CS584: PANDEMIC TWEET CLASSIFICATION

Ishaan Patel
Stevens Institute of Technology
ipatel9@stevens.edu

ABSTRACT

This project deals with analyzing various tweets about the COVID-19 pandemic and classifying them. Tweets have been pulled from Twitter and sentiment classification is applied.

1 Introduction

In particular, sentiment classification will be done, there are 5 class labels a given tweet can belong too: extremely positive, positive, neutral, negative, and extremely negative. Sentiment classification is an important technique of NLP that can be used to quantify emotions from a large volume of text rather quickly, which can then be used in a variety of applications. Sentiment classification has been used previously on other social media platforms as a means of quantifying human mood on specific opinions or objects. The objective of this project is too classify tweets that are about the COVID-19 pandemic into these 5 different class labels with different methods and analyze the accuracy and advantages of each method.

2 Background & Related Work

Sentiment analysis has come a long way in recent years. One of the first forays into this specific area came about in 1997, when McKeown and Hatzivassiloglou defined the term 'semantic orientation'. They achieved a precision of 90 percent with their model. In 2004, Pang and Lee apply machine learning to sentiment classification for picking out subjective sentences. They used Naive Bayes and SVM and achieved around 86 percent accuracy on their model. In 2005, Gruhl performed one of the first experiments to see if online comments had an impact on the sales of a product. He referred to sales data from Amazon, and made algorithms to predict the rise fall of specific books. He was able to conclude that positive comments led to increased sales. Another significant milestone occurred in 2012 when Kucuktunc and other researchers performed a large-scale sentiment analysis of Yahoo answers. They concluded that answers on the site varied according to user attributes, like the higher rated answers having a more neutral tone. They were also able to figure out emotions evoked from reading certain questions. The findings from this experiment were eventually used in various advertising and recommendation applications. Since then, sentiment analysis has evolved to be more useful in a variety of applications. [4] However, it is not a perfect technique. Since then, sentiment analysis has evolved to be more useful in a variety of applications. It remains a challenge for computer models to classify input text from the internet accurately, especially given how the context of an online message can significantly impact the sentiment of said message. It is why I want to look at the sentiment analysis problem with different modern techniques and figure out which method makes the most sense or is the most practical.

Kaggle provides two different sets of data pulled from Twitter related to the COVID-19 pandemic. The first data set contains 41,157 tweets, which are used as the training data. They each have corresponding sentiments to go with them. The second data set contains 3,798 tweets and the sentiments are not provided. [5] The training and testing data are both stored in separate csv files that have to be parsed properly.

3 Approach

The goal here for my project is to compare different approaches to sentiment classification and analyze them. Specifically, I want to look at the accuracy, efficiency, and the advantages of these different methods and draw a conclusion as to what method works the best. I will be using 3 main methods for solving the sentiment classification

problem. The first method involves building and training a simple recurrent neural network. The second method deals with building and training a bidirectional recurrent neural network. This is a more complicated variant of a recurrent neural network in which a forward and backward recurrent neural network work together. The last method I use is a 1 dimensional convolutional neural network. For me to compare these different neural networks, the problem needs to be solved properly in order to have good results to work with. Another one of the goals of this project as a result is to solve the Kaggle problem properly. This means the evaluation metrics must be valid and not have any issues. I go into more detail on the evaluation metrics below.

4 Experimental Design

4.1 Initial Setup

I wrote my program using the text editor Visual Studio Code, and utilized a Jupyter .ipynb notebook running Python 3. The main libraries I imported in are as follows:

- Sklearn - For splitting the training data into training and validation data sets
- Pandas - For parsing the .csv files
- TensorFlow/Keras - For building neural networks
- NumPy - For data manipulation
- NLTK - For preprocessing

As mentioned before, I used the data sets provided from Kaggle for the training and testing data. They were stored in two .csv files respectively. Both files contained attributes like username, screen name, location, date of tweet, and the actual tweet itself. The file with the training data contained the sentiments for tweets as well. The test dataset does not contain these sentiments, that must be derived. I parsed the two .csv files and pulled the original tweets from both data sets, and the sentiments from the training data set. I then converted the sentiments to labels and split the training data into training and validation data sets. I used 75% of the training data for the training data and 25% of the training data for the validation data. My training data contained 30,877 tweets, my validation data contained 10,290 tweets, and my test set contained 3,798 tweets.

4.2 Data Preprocessing & Word Embeddings

I then preprocessed the data I had. Standard preprocessing includes removing things such as punctuation, numbers, links, and making the text all lowercase, all of which I did. Data from Twitter requires additional cleaning however, due to the fact that tweets often contain things like hashtags or mentions of other users. I removed words beginning with hashtags or mentions of other twitter users from the data in addition to the previous preprocessing. I also opted to remove stopwords from the data as well, which are the most common words in the English language.

After preprocessing the data, I obtained word embeddings for my data sets. To do this, I fitted a tokenizer on the training data and converted the text in all my data sets to sequences. It should be noted I used the tokenizer from Keras, and that the tokenizer keeps a max of 5000 words. I decided to do this in order to exclude the most rarest and less frequent words in an effort to boost accuracy. I also made sure to pad my embeddings so my training, validation, and test embeddings were the same length for each sentence. I also built a vocabulary from the training data, I use the length of the vocabulary for one of my inputs in my neural network models.

4.3 Neural Network Overview

I built 3 different models for this project: a simple recurrent neural network (RNN), a bidirectional RNN, and a 1 dimensional convolutional neural network (CNN). Each model has it's own parameters that can be modified in order to minimize loss and increase accuracy. Some of these parameters include the number of epochs (number of iterations the network will run for), the batch size and the embedding size. For each model, each epoch records the loss, validation loss, accuracy, and validation accuracy. The goal is to minimize loss and have high accuracy without validation loss increasing, which is overfitting. I then made a graph for each model comparing the loss and validation loss, in an effort to visualize any potential overfitting issues and adjust my parameters accordingly. After that I obtain the predictions for the test dataset for each model and store the results into a new .csv file. The last part is needed for evaluation which I explain below

4.4 Evaluation Metrics

The main metric of evaluation to determine the success of the neural network models is the F_1 score. The F_1 score is the standard F score that takes the harmonic mean of the precision and recall of the model [2]. To understand F_1 score however, an understanding of precision and recall is required. Precision in this context can be defined as the ratio of a class of data that is correct. Recall can be defined as the ratio of data that belongs to the class and was classified correctly. This can be tricky to understand, but precision essentially looks at the accuracy of the predicted classes, while recall looks to see if the samples that actually belong to the class were classified properly [3]. Precision and recall both matter in evaluating the effectiveness of a model, but they also have an inverse correlation with each other. That is, a high precision means low recall and vice versa. The F_1 score is computed as such:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{1}$$

Something important to note though about the F_1 score: the formula is meant for binary classification, meaning it only works for distinguishing between 2 classes. Precision and recall are the same way as well. However, my project involves 5 different classes, so some additional work has to be done. The F_1 score for a multi-class problem can be computed by averaging the F_1 score for each different class [1]. Precision for binary classification is computed as:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2}$$

Recall for binary classification is computed as:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3}$$

So the F_1 score can also be shown as:

$$F_1 = \frac{1}{n}\sum_{i=1}^{n} 2 * \frac{precision_i * recall_i}{precision_i + recall_i} \tag{4}$$

Where n represents the total amount of classes and the precision and recall values depend on each class. The F_1 score goes from 0 to 1, 1 being the best value. So the goal is to obtain high F_1 scores with each model, which confirms it's effectiveness.

Another objective of my project is to compare different methods to sentiment classification and this involves looking at other aspects besides the F_1 score. Factors such as how long each model takes to train, loss, and accuracy can be considered in judging different networks. There are also unique aspects of each network that can be looked at as well, I'll go more into depth on each network in a later section.

5 Experimental Results

5.1 Simple Recurrent Neural Network

The simple recurrent neural network is a good model to start with: it is not overly complex and it is relatively easy to work with. It consists of 3 layers: an embedding layer, a LSTM layer, and a dense layer. The model structure and loss is displayed below:

Layer (type)	Output Shape	Param #
embedding_60 (Embedding)	(None, None, 64)	320000
lstm_55 (LSTM)	(None, 64)	33024
dense_60 (Dense)	(None, 5)	325

Total params: 353,349
Trainable params: 353,349
Non-trainable params: 0

Figure 1: Summary for the simple RNN

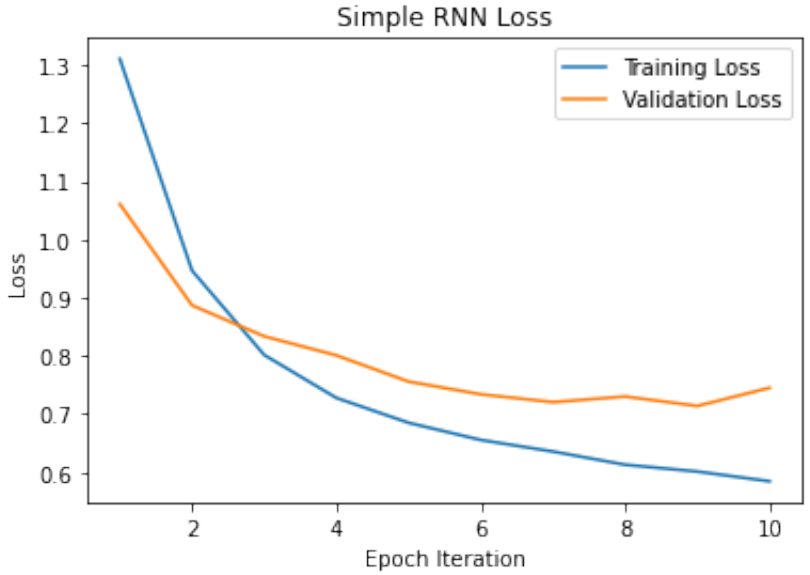

Figure 2: Training vs Validation loss for simple RNN

The validation loss does creep up a bit at the end and overfits a bit but for the most part, the model fits relatively fine. One advantage this RNN model has is that it can remember information through time due to the LSTM layer. It reduces the problem of the vanishing gradient. The vanishing gradient is the idea that when backpropagation through the network occurs, the gradient value progressively gets smaller. Another advantage I found was that this model was quick enough for training, which can be attributed to the fact that the model is not complex. It should also be noted I used dropout, 50%, to improve accuracy. I received an F_1 score of 0.8094.

5.2 Bidirectional Recurrent Neural Network

The bidirectional recurrent neural network is a more complicated variant of the recurrent neural network. It essentially involves a forward and a backwards recurrent neural network working together in order to achieve more accurate results. This model also contains an LSTM layer like the previous model in order to remember information as time progresses. Dropout was used before like the previous model in order to improve accuracy as well. Given the additional complexity of this model, another advantage is the theoretical increased accuracy of this model compared to a more simpler recurrent neural network architecture. The trade-off however of improved accuracy and increased complexity is that this model takes longer to train. The model structure and loss progression are shown below. Like

before there is a bit of overfitting at the end with validation loss rising up a bit but I accepted the model nonetheless. I obtained an F_1 score of 0.80106 from this bidirectional network.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_61 (Embedding)     (None, None, 64)          320000
_________________________________________________________________
bidirectional_6 (Bidirection (None, 128)               66048
_________________________________________________________________
dense_61 (Dense)             (None, 5)                 645
=================================================================
Total params: 386,693
Trainable params: 386,693
Non-trainable params: 0
```

Figure 3: Summary for the bidirectional RNN

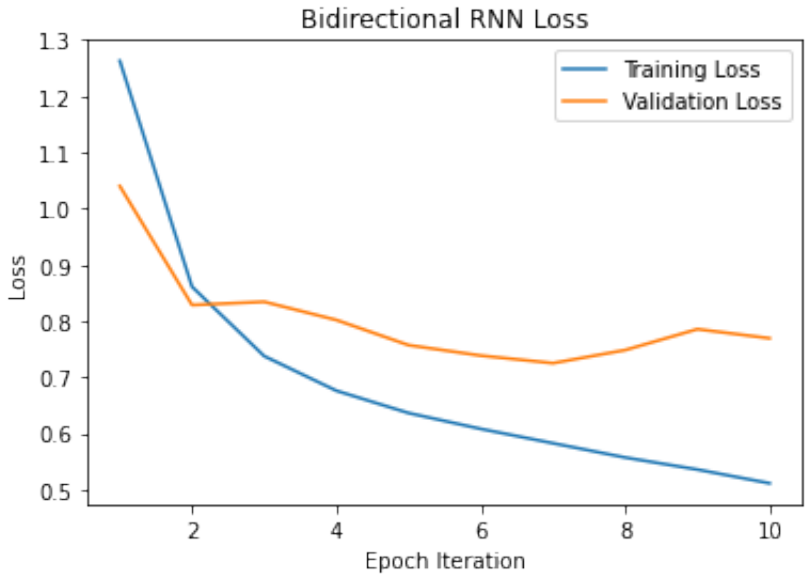

Figure 4: Training vs Validation loss for the bidirectional RNN

5.3 1-Dimensional Convolutional Neural Network

I found that my convolutional neural network trained very fast, I was honestly surprised at how much faster it was compared to my other models. The tradeoff though for this speed it seems is accuracy, this network is less accurate than the recurrent neural networks. Another disadvantage with this model is that it overfit very quickly. I had to implement some regularization and dropout in order to mitigate the overfitting. Even so, the model overfits a tiny bit at the end. I received an F_1 score of 0.78292 for this model, confirming that the model is less accurate then the recurrent neural networks. The model structure and loss progression of this model are depicted below.

Layer (type)	Output Shape	Param #
embedding_80 (Embedding)	(None, 41, 41)	205000
dropout_5 (Dropout)	(None, 41, 41)	0
conv1d_18 (Conv1D)	(None, 22, 64)	52544
global_max_pooling1d_13 (Glo	(None, 64)	0
dense_80 (Dense)	(None, 256)	16640
dropout_6 (Dropout)	(None, 256)	0
dense_81 (Dense)	(None, 5)	1285

Total params: 275,469
Trainable params: 275,469
Non-trainable params: 0

Figure 5: Summary for the CNN

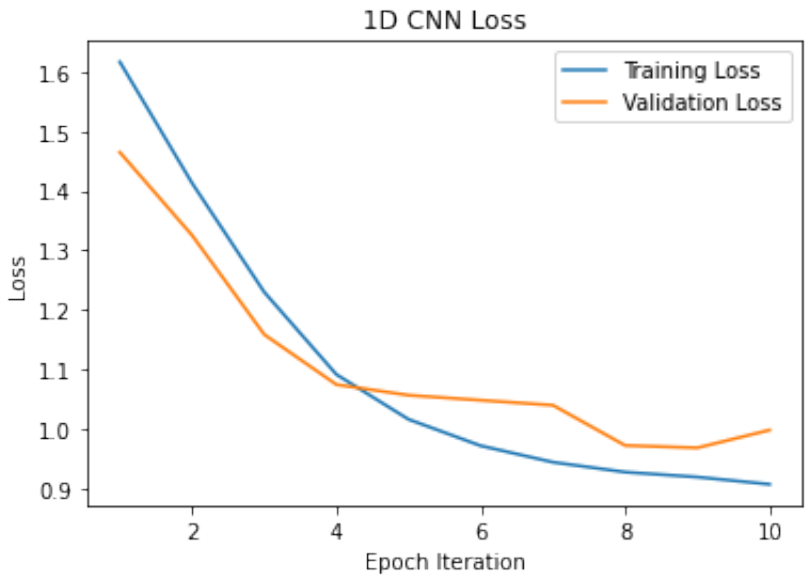

Figure 6: Training vs Validation loss for the CNN

5.4 Analysis of Results

There are several different metrics to look at and analyze. Here is a table of the relevant metrics.

Neural Network	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy	F_1 Score
Simple RNN	0.5844	0.7449	0.8047	0.7504	0.8094
Bidirectional RNN	0.5115	0.7694	0.8281	0.7463	0.80106
1D CNN	0.9073	0.9982	0.7403	0.7209	0.78292

Table 1: Loss, Accuracy, and F_1 score for each model

The results here are less interesting than I expected them to be. Based on the metrics here, it seems that the 1 Dimensional Convolutional Neural Network appears to be the least accurate model of the bunch. It has the highest training loss, the lowest accuracy, and the lowest F_1 score as well. The CNN model also had the lowest amount of trainable parameters compared to the other models, so this finding lines up. CNN is indeed less accurate than the recurrent neural networks, but also faster to train. The recurrent neural networks surprisingly don't contain any significant difference. The loss values, the accuracy scores, and the F_1 score of these two networks are approximately the same. I would have expected the bidirectional network to be more accurate than the simple RNN, especially given the longer training time. As a result of the data being less varied than anticipated, the only real takeaway I can gather from these metrics is that the convolutional neural network is less accurate but quicker to train then a recurrent neural network, and that there was no real difference between running a simple or bidirectional RNN.

Another metric to look at involves the distribution of the training data. Each model is intended to classify the test samples into 5 different sentiments: extremely negative, negative, neutral, positive, or extremely positive. We can plot the distribution of tweets into these 5 categories and check for any significant differences in how each model distributes the data.

Neural Network	Extremely Negative	Negative	Neutral	Positive	Extremely Positive
Simple RNN	523	1006	616	1029	624
Bidirectional RNN	504	1018	631	1084	561
1D CNN	397	1152	712	1067	470

Table 2: Sentiment Classification Results

The data here is not particularly interesting either. Across all 3 models, negative and positive sentiments were the most frequent, while extreme sentiments and neutral sentiments were less frequent. It seems that the tendency across all three models is to prefer less extreme sentiments compared to the more stronger sentiments. The convolutional neural network model has the most difference here, it classified more tweets as negative, neutral, or positive compared to the recurrent neural networks. It is also the least accurate model too, suggesting that it misclassified some tweets as being negative, neutral, or positive when they are not. The recurrent neural networks do not have much of a significant difference in terms of the distribution, they're very similar. This lines up with the models being very similar in accuracy.

6 Conclusion & Additional Remarks

6.1 Potential Data Issues

A potential problem with this project and the Kaggle problem is that the data provided by Kaggle may not be entirely accurate. Kaggle provides the F_1 scores of people who have made submissions for this problem, and some people have F_1 scores of 1.0. This would mean their models are exactly correct with no errors. Having a neural network that makes 0 mistakes on the test samples is highly unlikely however. Kaggle does not provide the labels for the test data set unfortunately, meaning that I do not have much to go off of in verifying that the data is legitimate. Something that occurred to me though is that it is possible the training data is not balanced. This means that the sentiments in the training data are too heavily skewed towards one or a few class and do not represent a fair distribution. This would certainly throw off the models since training the models would work best provided they have ample data for all possibly classes. Here is a table showing the distribution of the training data.

Extremely Negative	Negative	Neutral	Positive	Extremely Positive
5,481	9,917	7,713	11,422	6,624

Table 3: Training data distribution

Based on the distribution of the training data, it seems that the spread of samples among different classes is not exactly balanced. There are less tweets belonging to extreme sentiments and more tweets belonging to the negative, neutral, and positive classes. This pattern can be seen in how each model classifies the test data, with more tweets going into negative, neutral, and positive classes and less tweets going into the extreme sentiments. This lack of balance in the training distribution could definitely be a reason for the models to perform improperly. I also looked through the training data .csv to check the data itself, and it appeared to be valid with no glaring inconsistencies.

6.2 Remarks & Future work

For this problem, Kaggle only provided a single F_1 score for evaluation. It would have been nice to see additional information like the sentiments for the test data set and precision/recall scores for each different class. I would have been able to add further information to the report with those metrics and draw further conclusions. I learned a lot from this project and if I had the chance to do this project again, something I would do is try to reduce the overfitting more to ensure improved accuracy. All of my models overfit slightly towards the later epochs and I deemed it acceptable at the time, but the overfitting could have definitely been reduced. I could have also added additional layers into my model to increase complexity and improve the accuracy of the models as well.

References

[1] Baeldung. F1-score for multi-class classification. URL `https://www.baeldung.com/cs/multi-class-f1-score`.

[2] DeepAI. F-score. URL `https://deepai.org/machine-learning-glossary-and-terms/f-score`.

[3] Google Developers. Classification: Precision and recall. URL `https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall`.

[4] Devopedia. Sentiment analysis, 2021. URL `https://devopedia.org/sentiment-analysis`.

[5] Kaggle. Pandemic tweet challenge, 2021. URL `https://www.kaggle.com/c/pandemic-tweet-challenge/overview`.