Yuhsiang Hong

Yitao Liu

Tweet Classification of NBA against Python

- Project Report -

Introduction

Our goal is to build machine learning models to classify tweets with hashtag Python against tweets with hashtag NBA. The reason why we chose this topic is because we love watching NBA and we are doing the final project using Python. The other reason is that we don't want many collected tweets containing both hashtags, which means that we want two hashtags to be irrelevant.

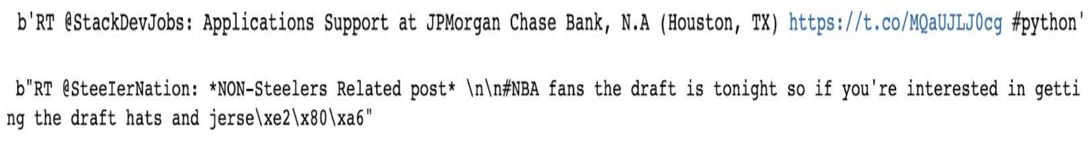
For our strategy, first, we collect as many as tweets as possible because we might remove some tweets after data cleaning. Second, we organize and clean tweet data. Third, we set TF-IDF and word embeddings as inputs for each model. Next, we run Naive Bayes, Logistic Regression and simple neural networks with one hidden layer and 50 units using scikit-learn as our baseline models. After that, we implement complex neural networks such as DNN and RNN using TensorFlow. Lastly, we compare results of using different models and draw our conclusion.

Data Collection

Tweets were collected by Tweepy Streaming and Cursor. The reason why we used two ways to collect tweets was because there were not enough Python tweets during that time. Thus, using Tweepy Cursor allowed us to collect tweets in the past.

1. Tweet Information

Tweets were created from November 13th to November 20th. We originally acquired 43810 Python tweets and 21499 NBA tweets. However, there were many tweets that were identical such as advertisement tweets. These kinds of tweets were usually created in a short period of time with the same context. Therefore we dropped these duplicate tweets and the total number of remaining Python tweets was 10133 and the total number of remaining NBA tweets was 15251. Figure 1 presents two examples of tweets with Python and NBA hashtags.



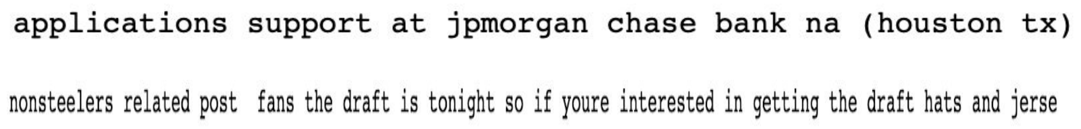
```
b'RT @StackDevJobs: Applications Support at JPMorgan Chase Bank, N.A (Houston, TX) https://t.co/MQaUJLJ0cg #python'

b"RT @SteeIerNation: *NON-Steelers Related post* \n\n#NBA fans the draft is tonight so if you're interested in getti
ng the draft hats and jerse\xe2\x80\xa6"
```

Figure 1: Tweets with hashtags of Python and NBA, in order. The above tweet is a Python tweet and the below tweet is a NBA tweet.

2. Tweet Cleaning

In Figure 1, we can see that tweets are very messy, so it is necessary to organize and clean them. The method we used to do this was using Regex in python, which we had already learnt in the lectures and was very useful to filter out strings with certain patterns.

First, we removed meaningless words and symbols such as bytes prefixes, url, UTF-8 characters, retweet indicators and account names. Second, we substituted newline indicators with whitespaces. The reason why we did that was we did not want words to squeeze together after we removed newline indicators so that they would not be considered as a single word. Third, we made all the tweets lowercase and dropped "python", "nba" and non-alphanumeric characters. The reason why we not just removed hashtags but the target words from the tweets' context was because we desired to make it harder for models to classify tweets, especially when we had already selected two very distinct words for hashtags.



```
applications support at jpmorgan chase bank na (houston tx)

nonsteelers related post  fans the draft is tonight so if youre interested in getting the draft hats and jerse
```

Figure 2: Python and NBA tweets after filtering out strings. The above tweet is a Python tweet and the below tweet is a NBA tweet. These tweets are the same as in Figure 1.

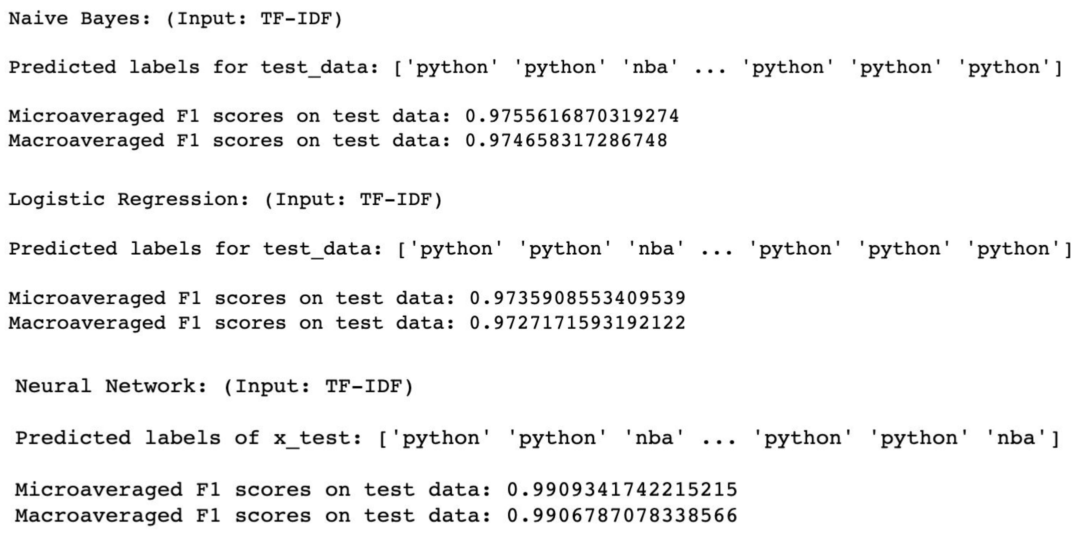
As we can see above, the tweets in Figure 2 are cleaner and more organized than in Figure 1. After the filtering process, the total number of remaining tweets were 25362 and we split the dataset into 90% for training and 10% for testing.

Baseline Models

Before implementing neural networks in TensorFlow, we trained three classification models using scikit-learn package to have an overview of the classification results. Three baseline models were Naive Bayes, Logistic Regression, and single hidden layer Neural

Networks with 50 units. We used TF-IDF and word embeddings (50d) as our inputs for each model. However, for Naive Bayes, we didn’ t use word embeddings as the input because they had negative values and Naive Bayes assumed variables had multinomial distributions, so they could not contain negative values. Therefore, we only used TF-IDF for Naive Bayes.

1. TF-IDF

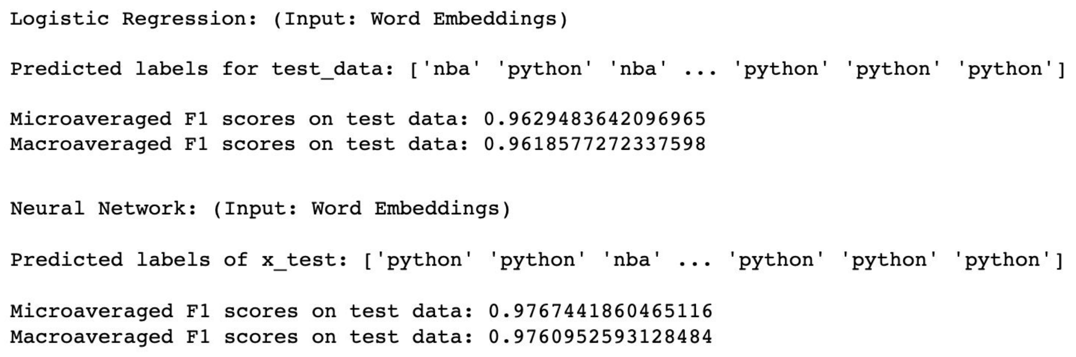


```
Naive Bayes: (Input: TF-IDF)

Predicted labels for test_data: ['python' 'python' 'nba' ... 'python' 'python' 'python']

Microaveraged F1 scores on test data: 0.9755616870319274
Macroaveraged F1 scores on test data: 0.974658317286748

Logistic Regression: (Input: TF-IDF)

Predicted labels for test_data: ['python' 'python' 'nba' ... 'python' 'python' 'python']

Microaveraged F1 scores on test data: 0.9735908553409539
Macroaveraged F1 scores on test data: 0.9727171593192122

Neural Network: (Input: TF-IDF)

Predicted labels of x_test: ['python' 'python' 'nba' ... 'python' 'python' 'nba']

Microaveraged F1 scores on test data: 0.9909341742215215
Macroaveraged F1 scores on test data: 0.9906787078338566
```

Figure 3: Classification Reports of baseline models. Models, from top to bottom, are Naive Bayes, Logistic Regression and Neural Networks.

As we can see in Figure 3, the F1 scores are already high on test data for each model. It is predictable since two hashtags we use are highly irrelevant. Neural Networks outperform other two models in this case, but we will check that if it has the same performance in cross validation.

2. Word Embeddings



```
Logistic Regression: (Input: Word Embeddings)

Predicted labels for test_data: ['nba' 'python' 'nba' ... 'python' 'python' 'python']

Microaveraged F1 scores on test data: 0.9629483642096965
Macroaveraged F1 scores on test data: 0.9618577272337598

Neural Network: (Input: Word Embeddings)

Predicted labels of x_test: ['python' 'python' 'nba' ... 'python' 'python' 'python']

Microaveraged F1 scores on test data: 0.9767441860465116
Macroaveraged F1 scores on test data: 0.9760952593128484
```

Figure 4: Classification Reports of baseline models. Models, from top to bottom, are

Logistic Regression and Neural Networks.

In Figure 4, we can discover that the performances of using word embeddings are slightly worse than TF-IDF. This phenomenon might be due to the fact that the dimension of word embeddings we use is not large enough. However, the F1 scores are still high and neural networks outperform other models once again.

3. Cross Validation (5-Fold)

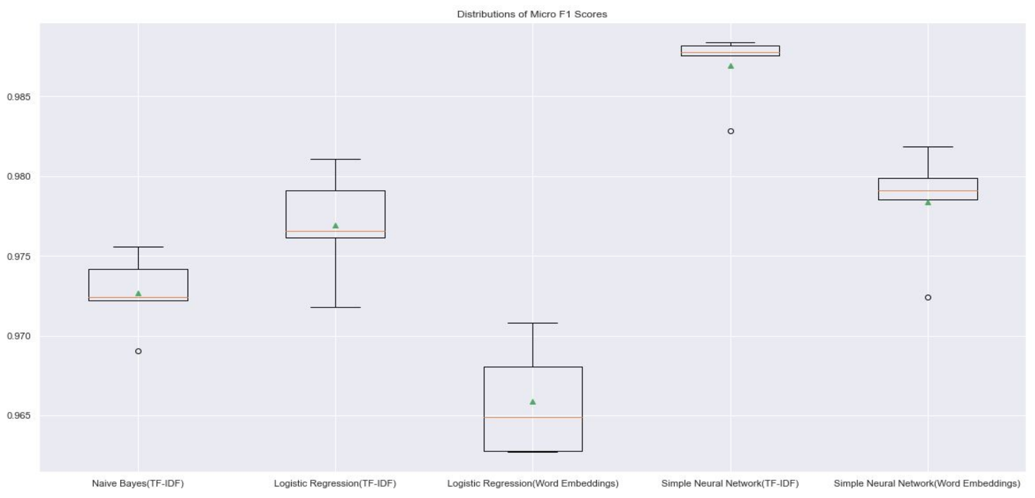


Figure 5: Distributions of micro F1 scores from different models using different inputs. From left to right, there are Naive Bayes using TF-IDF, Logistic Regression using TF-IDF and word embeddings, simple Neural Network using TF-IDF and word embeddings. The green triangles in each model's distribution are means of their F1 scores.

Cross validation reduces the bias that might be generated from data splitting. Therefore, we can make two conclusions from Figure 5. First, no matter what the input is, neural networks on average have better results than other models have because of the higher F1 scores. Second, the results of TF-IDF are better than word embeddings and one of the reasons is already mentioned above. One thing we want to mention is that we only include micro F1 scores here because macro and micro F1 scores are very similar, so we use micro F1 scores as representatives to show results.

TensorFlow Neural Networks

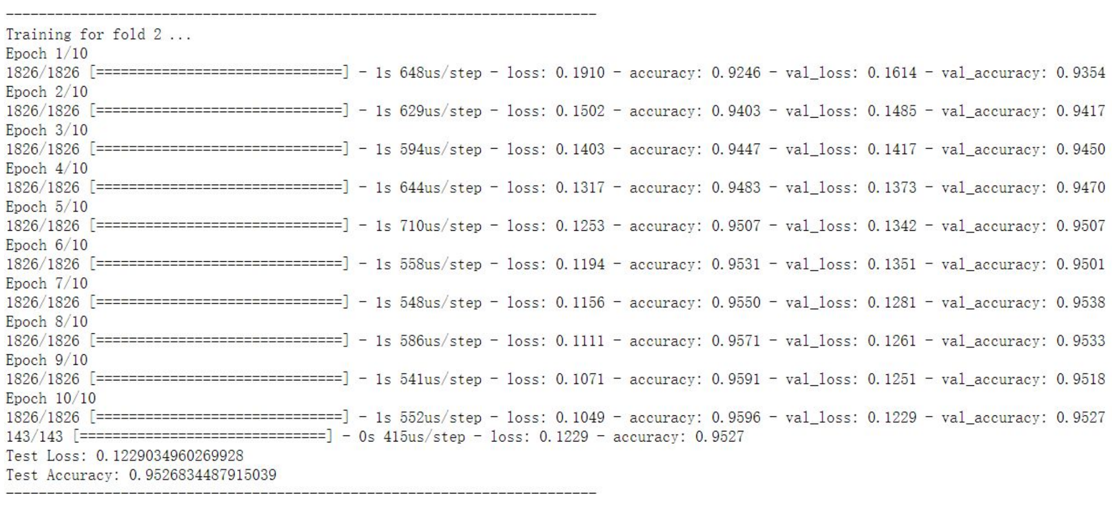
Next, we started to build neural networks using TensorFlow. TensorFlow is a free and open-source software library for machine learning, it is a very convenient and helpful tool for us to build neural networks models. Using TensorFlow, we built 4 models in total. Two of them are feedforward neural networks models, the other two are recurrent neural networks models (RNN).

For the feedforward neural network models, like before, we used the GloVe word embeddings (50d) as our input. For the RNN models, we used the Text Vectorization and Embedding layers provided by TensorFlow to process our tweets data as our input. For all of the four models, we applied the KFold method from scikit-learn package to perform 5-fold cross validation on our training data. We fitted the models with training data, evaluated the models with validation data. When we were fitting our models, we used 10 epochs and a batch size of 10. For the loss function, we chose binary cross entropy to update our model. We recorded the validation accuracy and loss for every fold, then calculated the average validation accuracy and loss for the models. We improved our models using techniques such as Dropout and Regularization. Last, we applied our models on our testing data to see how well they can classify our tweets data.

1. Feedforward Neural Network 1 Hidden Layer with 50 Units

Our first TensorFlow model is a simple feedforward neural networks model with one hidden layer. For the activation function of the hidden layer, we chose The rectified linear activation function (ReLU) since it is widely used. For the activation function of the output layer, we chose the sigmoid function since this is a binary classification problem.

Then we had to choose a proper number of hidden units in order to let our model perform well. So, we built and tested 3 models with 10, 50, and 100 hidden units separately. Figure 6 shows an intermediate step during a fitting process in a particular fold. During each epoch, the training/validation loss are decreasing, and the training/validation accuracy are increasing, this is expected.



```
------------------------------------------------------------------------
Training for fold 2 ...
Epoch 1/10
1826/1826 [==============================] - 1s 648us/step - loss: 0.1910 - accuracy: 0.9246 - val_loss: 0.1614 - val_accuracy: 0.9354
Epoch 2/10
1826/1826 [==============================] - 1s 629us/step - loss: 0.1502 - accuracy: 0.9403 - val_loss: 0.1485 - val_accuracy: 0.9417
Epoch 3/10
1826/1826 [==============================] - 1s 594us/step - loss: 0.1403 - accuracy: 0.9447 - val_loss: 0.1417 - val_accuracy: 0.9450
Epoch 4/10
1826/1826 [==============================] - 1s 644us/step - loss: 0.1317 - accuracy: 0.9483 - val_loss: 0.1373 - val_accuracy: 0.9470
Epoch 5/10
1826/1826 [==============================] - 1s 710us/step - loss: 0.1253 - accuracy: 0.9507 - val_loss: 0.1342 - val_accuracy: 0.9507
Epoch 6/10
1826/1826 [==============================] - 1s 558us/step - loss: 0.1194 - accuracy: 0.9531 - val_loss: 0.1351 - val_accuracy: 0.9501
Epoch 7/10
1826/1826 [==============================] - 1s 548us/step - loss: 0.1156 - accuracy: 0.9550 - val_loss: 0.1281 - val_accuracy: 0.9538
Epoch 8/10
1826/1826 [==============================] - 1s 586us/step - loss: 0.1111 - accuracy: 0.9571 - val_loss: 0.1261 - val_accuracy: 0.9533
Epoch 9/10
1826/1826 [==============================] - 1s 541us/step - loss: 0.1071 - accuracy: 0.9591 - val_loss: 0.1251 - val_accuracy: 0.9518
Epoch 10/10
1826/1826 [==============================] - 1s 552us/step - loss: 0.1049 - accuracy: 0.9596 - val_loss: 0.1229 - val_accuracy: 0.9527
143/143 [==============================] - 0s 415us/step - loss: 0.1229 - accuracy: 0.9527
Test Loss: 0.1229034960269928
Test Accuracy: 0.9526834487915039
------------------------------------------------------------------------
```

Figure 6: An intermediate step during a fitting process in one fold.

We also plotted the trending of loss and accuracy for each fold. Figure 7 shows the trending of training/validation loss and training/validation accuracy with respect to epochs in a particular fold.

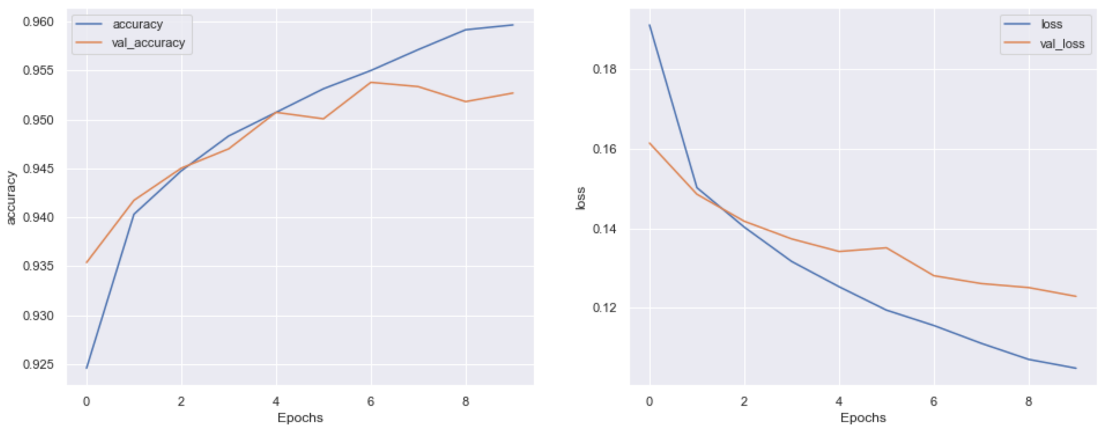
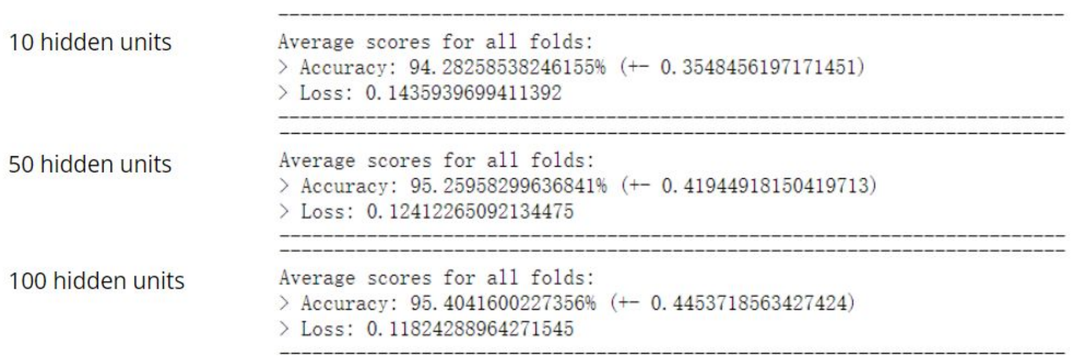


Figure 7: Accuracy and loss trending plot in a particular epoch.

Then, we compared the average validation accuracy and loss of models with 10, 50, and 100 hidden units. Figure 8 shows the comparison. After comparing the results, we found that changing the number of hidden units from 10 to 50 slightly increases the accuracy, but there is no significant increase from 50 to 100. Also, it has a side effect of overfitting. So, we finally chose 50 hidden units, 1 hidden layer as our architecture.



10 hidden units

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 94.28258538246155% (+- 0.3548456197171451)
> Loss: 0.1435939699411392
------------------------------------------------------------------------
```

50 hidden units

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 95.25958299636841% (+- 0.41944918150419713)
> Loss: 0.12412265092134475
------------------------------------------------------------------------
```

100 hidden units

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 95.4041600227356% (+- 0.4453718563427424)
> Loss: 0.11824288964271545
------------------------------------------------------------------------
```

Figure 8: Comparison between 3 architectures.

After deciding our architecture, we needed to solve the overfitting issue. We used the Dropout method with a 50% chance to improve our model. Figure 9 and 10 shows the trend of accuracy and loss in a particular epoch of the model before/after dropout.

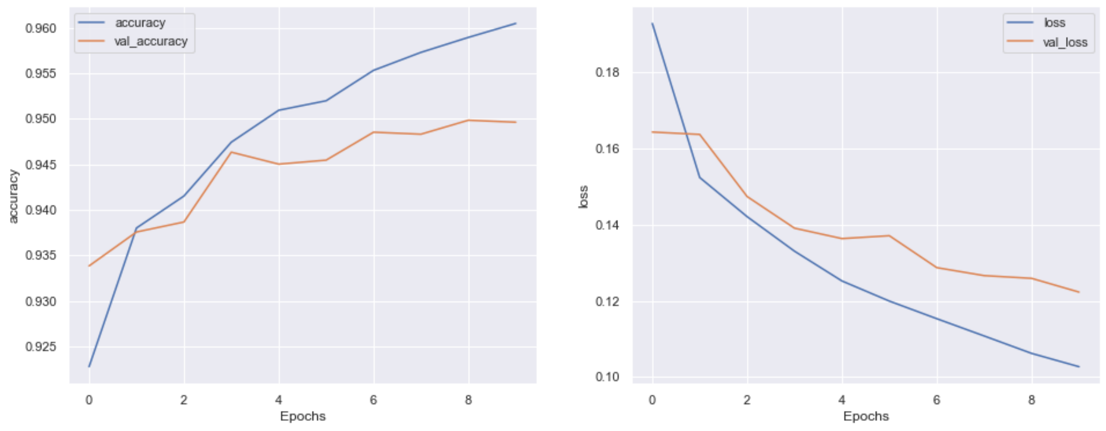


Figure 9: First model before dropout.

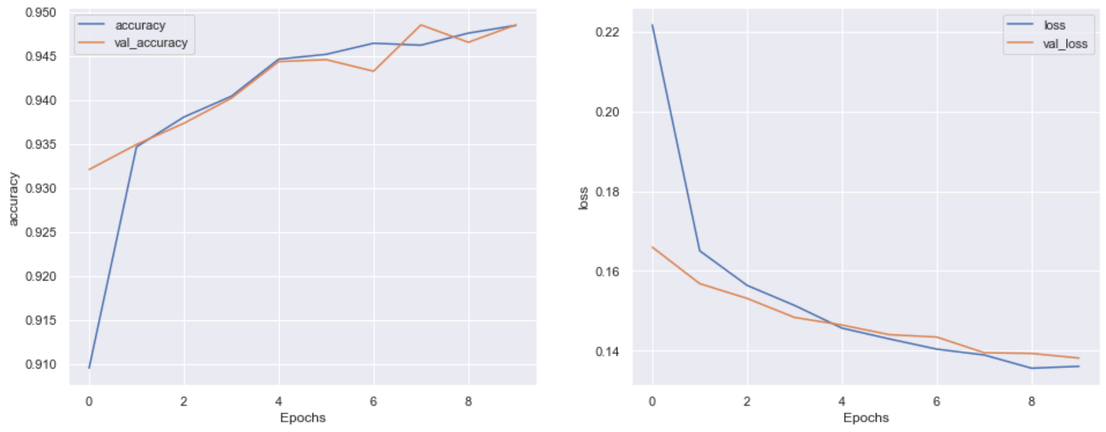


Figure 10: First model after dropout.

Next, we can use the improved model to do the classification on our testing data. We used a heatmap of confusion matrix and F1 scores to evaluate the classification result. Figure 11 is the heatmap, Figure 12 is the F1 scores.

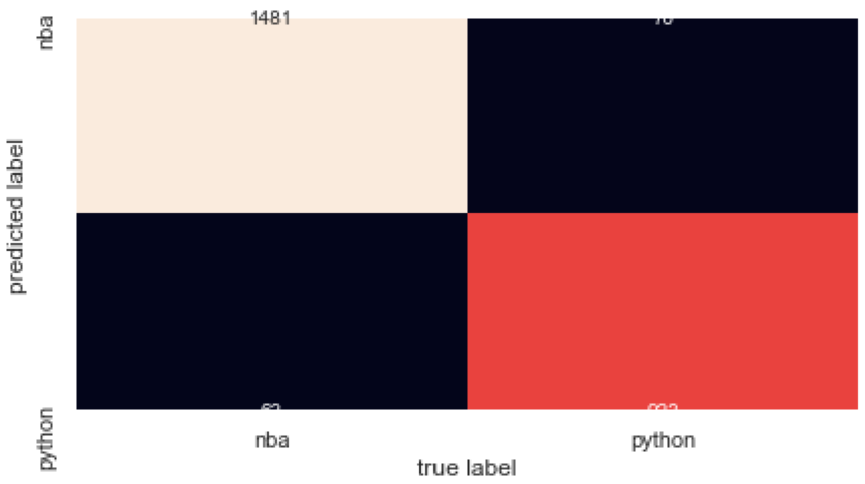
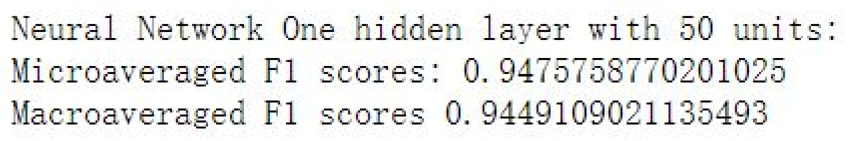


Figure 11: Heatmap of the classification result of the first model.



```
Neural Network One hidden layer with 50 units:
Microaveraged F1 scores: 0.9475758770201025
Macroaveraged F1 scores 0.9449109021135493
```

Figure 12: F1 scores of the classification result of the first model.

2. Feedforward Neural Network 3 Hidden Layers with 50 Units

Our second TensorFlow model is a neural networks model with multiple hidden layers. Like before, we chose ReLU as the activation function of the hidden layers, and we chose the sigmoid function as the function of the output layer. After doing some comparison, we found that increasing the number of layers does not significantly change the validation accuracy, it only improves the training accuracy. Therefore, we selected 3 hidden layers and 50 units as our second model's architecture. Like before, we used dropout to solve the overfitting problem. Figure 13 and 14 shows the trend of accuracy and loss in a particular epoch of the model before/after dropout.

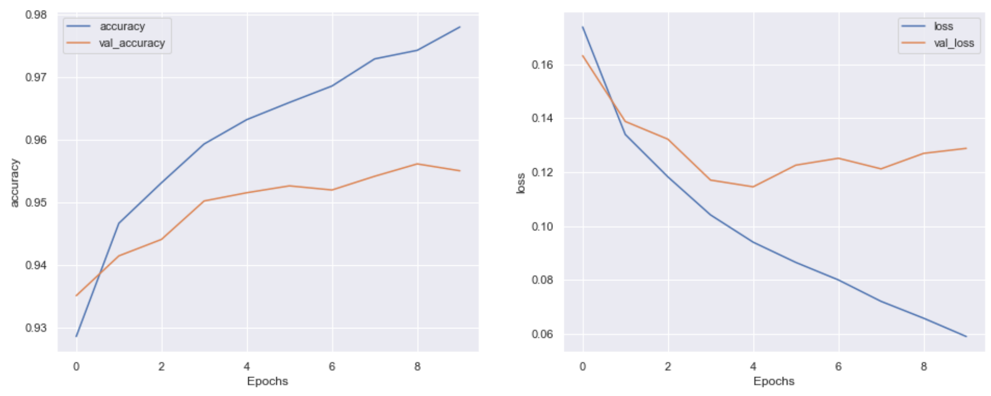
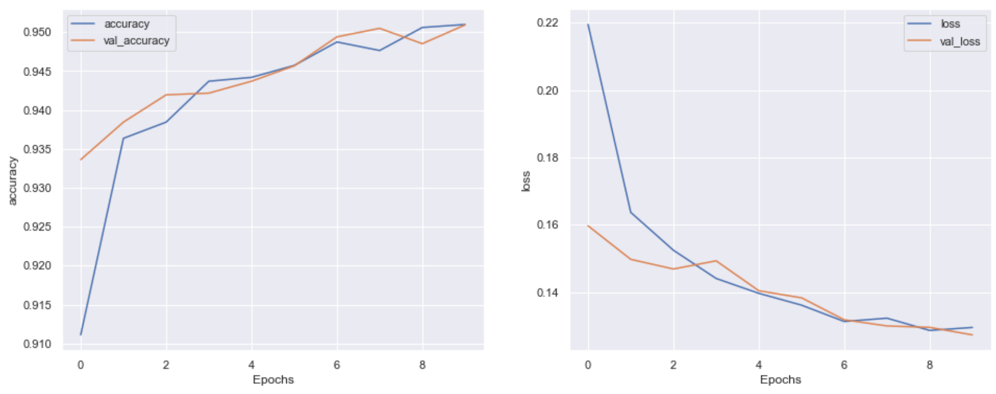


Figure 13: Second model before dropout.



`**Figure 14**: Second model after dropout.

Figure 15 and 16 shows the classification result of testing data.

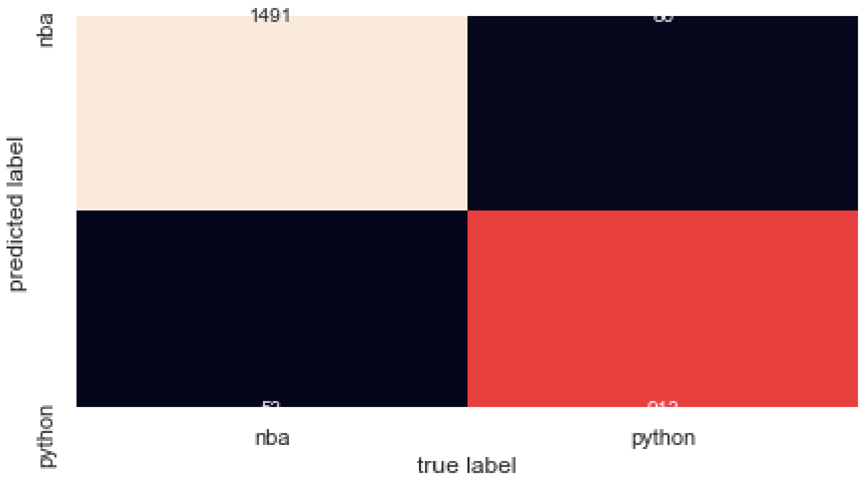
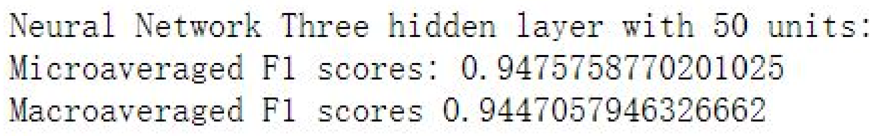


Figure 15: Heatmap of the classification result of the second model.



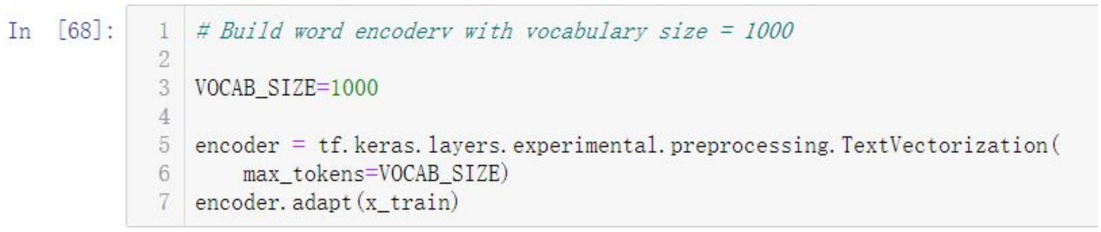
```
Neural Network Three hidden layer with 50 units:
Microaveraged F1 scores: 0.9475758770201025
Macroaveraged F1 scores 0.9447057946326662
```

Figure 16: F1 scores of the classification result of the second model.

3. Recurrent Neural Networks: Long Short-Term Memory

Our third TensorFlow model is a recurrent neural networks Long Short-Term Memory (LSTM RNN) model. The reason we used a RNN model was because it is a text classification problem. The input is a tweet sentence, in other words, the input is a sequence of words. So, we wanted to keep updating the relationship between the labels and the input while we go through the sentence. The reason we used a LSTM model was because the sentence could be long, and we wanted to preserve all the information from the beginning to the end of our sentence.

Now, instead of using the GloVe embedding, we used TensorFlow's text vectorization layer to encoding the sentences. We chose 1000 as the vocabulary size. Figure 17 shows the text vectorization code. We used the encoded words as the input of our third and fourth model.



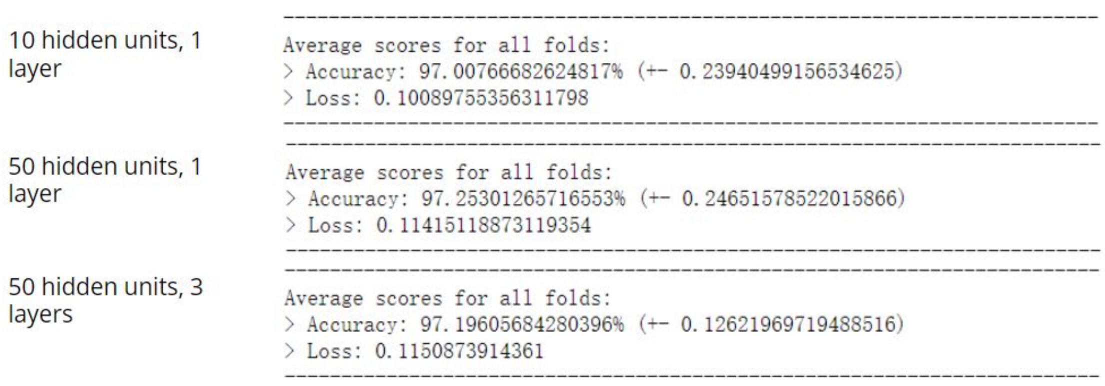
```
In [68]:
# Build word encoderv with vocabulary size = 1000

VOCAB_SIZE=1000

encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(x_train)
```

Figure 17: Text vectorization

Like first and second models, the activation function of the output layer is sigmoid function. We tried several different kinds of architecture, and compared them. Figure 18 shows the comparison. We found out 2 points:

- Increasing hidden units does not improve validation accuracy (It only increases the train accuracy).
- Adding layers does not improve accuracy.

In order to keep the model simple, we finally chose 50 hidden units, 1 layer.



10 hidden units, 1 layer

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 97.00766682624817% (+- 0.23940499156534625)
> Loss: 0.10089755356311798
------------------------------------------------------------------------
```

50 hidden units, 1 layer

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 97.25301265716553% (+- 0.24651578522015866)
> Loss: 0.11415118873119354
------------------------------------------------------------------------
```

50 hidden units, 3 layers

```
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 97.19605684280396% (+- 0.12621969719488516)
> Loss: 0.1150873914361
------------------------------------------------------------------------
```

Figure 18: Comparison between 3 architectures.

Like before, we used dropout to fix the overfitting issue. This time, we also used Regularization and reduced the number of hidden units. The improved model was a lot better than before.

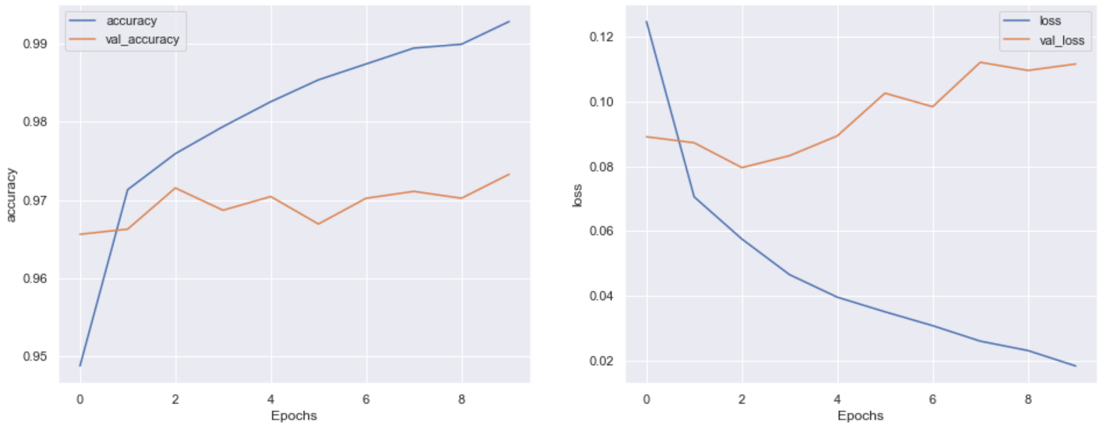


Figure 19: Third model before dropout, regularization, and reducing hidden units.

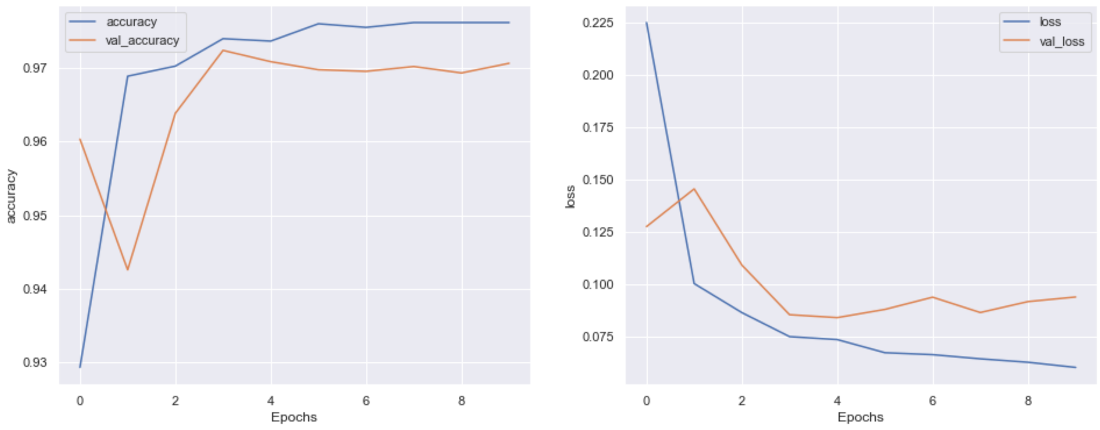


Figure 20: Third model after dropout, regularization, and reducing hidden units.

Figure 21 and 22 shows the classification result of testing data. We can see that compared to simple neural networks models, the F1 scores significantly increased.

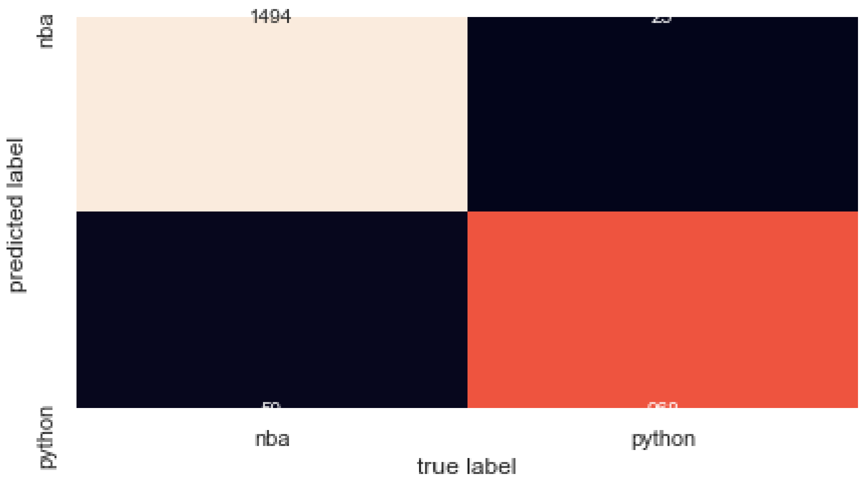
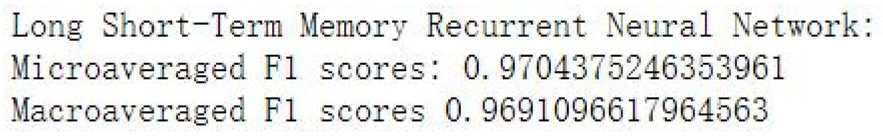


Figure 21: Heatmap of the classification result of the third model.



```
Long Short-Term Memory Recurrent Neural Network:
Microaveraged F1 scores: 0.9704375246353961
Macroaveraged F1 scores 0.9691096617964563
```

Figure 22: F1 scores of the classification result of the third model.

4. Bidirectional LSTM RNN

Our fourth TensorFlow model is a bidirectional LSTM RNN model. The reason we used a bidirectional model was because we had the entire input sequence, and the bidirectional model is powerful under this situation. The architecture of the model is 50 hidden units, 1 layer. Like the other 3 models, the activation function of the output layer is sigmoid function. Like the third model, we used dropout, regularization, and reducing hidden units to solve the overfitting issue. The improved model was a lot better than before.

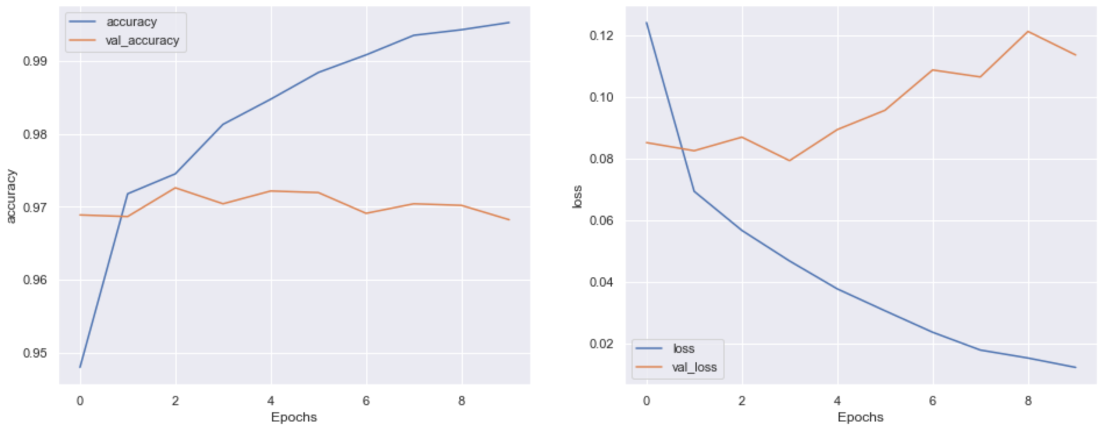


Figure 23: Fourth model before dropout, regularization, and reducing hidden units.

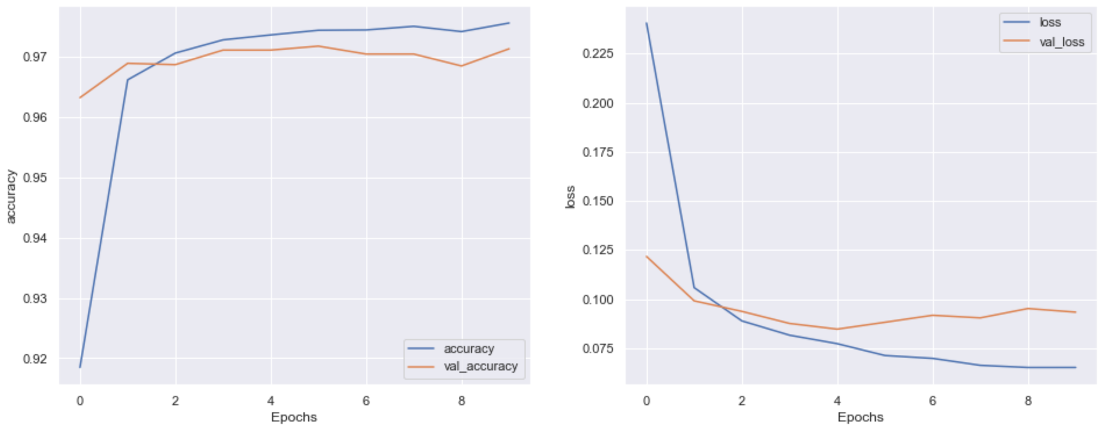


Figure 24: Fourth model after dropout, regularization, and reducing hidden units.

Figure 25 and 26 shows the classification result of testing data. We can see that compared to the LSTM model, the F1 scores of the bidirectional LSTM model slightly increased.

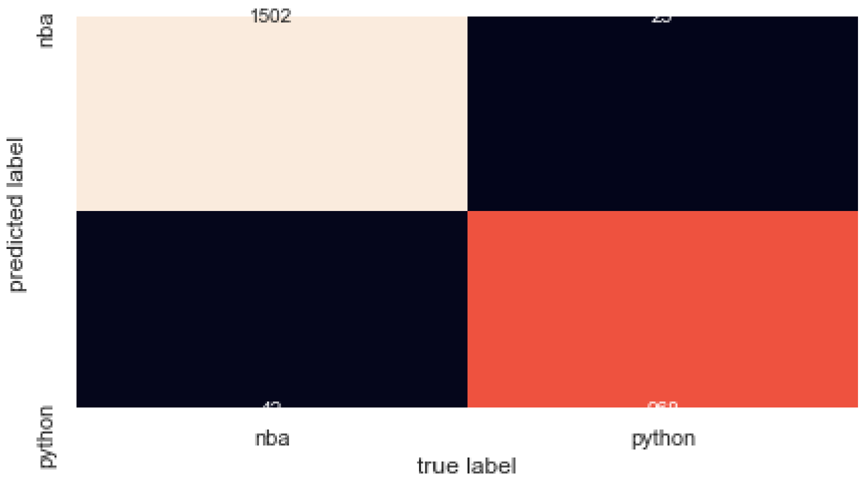
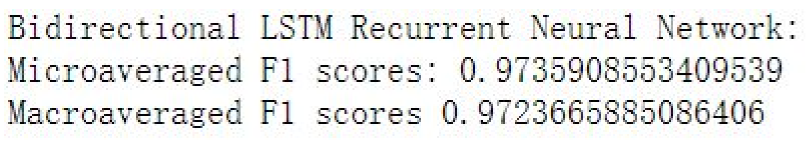


Figure 25: Heatmap of the classification result of the fourth model.



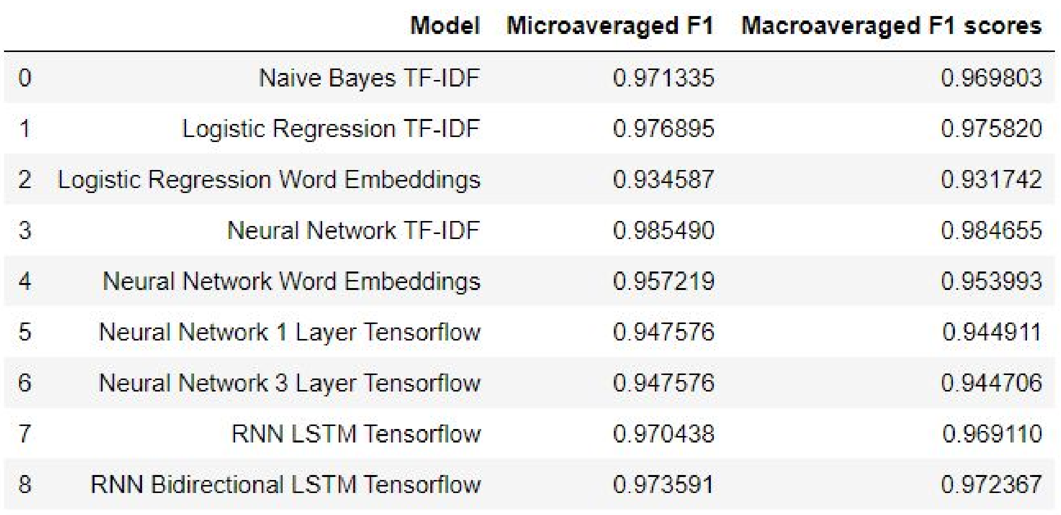
```
Bidirectional LSTM Recurrent Neural Network:
Microaveraged F1 scores: 0.9735908553409539
Macroaveraged F1 scores 0.9723665885086406
```

Figure 26: F1 scores of the classification result of the fourth model.

Conclusion

Figure 27 is a table of the F1 scores of all the models we had tried, including baseline models and TensorFlow models. From the table, we can make several conclusions:

- For our tweet data, the TF-IDF input tends to perform better than GloVe 50d embedding input.
- For our tweet data, Neural Networks models tend to perform better than Naive Bayes and Logistic Regression models.
- For our tweet data, Recurrent Neural Networks models tend to perform better than simple Feedforward Neural Networks models.
- For our tweet data, using Bidirectional LSTM RNN does help increase our classification accuracy.



	Model	Microaveraged F1	Macroaveraged F1 scores
0	Naive Bayes TF-IDF	0.971335	0.969803
1	Logistic Regression TF-IDF	0.976895	0.975820
2	Logistic Regression Word Embeddings	0.934587	0.931742
3	Neural Network TF-IDF	0.985490	0.984655
4	Neural Network Word Embeddings	0.957219	0.953993
5	Neural Network 1 Layer Tensorflow	0.947576	0.944911
6	Neural Network 3 Layer Tensorflow	0.947576	0.944706
7	RNN LSTM Tensorflow	0.970438	0.969110
8	RNN Bidirectional LSTM Tensorflow	0.973591	0.972367

Figure 27: Table of F1 scores of all the models we tried.

Overall, the best model for our classification is the Neural Network model implemented by scikit-learn package, using TF-IDF input.

Summary of the Research Paper

The research paper we investigated is *Tweets Classification on the Base of Sentiments for US Airline Companies*. The paper constructs voting classifiers to help sentiment analysis for such organizations. The dataset name is "twitter-airline-sentiment" and it has a total

14,640 records, including tweets for six airlines of the US. Each record is labeled as positive, negative, or neutral according to the sentiment polarity. There are multiple machine learning algorithms are used to evaluate the performance, including SVM, AdaBoost, Decision Tree, Naive Bayes, Extra Trees Classifier, Random Forest, Logistic Regression, Stochastic Gradient Descent Classifier, and LSTM.

There are some similarities between this paper and our project. First, we removed stop words from all the tweets in data preprocessing and the paper also did the same thing because it is unnecessary to analyze stop words in tweets. Second, the results we got are similar to this paper. We also found out that TF-IDF as inputs for classification models performs better than word embeddings as inputs for classification models. This phenomenon encourages us that our project is heading in a good direction and our conclusion can hold in different scenarios.

Contribution

Yuhsiang Hong: Introduction, Data Collection, Baseline Models, Summary of the Research Paper
Yitao Liu: TensorFlow Neural Networks, Conclusion

Appendix

Research Paper: Tweets Classification on the Base of Sentiments for US Airline Companies

Link:
https://www.researchgate.net/publication/337050333_Tweets_Classification_on_the_Base_of_Sentiments_for_US_Airline_Companies