News Sentiment Analysis for Stock Data

Group 32: Felix Chung, Changyu Wu, Hanyu Zhang

1. Introduction

As one of the most important financial security, stock is a great option for financial investment. However, predicting the trends of stocks is not an easy task. Stock price is mainly determined by a group of buyers and sellers that construct the market, and many of these buyers and sellers are ordinary people who trade irrationally. Most ordinary people will not conduct full financial analysis of the company and trade security based on psychological and social factors. Here, we decide to focus on one of the most significant influence factors of stock price variation, news, to discover the relationship between news and stock price. In our project, we will analyze sentiment of news headlines, labeling them with positive, negative or neutral tags, which represent predictions for stock price changes: up, down, or staying the same. For future use, we can collect recent news into our dataset to make predictions as references for our investments.

To achieve our objectives, first, we combined two datasets and made a quick visualization on it. In order to do the following analysis, we cleaned and upsampled the dataset. For sentiment analysis, we created a Naive Bayes model as the baseline, used the sklearn library to construct two SVM models, and built a LSTM neural network with pytorch. We utilized accuracy and F1-score for model evaluation.

2. Method

2.1 Dataset

Our dataset includes three columns and over 15k rows. The three columns are the news headline, the stock ticker, and a sentiment label ranging between 0, 1, and 2 that indicates the stock trend after publishing the news headline. 0 means the stock went down by market close the day the article was published, 1 means the stock went up, and 2 means the price stayed the same.

A visualization of the stock labels count is shown in Figure 1.

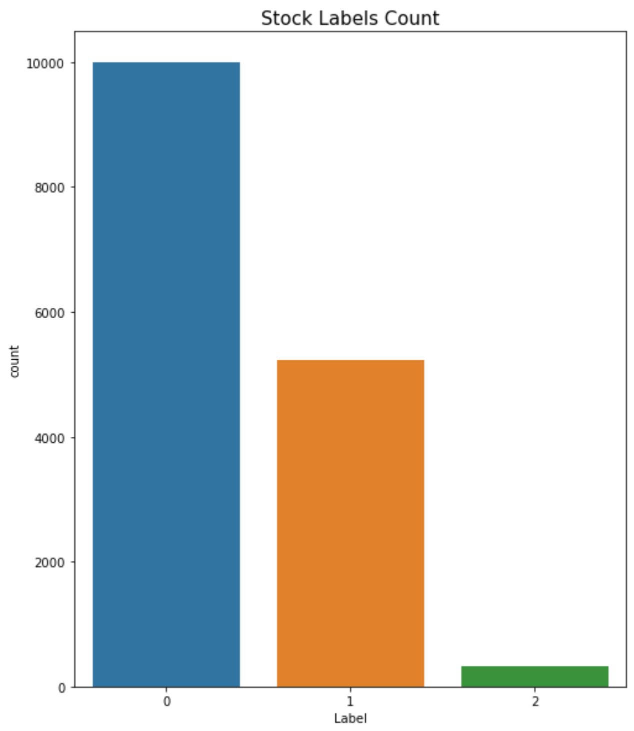


Figure 1: Stock Labels Count. In the original dataset, label 0 contains obviously more rows than others.

2.2 General Model Preparation

To prepare our data for the classification models, we cleaned our data by removing links and punctuations, changing all words to lowercase, and applying porter stemmer to the text.

Porter stemmer is a process for removing the common morphological and inflexional endings, such as words' ending in plural form, from words in English. For example, the words that end with sses will become ss. It helps reduce redundancy in similar words.

In our SVM and LSTM model, we removed stopwords, which is a set of commonly used words. In the Naïve Bayes model, we explore the effect of stopwords and see whether removing stopwords will lead to a significant change on our prediction model.

From the dataset visualization, it indicated that the number of data with label 1 and label 2 are far less than label 0. Since Naïve Bayes is not robust to class imbalance, we decided to do upsampling on the data with label 1 and 2 using the "resample" library from sklearn. For models that are not sensitive to the imbalanced dataset, SVM and LSTM, we decided to test and compare the difference between using an imbalanced and a balanced dataset.

2.3 Classification Models

We used three models, NB (Naïve Bayes), SVM (Support Vector Machine) and LSTM (Long Short-Term Memory) to predict the trend of different stocks with news headlines. We regarded the Naïve Bayes model, a less complex model compared to neural networks, as the baseline of our predictions.

2.3.1 Naïve Bayes

Naive Bayes is a classification technique based on Bayes' Theorem using probability to predict the chance of a given item belonging to a certain class. In this model, we used Naive Bayes to calculate our baseline probability.

To prepare the model, we used a bag-of-words model, which is to keep track of all the words that appeared in our training data, and generated a dictionary with the counts of occurrence for each word. With a bag-of-words for each class, our training model will calculate the probability of each word given its class and the probability of each class to predict the probability of a random word belonging to a certain class as the prior probability.

We also applied additive smoothing to our training data by adding count of 1 regardless of the word appearance in each class. This is because there is a chance that some word appeared in one class but not in others. Without additive smoothing, when we calculate the probability of some word that does not appear in one class, the probability for that particular class will be 0. For example, let [0.9, 0.9, 0.9, 0.0] be the probability for each word of a four word headline that this headline belongs to label 1, and [0.1, 0.1, 0.1, 0.1] to be the probability that each word belongs to label 2. Without additive smoothing, the probability of this headline being label 1 will be 0 and the probability of being label 2 will be 0.0001. This will cause misprediction even though the text should have a higher chance to be label 1. By applying additive smoothing, each word that appeared in our training data will have some probability in all of the classes. Therefore, words that never appear in a class will have a low probability instead of zero.

To predict the class of given text, we initiated the probability of each class with the prior probability. Then, we tokenized each text. If the word of the tokenized text appeared in our training data, we would have added the log probability of the text for each class to the prior probability of each class. In the end, we compared the probability of the given text and the prior probability and returned the class with the maxim probability.

2.3.2 SVM

Support Vector Machine, also known as SVM, is a classification technique by constructing hyperplanes in multidimensional space to separate different classes. SVM finds the hyperplane with maximum margin from nearest data points. If the input is hard to segregate, kernel functions can transform input data into a higher dimension and make it easier to separate the classes. In our case, each word will become a data point, and we will use a kernel function that can separate the points of our dataset into three classes.

We implemented SVM with the sklearn SVM package, and used two different kernel functions for training, the linear kernel and the radial basis function (RBF) kernel. This way we can compare the kernel functions and find the one that fits our dataset better.

2.3.3 LSTM

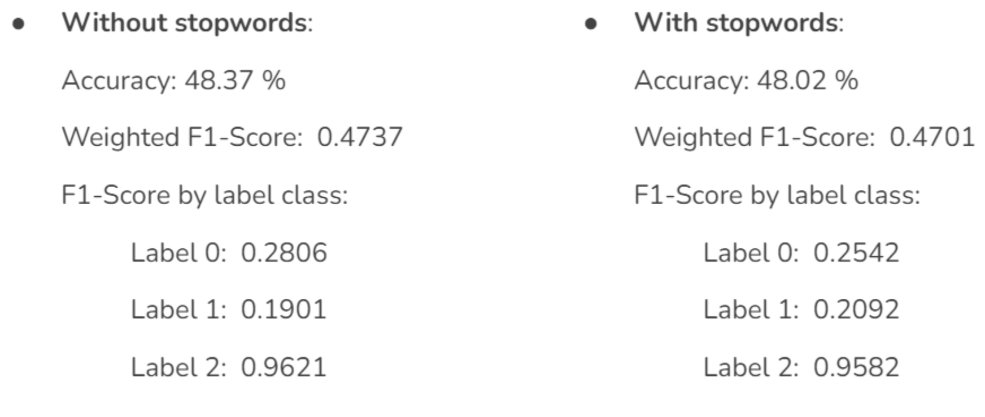
Long short-term memory (LSTM) is an artificial recurrent neural network architecture, which processes long-term dependencies and stores information at a given state. In our project, we used the torch library to build a Bi-Directional LSTM Neural Network.

Before training started, we prepared the dataset. We transferred the original dataset into torch form with torchtext.legacy.data. And then, we vectorized the text data with GloVe. The Bi-LSTM Neural Network was built in the regular way. We set the hyper-parameters and optimizer. The embedding dimension is set to 100, the hidden dimension set to 32. The input dimension is the length of HEADLINE, which is 64. The output dimension is 3, the length of LABEL. We used Adam as the optimizer, and Cross Entropy Loss as the loss criterion. In reality, there was no big difference when we used different optimizers in this project. To calculate accuracy, we wrote a function converting the 3-dimensional tensor into 1 dimension, so we could get the prediction of label 0, 1, or 2. After training and evaluating functions implemented, we started training this neural network.

3. Results

3.1 Naïve Bayes

We trained two Naive Bayes models, one with and one without stopwords to explore the effect of stopwords on prediction. The accuracy and F1-scores are shown in Figure 2.



- **Without stopwords**:

  Accuracy: 48.37 %

  Weighted F1-Score: 0.4737

  F1-Score by label class:

  Label 0: 0.2806

  Label 1: 0.1901

  Label 2: 0.9621

- **With stopwords**:

  Accuracy: 48.02 %

  Weighted F1-Score: 0.4701

  F1-Score by label class:

  Label 0: 0.2542

  Label 1: 0.2092

  Label 2: 0.9582

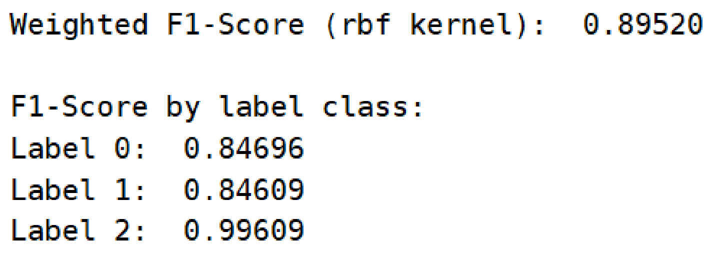
Figure 2: The accuracy and F1-scores of Naive Bayes model.

The two models did not have a visible difference in classification results. This is mainly because stopwords are just as likely to appear in all headlines regardless of their class. The information gain from stopwords is very low. Therefore, we concluded that stopwords did not affect the accuracy of Naive Bayes model. Besides, we got the baseline accuracy of our dataset prediction, which is around 48%.

3.2 SVM

We explored the effect of label rebalancing with SVM by training two SVM models with and without upsampling the data. Without rebalancing the data, the result of linear kernel is 65.56% and RBF kernel has 72.65%. After we upsampled the data, the accuracy of the linear kernel became 77.82% and the RBF kernel 89.54%.

The F1-score across classes is listed in Figure 3.



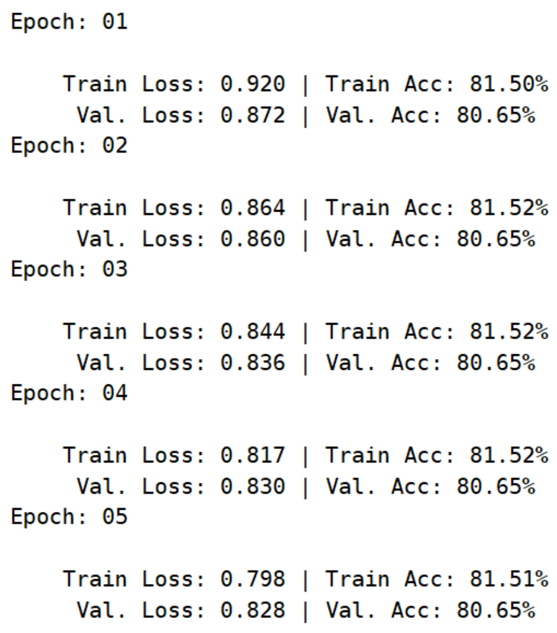
```
Weighted F1-Score (rbf kernel):  0.89520

F1-Score by label class:
Label 0:  0.84696
Label 1:  0.84609
Label 2:  0.99609
```

Figure 3: SVM (rbf kernel) F1-Score.

According to the result, first, based on the model predictions for upsampling and original dataset, we found that SVM models with data upsampling provided us better results. Second, we thought the SVM with RBF kernel model was overfitted for label 2 prediction because it had an extremely high F1 score. Probably, it was because there were very few samples with label 2 in the original dataset, the upsampling resulted in a highly repetitive dataset. However, the predictions about label 0 and 1 were perfect with this kernel. Based on some case tests, we did not find obvious overfitting cases, so we concluded that the SVM with RBF kernel model was the best.

3.3 LSTM

As mentioned above, we wrote an evaluating function and used accuracy as the indicator. The LSTM model gave the results in Figure 4.



```
Epoch: 01

    Train Loss: 0.920 | Train Acc: 81.50%
     Val. Loss: 0.872 | Val. Acc: 80.65%
Epoch: 02

    Train Loss: 0.864 | Train Acc: 81.52%
     Val. Loss: 0.860 | Val. Acc: 80.65%
Epoch: 03

    Train Loss: 0.844 | Train Acc: 81.52%
     Val. Loss: 0.836 | Val. Acc: 80.65%
Epoch: 04

    Train Loss: 0.817 | Train Acc: 81.52%
     Val. Loss: 0.830 | Val. Acc: 80.65%
Epoch: 05

    Train Loss: 0.798 | Train Acc: 81.51%
     Val. Loss: 0.828 | Val. Acc: 80.65%
```

Figure 4: LSTM training and evaluating accuracy with upsampled dataset.

The accuracy of our LSTM model was about 80%. For comparison, we trained this model with the original dataset, which means it had not been upsampled. The predicting accuracy was also around 80%. Therefore, we concluded that upsampling would not affect the prediction of the LSTM model.

4. Discussion

Among the three models, Naive Bayes had the lowest accuracy of 48%, which was also the baseline. We could see that the SVM and LSTM models improved from the baseline accuracy a lot. Yet surprisingly, the SVM with RBF kernel model had the highest accuracy. It may happen for various reasons. First, we did not provide the best hyper-parameters for our LSTM model. Second, our model was not complicated enough to train the data well. Third, we needed more data to train this model. Moreover, although we had several case tests, the SVM model still could overfit the data and hence gave higher predicting accuracy.

Besides, we found that the Naive Bayes model was very sensitive to the data imbalance, which was consistent with our expectation. However, the experiment indicated that SVM models were also sensitive to the imbalanced dataset. We thought it was because the number of labels in our dataset had a great difference. The number of rows with label 0 was almost twice as much as label 1, and more than ten times the number of labels 2. Upsampling yielded higher classification accuracy for both Naive Bayes and SVM, but the result of LSTM was unaffected.

5. Conclusion

Our project used three classifying models to predict stock trends, and the SVM with RBF kernel model gave us the highest accuracy for our dataset.

For future work, we can build more complicated Neural Network models for a better prediction. We could collect more data for training models. We may add other parameters that also influence stock price variation, such as trading volume and opening/closing prices. Besides, we can do cross validation to explore our models deeper.

References

[1]https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

[2]https://discourse.qingxzd.com/t/pytorch/58

Project Code Link

https://github.com/Lethe1999/CS6120FinalProject

Dataset Link

https://www.kaggle.com/datasets/sidarcidiacono/news-sentiment-analysis-for-stock-data-by-company