Real or Not? Natural Language Processing With Disaster Tweets

Shifa Shaikh, Rojin Zandi

EE258 – Neural Networks

Project-2

Fall-2020

Abstract

The main goal of the project is to acquire highest accuracy and performance on the given dataset. The dataset contains tweets related to disaster and not related to disaster. Basically, the idea is to get a model that can classify data into disaster tweets and non-disaster tweets. As we are dealing with tweets (text data), we use LSTM (Long-Short Term Memory) model as our first model to train it from scratch and use another pre-trained model; GloVe (Global Vectors for Word Representation) to note the results. Additionally, we try to improve the performance of the pre-trained model by using an early stopping regularization technique. Our results conclude that the given dataset performs better on a pre-trained model as we have limited amount of data for the training.

Keywords: Disaster Tweets, LSTM, GloVe, pre-trained, early stopping

Contents

Abstract 2

Dataset Description 4

Data Visualization 5

Data Cleaning and Pre-processing 7

Data Cleaning 7

Splitting of Data 7

Pre-processing 8

Models Utilized 9

Model: 1 9

Model-1 Training Inference 10

Model-1 Improvements 11

Model: 2 11

Model-2 Training Inference 12

Model-2 Improvements 13

Summary 15

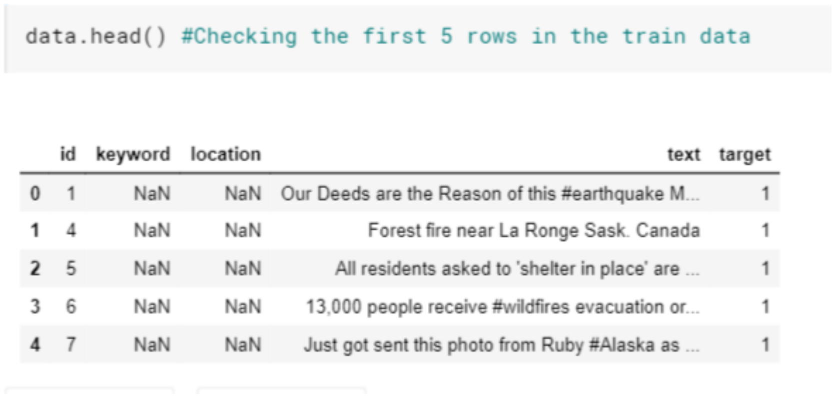
Discussions & Conclusions 16

References 17

Dataset Description

We got the dataset by entering one of the Kaggle challenges. The challenge can be found here. The dataset contains three .csv files: train, test, and sample_submission. The train and test files are for training and testing the model, respectively. The submission is for submitting the results on the test data.

Before training we analyze the dataset using Pandas, Matplotlib, and Seaborn libraries. The figure-1 below shows the first five rows in the train file. As we can see there are total five columns (features) and those are id, keywords, location, text, target. The column 'target' indicates if the tweets (in text column) is a disaster tweet (target == 1) or not (target == 0). We are given inputs and the corresponding output, so our application is based on Supervised Learning.

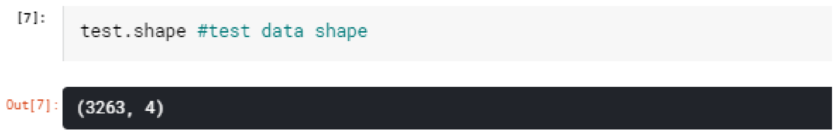


```
data.head() #Checking the first 5 rows in the train data
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

Figure-1: First five rows from training data

The train set has approximately 7.6k datapoints while the test set has about 3.2k datapoints. Conversely, the test dataset lacks one column i.e., 'target'. Our goal is to predict or classify the real disaster tweets in the test set and then submit the predictions. The figure-2 shows the shape of the test set.



```
[7]:  test.shape #test data shape

Out[7]:  (3263, 4)
```

Figure-2: Test Dimension

Data Visualization

This section describes the data visualization techniques used in our code. The figure-3 shows the count plot of the words used in 'keyword' column. The plot is of top 10 keywords used in each tweet. From the results, we can assume that the word 'fatalities' is used the most in our data. The plot is irrespective of the disaster or non-disaster tweet.

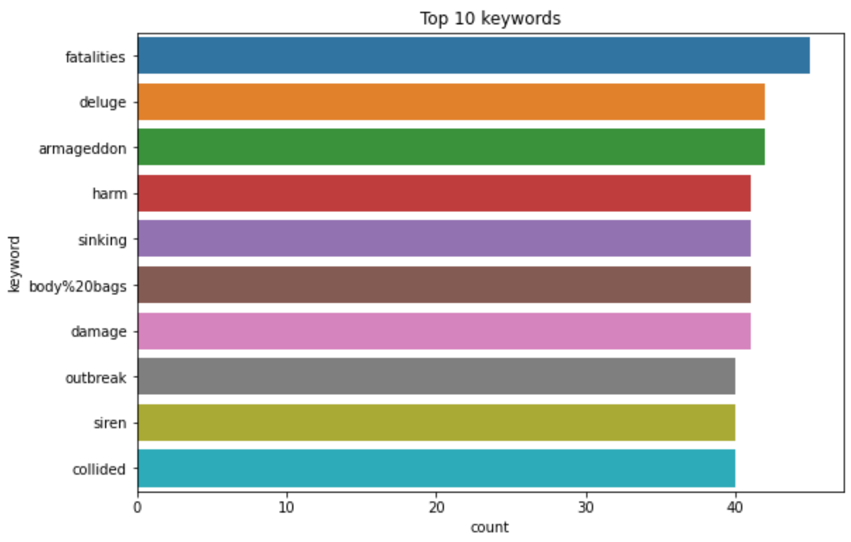


Figure-3: Top 10 keywords in the training dataset

The figure-4 displays two bar plots: cyan colored plot for real disaster tweets and yellow plot for non-disaster tweet. The Y-axis shows the words used in 'keywords' and X-axis shows the number of times the 'keywords' appear in the 'target'. The disaster tweets contain keywords like outbreak, wreckage, typhoon, etc while the non-disaster tweets have keywords like harm, Armageddon, fear, etc.

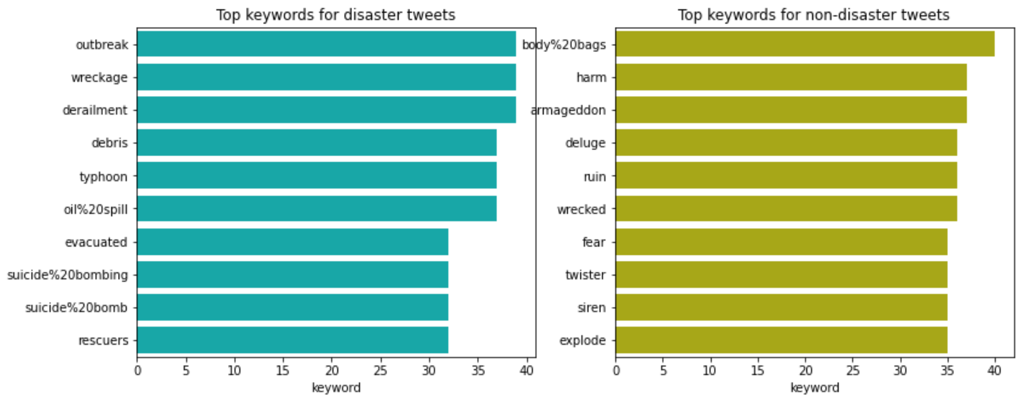


Figure-4: Disaster Tweets and Non-Disaster Tweets Keywords

We now explore different locations in the tweets. Just as we plot count plots and bar graph for ‘keywords’, we try to check the ‘locations’ and learn about it. The figure-5 below describes the data evaluated from the training dataset.

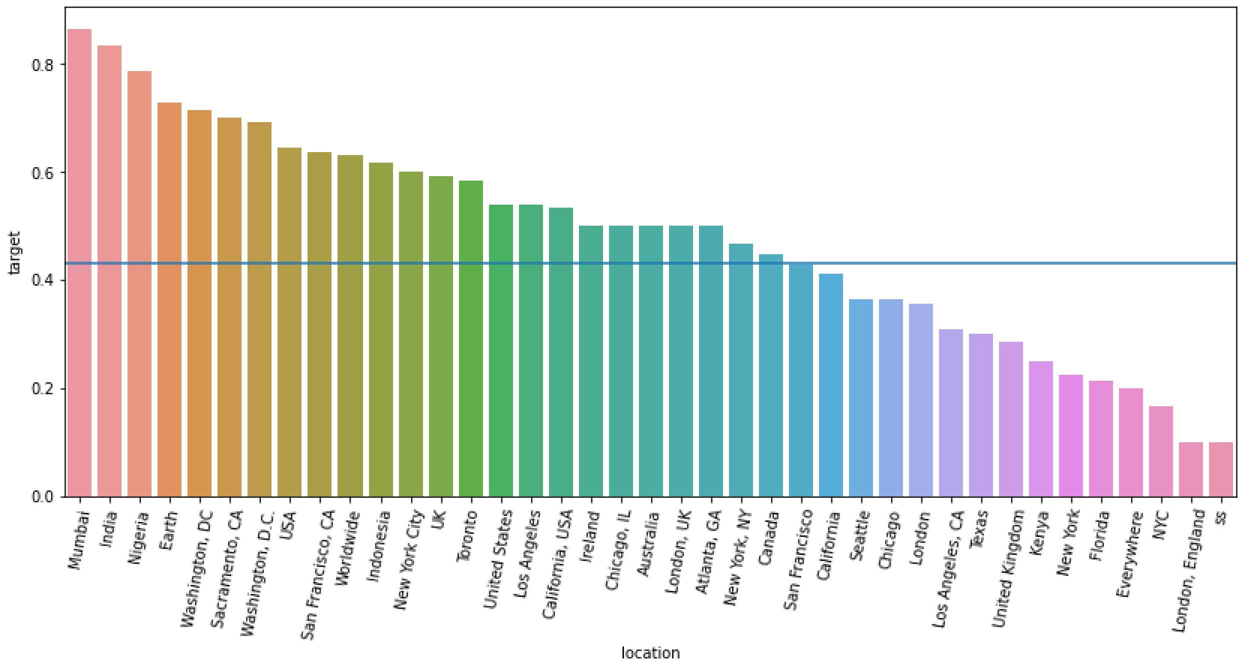


Figure-5: Locations in the training dataset

Data Cleaning and Pre-processing

A text data can contain unnecessary words like stop words, punctuation marks, URLs (Uniform Resource Locator), etc. Nowadays, people even use emojis in the text to express their feelings. People also tend to use contractions in the text such as can’t, shouldn’t, didn’t, etc. Such type of words or symbols can confuse our model during the training. To ignore or remove such words from our text we use NLTK library. Also, the computers/models only understand numbers, so we must convert the text data into a data that is understandable by the model. After the cleaning process we convert the text into vectors using TensorFlow/Keras preprocessing API.

Data Cleaning

We create a function: clean_text() to carry out the data cleaning process. The function processes the tweet ‘text’ and returns a clean text for further processing. It removes the stop-words, punctuation marks, URLs, etc. from it. At the end we remove the emojis from our text and return a cleaned version of it. We referred to GitHub for replacing the emojis. The data cleaning process first starts with converting the text into lower case, then replacing the contractions with the corresponding full words and then calling the clean_text function. The data cleaning is done on the ‘text’ column in both training as well as test files.

Splitting of Data

For cross-validation purpose, we split our training data into a train set and a validation set. We choose the ratio of 80:20 for train and validation to get insights of the training and estimate the test accuracy.

Pre-processing

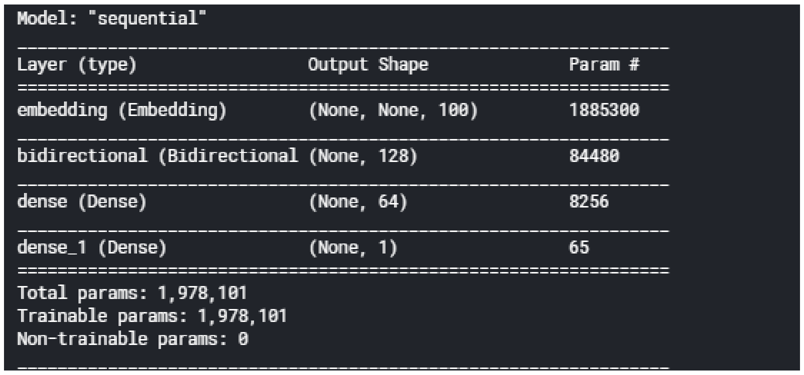
To convert the text data into computer understandable format we convert the texts into vectors. The TensorFlow and Keras has Tokenizer API for the pre-processing of text data. It allows the text to vectorize by turning each text word into a sequence of numbers. We fit Tokenizer on our train set. Then we convert the tokenized data into sequences. The generated sequences are of different lengths, so we pad it by adding zeroes into the sequences to make all the sequences equal. If the sequences exceed certain maximum length the sequences are truncated. Here we have kept the limit as 20. We did post-padding and post-truncating, vice versa is also possible. Similarly, we undergo the same pre-processing for validation and test datasets. We even define a function at the end to convert the integer values or vectors back into text sequences.

Models Utilized

We implemented total 2 models excluding the sub-models. The description of each model is given below. The architecture and the description will be followed by the training inferences and results we acquired. Also, we discuss what steps we took to improve the performance.

Model: 1

We start with an Embedding layer with total number of words as one of the arguments. It works as an argument for the input_dim parameter. The next layer is a bidirectional LSTM layer with units equal to 64. We even set recurrent dropout to 0.1 to overcome overfitting issues in RNN. The LSTM layer is followed by two Dense layers. The last layer uses one output neuron with a sigmoid activation function for classifying real disaster tweets and non-disaster tweets. The figure-6 below shows the summary of the Model-1.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         1885300
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               84480
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,978,101
Trainable params: 1,978,101
Non-trainable params: 0
_________________________________________________________________
```

Figure-6: Model-1 Summary

Other parameters:

- Learning rate: 0.0001
- No. of epochs: 15
- Optimizer: Adam
- Loss: Binary cross entropy because we have two classes

Maximum Accuracy:

- Training: 99.39% and its corresponding Validation accuracy: 74.52%
- Validation: 77.54% and corresponding Training accuracy: 88.21%

Model-1 Training Inference

The figure-7 below shows the accuracy vs epoch and loss vs epochs plot. We can clearly see that the training accuracy (blue colored curve) and validation accuracy (red colored curve) has overfitting issues. Also, the validation loss increases as we train the model for more number epochs.

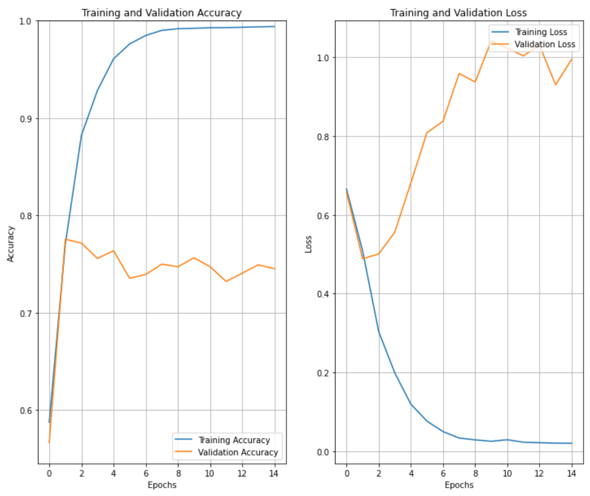


Figure-7: Accuracy vs Epochs and Loss vs Epochs

The figure-8 is a confusion matrix of the results. From the figure we can see that the TP (successfully predicted real disaster tweets) is 478 which is lower.

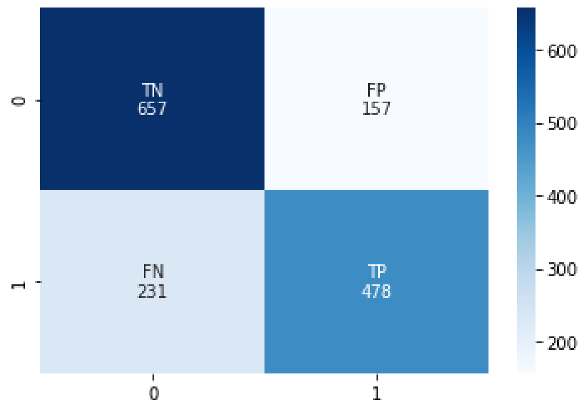


Figure-8: Confusion Matrix of Model-1

Model-1 Improvements

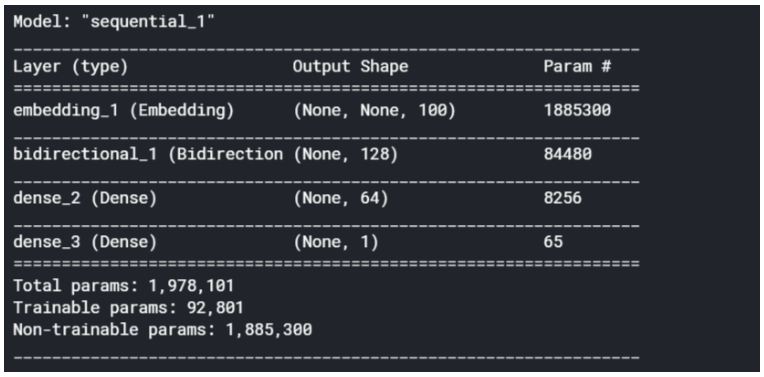
We tried changing different parameters to improve the performance. For the baseline model, we used a few Convolution layers with LSTM layers. We got around 57% of accuracy (according to the presentation submission) for training as well as validation. We did not see any overfitting issues with the model. But when we increased the complexity and number epochs, there was no change in the accuracy. It was constant although trying everything. Then we figured that our pre-processing has some issues. We changed the pre-processing part and trained the model. After reducing complexity and changing other parameters we got maximum accuracy according to the Model-1, though this model still possesses overfitting problems. Using drop-out layers, L1 and L2 regularization techniques did not show any significant progress. The overfitting issue can even be due to lesser amount of training data. So, we conclude that the due to lack of training examples the model is unable to converge.

Model: 2

Th transfer learning can be advantageous when we have a smaller amount of training data. We work out transfer learning for the dataset. The architecture and the parameters of

Model-2 are analogous to the Model-1. The only difference is that the Model-1 was trained from scratch by us according to the previous section but here in Model-2, we use a pre-trained model: GloVe (Global Vectors for Word Representation).

We configure the embedding initializer for the model according to our dataset by forming an embedding matrix. The summary of the Model-2 is shown below in the figure-9. We train the model with same number of epochs.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         1885300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               84480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,978,101
Trainable params: 92,801
Non-trainable params: 1,885,300
_________________________________________________________________
```

Figure-9: Model-2 Summary

Maximum Accuracy:

- Training: 86.81% and its corresponding Validation accuracy: 81.75%
- Validation: 82.01% and corresponding Training accuracy: 84.75%

Model-2 Training Inference

From the figure-10, we can still see the over-fitting issues. But it is comparatively low than the Model-2. The model converges faster. Also, we can see the improvement in the confusion matrix (figure-11). The last model had TP = 478 and for Model-2 the TP = 535. The TN values also increased significantly.

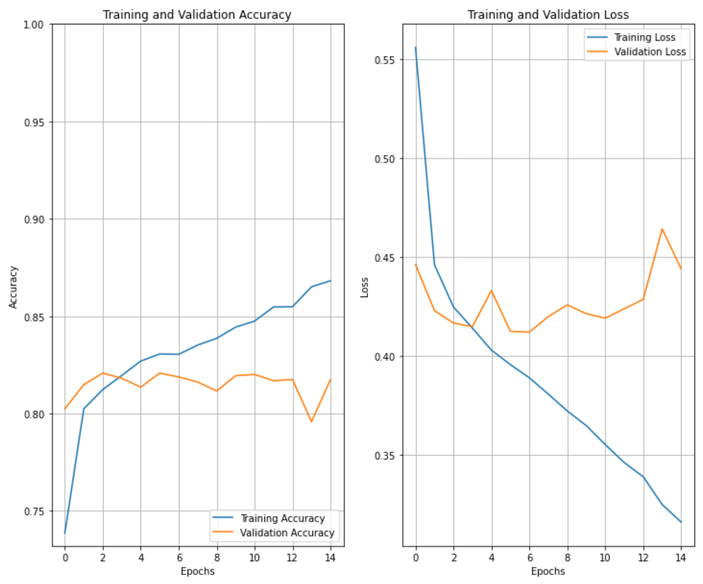


Figure-10: Pie chart results

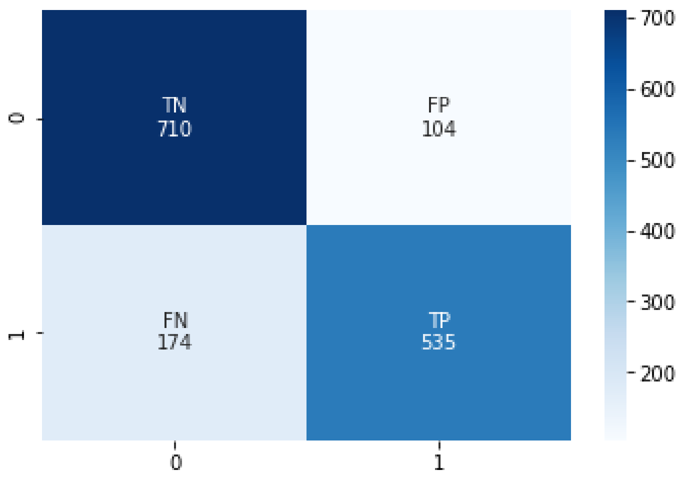


Figure-11: Confusion Matrix Model-2

Model-2 Improvements

To overcome the overfitting, we try to implement Early Stopping regularization technique. The training is like the previous models, we just add one additional parameter for the early stopping. The code is as follows:

```
#configure the early-stop
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=5,
verbose=0,restore_best_weights=True, mode='auto')

#train the Model-2 again but this time with early-stop
history22 = model2.fit(training_padded, y_train,epochs=EPOCHS,batch_size=12,
validation_data=(validation_padded, valid_labels), verbose=2,callbacks=[earlystop])
```

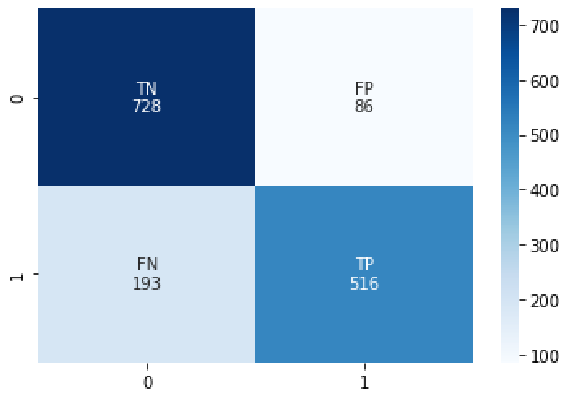


Figure-12: Confusion Matrix of Model-2 with early stopping

If we compare the confusion matrices of Model-2 with (figure-12) and without early stopping criteria then we can see that we improved the detections of TN but degraded the performance of the TP. Additionally, with the increase in TN we decreased the number of FP. In a nutshell, the early stopping technique did not show any significant improvements in the performance.

Summary

The table shows the performance results of each model for better comparison. The table itself is self-explanatory.

Models	Training Accuracy	Validation Accuracy	Precision	Recall	TP (Real Disaster)
Model-1	99.36	75.11	[0.77 0.73]	[0.75 0.75]	530
Model-2	86.81	81.75	[0.80 0.84]	[0.87 0.75]	535
Model-2 (with early stop)	89.95	81.22	[0.79 0.86]	[0.89 0.73]	516

Table-7: Summary of performance

Discussions & Conclusions

The given dataset contains tweets posted by twitter users. It contains tweets related to real disasters. The idea is to separate the real disaster tweets from other tweets. We implement 2 models in our project to do so. Every model suffers from over-fitting problems and which cannot be reduced with regularization techniques. The training dataset contains 7.3k datapoints only which is a fewer amount of data. If only we had more data available, then we would have overcome the overfitting issue.

We conclude that Model-2 performed the best among all. The pre-trained model has advantage of overcoming the over-fitting problems as they are trained on larger dataset and we can configure the weights of the trained network to fit to our dataset.

References

1. Challenge https://www.kaggle.com/c/nlp-getting-started
2. Our code NLP Project2 | Kaggle
3. GloVe Model GitHub - stanfordnlp/GloVe: GloVe model for distributed word representation
4. NLTK Stop words NLTK's list of english stopwords · GitHub
5. Emojis Remove all traces of emoji from a text file. · GitHub