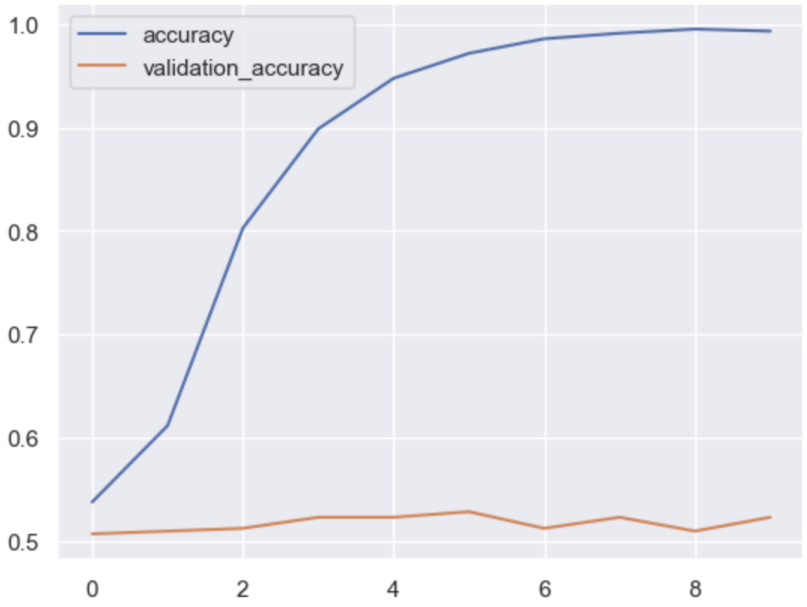

Figure 4. *Accuracy of LSTM Model for Real-world Data*

Figure 4 shows the accuracy of the model for real-world data. The accuracy of the training dataset continues to increase but that of the validation dataset fluctuates around 0.5. This indicates the problem of overfitting. This can be caused by the limitation of this dataset that the number of entries is limited, and the limited association between news headlines and changes in stock price. To better solve the issue of overfitting, future work includes using varying hyperparameters for better tuning.

4.2. Synthetic Data

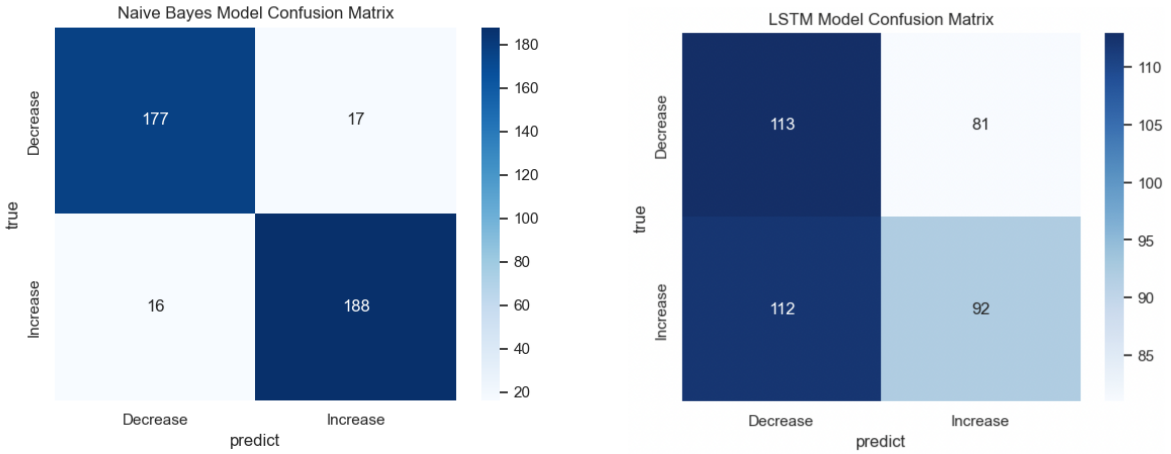

Figure 5. *Confusion Matrix of Naive Bayes Model and LSTM Model for Synthetic Data*

The train/test split for the real world was to select the data from 2008-08-08 to 2014-12-31 as the training set, and the data from 2015-01-02 to 2016-07-01 as the testing set to obtain roughly an 80%/20% split. From the two confusion matrices above, we can calculate that the accuracy for the Naive Bayes model is 0.9171, whereas the LSTM model has an accuracy of 0.5151. The AUC of the Naive Bayes model is 0.9170 while the AUC for the LSTM is 0.5167. Therefore, the naive Bayes model outperformed the LSTM model for "synthetic data".

The results are within our expectation since the synthetic data is generated from the generative model with the assumption of the Naive Bayes model satisfied. Therefore, the accuracy of the Naive Bayes model tends to be higher.

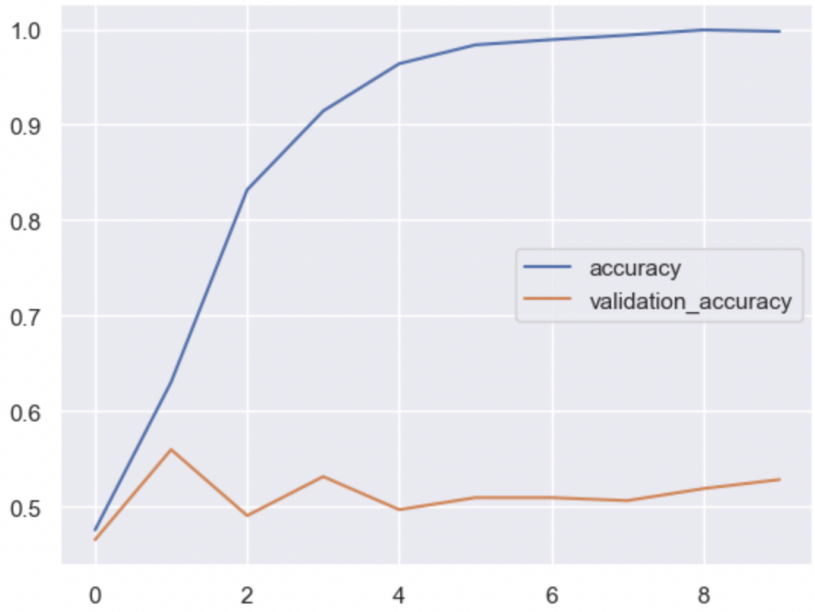

Figure 6. *Accuracy of LSTM Model for Synthetic Data*

For the LSTM model, the synthetic news headlines generated by LSTM are obscure and hard to understand. These are some of the examples: "six tons islamic state running philippines hamas leader blew hundreds are around interrogator" and "judge suspends protection nasa circumcision of food and expected year cost satellite colom as yorker bin". Most of the headlines are difficult to understand and some even do not make sense logically.

4.3. Model Comparisons

From our results of two models:

- The model accuracy of the Naive Bayes model is much higher than the LSTM model. For real-world data, the accuracy of the Naive Bayes model is 0.3413 higher than that of the LSTM model. For synthetic data, the accuracy of the Naive Bayes model is 0.402 higher than that of the LSTM model.

- According to the computation time we calculated, the training of the LSTM model could take about 10+ seconds on the CPU. By contrast, it only takes the Naive Bayes method around 2 seconds in the training process.
- The number of parameters that we can alter for the Naive Bayes model is very few, in comparison to LSTM where lots of fine-tuning can be performed.

On the one hand, the Naive Bayes model performs well when the assumptions of Naive Bayes are satisfied. One of the assumptions of the Naive Bayes model is that the features are independent of each other. When this assumption is satisfied, the performance tends to be good even when the dataset is small. The implementation of the Naive Bayes method is relatively more straightforward than the LSTM model. Scaling Naive Bayes implementation to large datasets having millions of documents is relatively easy whereas for LSTM we certainly need plenty of resources. The Naive Bayes model is **generative** as it aims to find the pattern of how data is generated (distribution) during the training process.

On the other hand, LSTM performs well for time series data since it can maintain long-term memory well. The performance of the LSTM model, however, largely depends on the dataset itself, resulting in low accuracy. Since our dataset is not large and the number of features is limited. The limitation of the dataset can also lead to overfitting issues. As we have noted in our model accuracy tables, LSTM models tend to have overfitting issues. In addition, the process of LSTM is less transparent than the Naive Bayes model, which is like the black box. It is hard to interpret what exactly happens in the network. The LSTM model is **discriminative** since it aims at finding the difference between the examples of "1" and "0" for classification.

V. Conclusion

For this project, we trained both the Naive Bayes model (the generative probabilistic model) and the LSTM model (the discriminative neural network) to perform sentiment analysis to predict stock market movement using the daily news headlines dataset. These two approaches are applied to both the real-world data and the synthetic data generated from the Naive Bayes model. In conclusion, in both "real" data and synthetic data cases, our Naive Bayes model performs better than the LSTM model in terms of higher model accuracy and lower model complexity in terms of time and space.

VI. Limitation and Future Work

The accuracy of the LSTM model is not satisfactory. One of the reasons is the limitation of the dataset itself - the headlines were collected from the news all over the world, while the DJIA Adj Close value is the stock market index only in the US. In addition, since the dataset is not large enough with only 1989 entries, the sentiment analysis on a relatively small scale in this case also results in the low accuracy of the LSTM model. Future work can be extended by using various hyperparameters, expanding the dataset size, and focusing the domain of the dataset on the financial news in the United States.

References

C.D. Manning, P. Raghavan and H. Schuetze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265. Retrieved Dec 11, 2022, from https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html

Scikit-learn: Machine Learning in Python, Pedregosa, *et al.*, JMLR 12, pp. 2825-2830, 2011.

Sun, J. (2016, August). *Daily News for Stock Market Prediction*. Kaggle. Retrieved November 16, 2022, from https://www.kaggle.com/aaron7sun/stocknews