DSCI-552 Section 32415D: Spring 2021

Problem Set 5: Multi-Class Tweets Classification

Jianing (Julia) Chen

MS Applied Data Science, University of Southern California,

Los Angeles, California 90089, USA

(Dated: March 31, 2021)

Abstract

This report works on Twitter messages and utilizes both Naive Bayes, Recurrent Neural Networks (RNN), and RNN Long Short-Term Memory (LSTM) architecture to sort the messages into five categories. This classification will help us know how people's perception changes and will help federal agencies plan for the future. The result shows that RNN and RNN with LSTM architecture can give good precision results. But the best result is RNN with LSTM architecture because it can deal with longer sentences. My final result gives over 0.72 on both micro and macro F1 score, and the achieves 0.66110 accuracy on Kaggle.

I. INTRODUCTION

I aim to build text classifiers in this problem set. In the beginning, I preprocessing the datasets, such as splitting the datasets and applying tokenization to the tweets so that the computer can understand it. The baseline model I use naive bayes model from sklearn. To improve the accuracy of classification, I also build Recurrent Neural Networks and Long Short-Term Memory by using Tensorflow. To measure models' performance, I calculate classification report which can display the precision, recall, F1 scores for the model.

II. DATA EXPLORATION & DATA PREPROCESSING

For this problem set, we are given training text data with labels and testing data. The training data have 37041 instances of the Twitter message. The labels range from 0 to 4, and represent the feeling of users. Where 0 means extremely negative, 1 means negative, 2 means neutral, 3 means positive and 4 means extremely positive. The representation of each label and its count shows on TABLE I. We can see that most people have positive attitudes toward epidemiological situations, and the overall positive attitude people (including extremely positive) are more than negative altitude (including extremely negative).

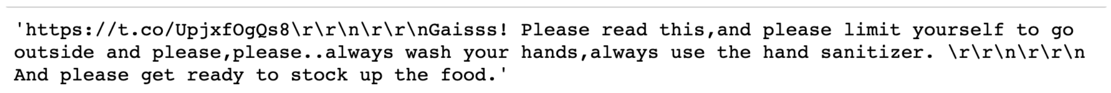
TABLE I. Count for Each Label

Label	Feeling	Count
0	Extremely Negative	4946
1	Negative	8930
2	Neutral	6930
3	Positive	10282
4	Extremely Positive	5953

The training data has two parts: the text (in CSV file), and the other is the labels (in a csv file). Few steps I did for the data preprocessing:

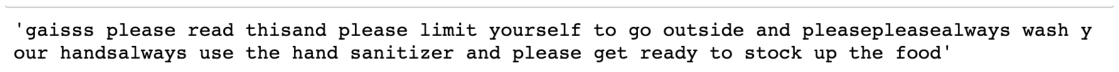
- split the data into two parts: 80% training set, 20% validation set. I will train my model based on the training set and use the validation set to tune my model. The final result will test on the testing set and submit to the Kaggle.
- apply Tensorflow Tokenizer to training text, validation text, and testing text. The process of tokenization is to represent words in a way that computers can understand

and be able to train them. After done Tokenizer, I apply one-hot encoding to all three text datasets.



```
'https://t.co/UpjxfOgQs8\r\r\n\r\r\nGaisss! Please read this,and please limit yourself to go
outside and please,please..always wash your hands,always use the hand sanitizer. \r\r\n\r\r\n
And please get ready to stock up the food.'
```

FIG. 1. Example of Twitter message before dealing with punctuation



```
'gaisss please read thisand please limit yourself to go outside and pleasepleasealways wash y
our handsalways use the hand sanitizer and please get ready to stock up the food'
```

FIG. 2. Example of Twitter message after dealing with punctuation

- set the maximum number of words to keep, based on word frequency as 5000, and filter out the special characters, such as exclamation marks, question marks, hashtags, and so on. Besides, I define the Out Of Vocab token (OVV), which will replace any unknown words with a token of our choosing.
- since it is a multi-class classification, so also apply one hoy encoding to labels datasets: train_y and val_y.

III. MODEL SELECTION

A. Sklearn Naive Bayes Classifier

The Naive Bayes classifier's idea is created based on Bayesian Theorem[1]. The fundamental assumption hold by this classifier is to treat each word independently. That is to say, the Naive Bayes classifier will ignore all rules, such as the occurrence of one word does not affect the probability of the other word's happening but keeping track of the likelihood of the labels by given words or phrases. For example, the Naive Bayes classifier will treat the word "summer" has no relationship with "hot". Therefore, statistically speaking, the Naive Bayes classifier leads to high bias by given limited training samples. Despite the assumption made by the Naive Bayes limit us to get correct classifiers in real-world situations, it is still good to use it as our baseline model to have an idea of the general accuracy of models and improve from here.

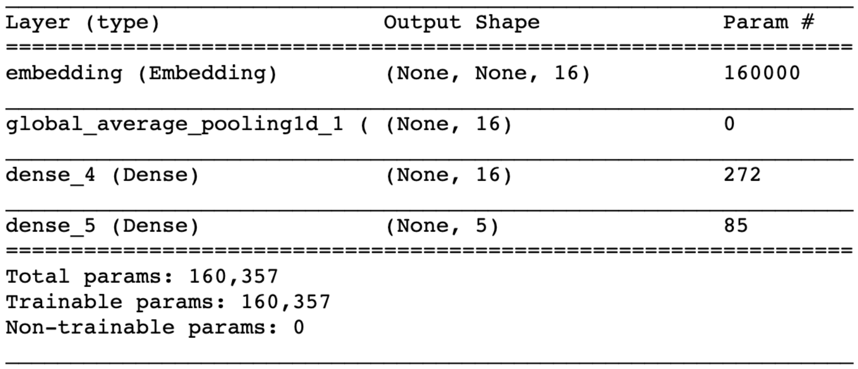
[1] Bayesian Theorem: describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

Because we have 5 different labels to classify, the specific model I used is multinomial naive bayes. Compared to regular Naive Bayes, THE multinomial Naive Bayes is a specific algorithm which uses a multinomial distribution for each of the features.

B. Tensorflow RNN Model

The Recurrent Neural Networks (RNNs) work on the principle of saving the output of a particular layer then feeding this back to the input in order to predict the output of the layer, that is why RNNs are capable of handling sequential data[1].

The first layer of the Recurrent Neural Network for natural language processing be the embedding layer. How we embed words will determine the direction of each word learned by each epoch during the training process. The second layer will be the global average pooling layer, then we will have one dense layer with ReLu activation, then another dense layer with Softmax activation.



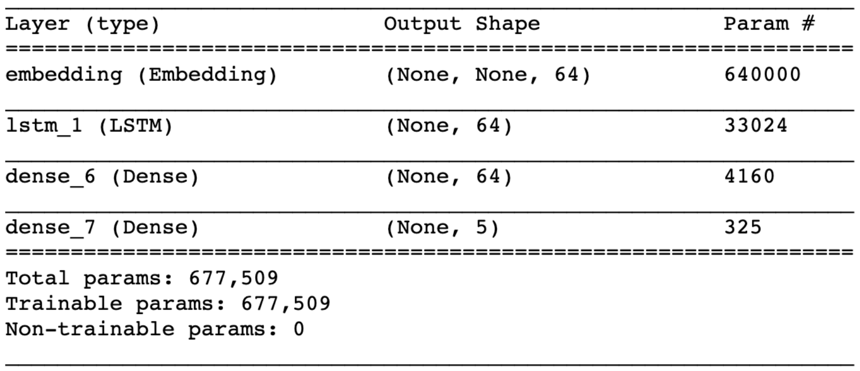
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 16)          160000
_________________________________________________________________
global_average_pooling1d_1 ( (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 85
=================================================================
Total params: 160,357
Trainable params: 160,357
Non-trainable params: 0
_________________________________________________________________
```

FIG. 3. RNN Structure

C. Tensorflow RNN LSTM Architecture

What makes Long Short-Term Memory architecture different from the standard RNNs is: the standard RNNs model has a repeating module in a straightforward structure, so over a long distance, the word context can be deluded. While LSTMs has repeating module contains more layers, such as sigmoid layers and a tanh layers, it can pass cross longer sentences to see the word's impact in the early sentence and determine the semantics the whole sentence[2].

What I put inside the model pretty much is the same as previous model, except changing the pooling layer as LSTM layer. The detailed model structure shows below:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          640000
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 325
=================================================================
Total params: 677,509
Trainable params: 677,509
Non-trainable params: 0
_________________________________________________________________
```

FIG. 4. LSTM Architecture Structure

IV. MODEL EVALUATION

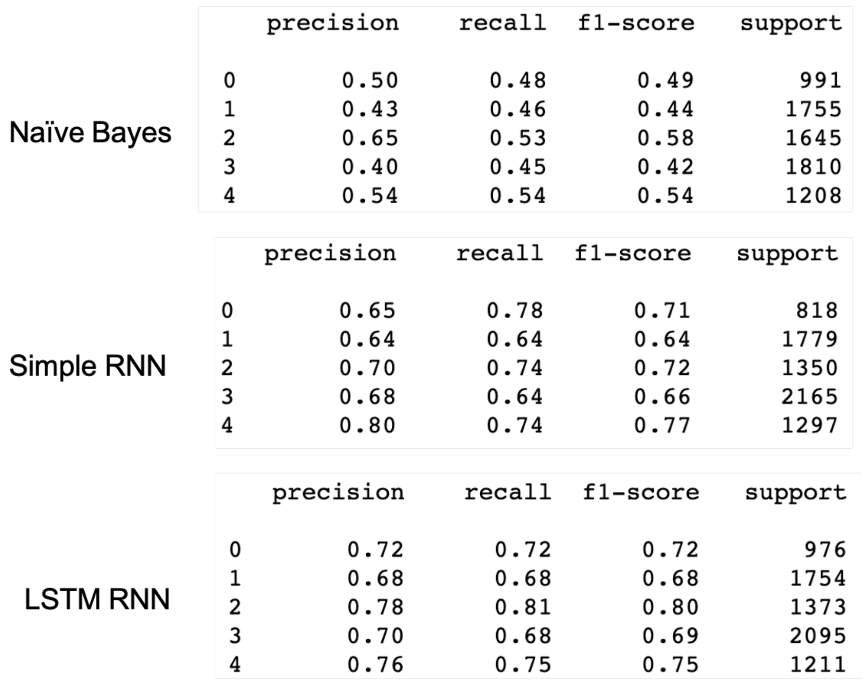
There are total 3 models, the first one is multinomial naive bayes, second one is simple RNN, and the third one is RNN with LSTM layer. I utilize Micro F1, Macro F1, and the accuracy score. The result is tested based on the validation dataset. In total, there are 7409 records of tweets in validation dataset. Summary of scores on each model shows on TABLE II..

TABLE II. Result for Each Models

Model	Micro F1	Macro F1
Naive Bayes	0.4862	0.4943
Simple RNN	0.6908	0.6995
LSTM RNN	0.7233	0.7293

Before we are getting to know the results, I will introduce the Micro F1 and Macro F1. In the binary classification confusion matrix, we will have a 2*2 matrix, so the F1 score is calculated by $\frac{2*Precision*Recall}{Precision+Recall}$, but in multi-class classification, we do not calculate an overall F-1 score. Instead, we use the F-1 score per class. The micro F1 calculates values globally and takes the weight of the metric towards the largest one. While the macro F1 tends to average each class's F-1 score, macro F1 is insensitive to the classes' imbalance[3]. What we can see from the TABLE II. is that for each model, it's micro F1 is slight lower than macro

F1, but very close. It means our validation set is very balance. Besides, the LSTM RNN model has both micro F1 and macro F1 larger than the simple RNN model, and simple RNN has both micro F1 and macro F1 larger than the Naive Bayes model. Therefore, we will tend to choose the third model. Before we really decide which model, we can always see more detail, such as looking at the precision rate and recall rate for each label in every model.



Naïve Bayes

```
   precision    recall  f1-score   support

0       0.50      0.48      0.49       991
1       0.43      0.46      0.44      1755
2       0.65      0.53      0.58      1645
3       0.40      0.45      0.42      1810
4       0.54      0.54      0.54      1208
```

Simple RNN

```
   precision    recall  f1-score   support

0       0.65      0.78      0.71       818
1       0.64      0.64      0.64      1779
2       0.70      0.74      0.72      1350
3       0.68      0.64      0.66      2165
4       0.80      0.74      0.77      1297
```

LSTM RNN

```
   precision    recall  f1-score   support

0       0.72      0.72      0.72       976
1       0.68      0.68      0.68      1754
2       0.78      0.81      0.80      1373
3       0.70      0.68      0.69      2095
4       0.76      0.75      0.75      1211
```

FIG. 5. Precision Rate and Recall Rate

The meaning of precision is the fraction of the tweets correctly classified divided by the total number of elements that actually belong to that class. The meaning of recall is the fraction of the tweets correctly predicted divided by the number of all relevant samples. Relatively, we will take the precision rate more important than recall rate because it measures the percentage of tweets flagged as a particulate sentiment that were correctly classified. Especially in the negative sentiment and extremely negative sentiment. To correctly classify those labels, let us know what people are complaining about and what makes them feel so negative. This can help plan the next moves by various federal agencies. From FIG. 5 we can see that LSTM RNN model has the highest precision on class 0 (extremely negative) and class 1 (negative).

In summary, we will purpose to use the third model as the final result, because it achieves the highest precision rate for label 0, label 1, label 2 and label 3 and the overall F1 score are also higher than other two models.

This report works on Twitter messages and utilizes both Naive Bayes, Recurrent Neural Networks (RNN), and RNN Long Short-Term Memory (LSTM) architecture to sort the messages into five categories. This classification will help us know how people's perception changes and will help federal agencies plan for the future. The result shows that RNN and RNN with LSTM can give good precision results. But the best result is RNN with LSTM because it can deal with longer sentences.

V. CONCLUSIONS

In the final analysis, the LSTM RNN model gives the best result. It gives 0.7233 micro F1 score, and 0.7293 macro F1 score, also it achieves 0.66110 accuracy on Kaggle.

The pitiful of the first two models are: the Naive Bayes model has an assumption that each word will appear independently, but in reality, the occurrence of words will have some correlation with others. The simple RNN model improves text classification accuracy because it has connections between nodes along a sequence. However, the RNN model has limitations when applied to longer sentences because it does not remember the context and meaning.

By looking at the precision rate and recall rate on FIG. 5, the precision rate of class 4 (extremely positive) on the LSTM RNN model is lower than simple RNN, but the other 4 classes are higher than the simple RNN model. Besides, the overall F1 score also increases by using LSTM architecture on TABLE II, it is because the LSTM layer on the model can help the machine to memory longer sentences.

DATA AVAILABILITY

Data is available at: Github

CODE AVAILABILITY

Code is available at: Github

[1] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. (O'Reilly Media, Inc., 2017).

[2] C. Olah, *Understanding LSTM Networks* (2015).

[3] R. Pahwa, *Micro-Macro Precision,Recall and F-Score* (2017).