CS584: CLASSIFYING DISASTER TWEETS

Gaurav Narvwani
Stevens Institute of Technology
gnarvwan@stevens.edu

ABSTRACT

The goal of this project is to classify a given tweet as a disaster tweet or not.

1 Introduction

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies). For example the author of a tweet could explicitly use the word "ABLAZE" but could mean it metaphorically. This is clear to a human right away, especially with the visual aid. But it's less clear to a machine. But, it's not always clear whether a person's words are actually announcing a disaster[1]. This project aims at classifying tweets to find out if they are talking/announcing a disaster or if its just a regular tweet. Naturally this is also a sentiment analysis task. Sentiment analysis tasks involve interpreting and classifying subjective data using techniques from Natural Language Processing (NLP) and Machine Learning (ML). As a result of much of the world's data being in a digital format, many businesses are leveraging the power of sentiment analysis to gather more information on brand reputation, customer's thoughts towards various products, and many more.

2 Background

This topic is from a kaggle competition and has seen a lot of traction to it. Over a thousand entries and over four thousand participants have entered this competition and contributed by submitting their attempts to solve this problem. This has become a famous competition as it is a moderately challenging task for students and researchers in this field as well as it is a challenge which can greatly help society by predicting disasters sometimes faster than any other means available today.For example there's a tweet by a user stating they'd like to set "the sky ablaze". When thinking of "ablaze" as a verb, this tweet could be considered a disaster since that would mean the sky is burning fiercely. However, we know by the context provided in the tweet, the user means the adjective version of "ablaze" which means they are stating that the sky is very brightly coloured or lighted. This is pretty obvious, but not so to a computer. Hence, it's our job to come up with solutions to overcome try as best as possible to overcome the various ambiguities of language so the people we serve are not constantly bombarded with false claims of disasters due to language discrepancies.

Due to the nature of this topic, there has been a lot of research and previous work done in this area. One such example is a paper titled "NLP Disaster Tweets Classification" [2] by Kristopher Flint and Zachary Kellerman from Tennessee Tech in 2020. They started by preprocessing the data set. For this they used Term Frequency Inverse Document Frequency (TF-IDF). They also generated word clouds to visualise the results. They then began model selection and decided to use a variety of different models. They used the models Naive Bayes, Support Vector Machines, Random Forest, Ensemble Learner based on the above 3 models, Sequential Neural Network and Bert. They wanted to compare the results of these models and see how different approaches performed in this case. This approach also inspired me to try multiple approaches to my own implementation. For their first four approaches they got an accuracy of 81 and their BERT model gave them an accuracy of 84. Since BERT is Googles own framework, it made sense that the best results came from that approach. Besides this there are many articles on platforms such as medium and towardsdatascience that elaborate their approach and provide their implementation. One more such article is the one titled "NLP: My Solution to Kaggle's Disaster Tweet Competition" [3] by Tracyrenee which was a very good read and helped me develop my ideas for this project.

3 Datasets

The dataset for this project is taken from a Kaggle competition. This dataset was originally created by the company figure-eight and was shared on an open source website [4]. The dataset consists of five features. They are ID, text, location, keyword and target. The ID is an unique identifier for each tweet. The text contains the text of the tweet. The location contains the location from where the tweet was sent. The keyword denotes a particular keyword from the tweet that maybe important. The target is the output variable and contains 1 if the tweet is a disaster tweet and 0 if not. The columns location and keyword contains quite a few missing variables. The dataset is split into test and train from the beginning. The train set contains 7500 entries while the test set contains around 3200 entries.

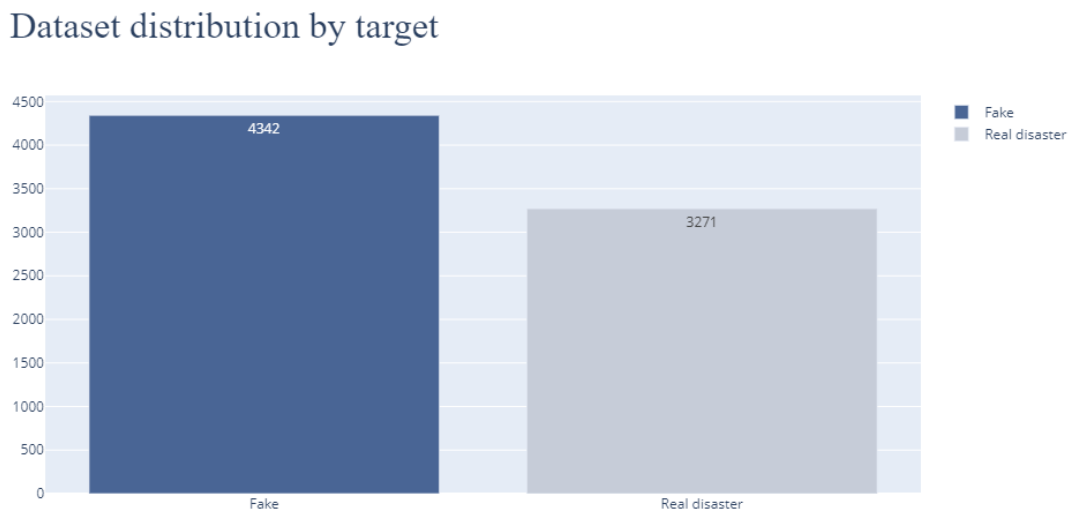
4 Your Approach

This is a classification problem and thus we need to used classification models to approach this problem. After going through previous work done in this area i was also inspired to try the approach to compare different models. I wanted these models to be completely different approaches. This would also give an insight on how these models work in such problems. This being an NLP task, I decided to try to compare a machine learning based model and a more traditional NLP based model. My objective here was not only to produce a good model for the task, but to see how does a traditional machine learning model compare to a NLP based on in this problem. If these were to be comparable, I could also build a voting classifier between them.

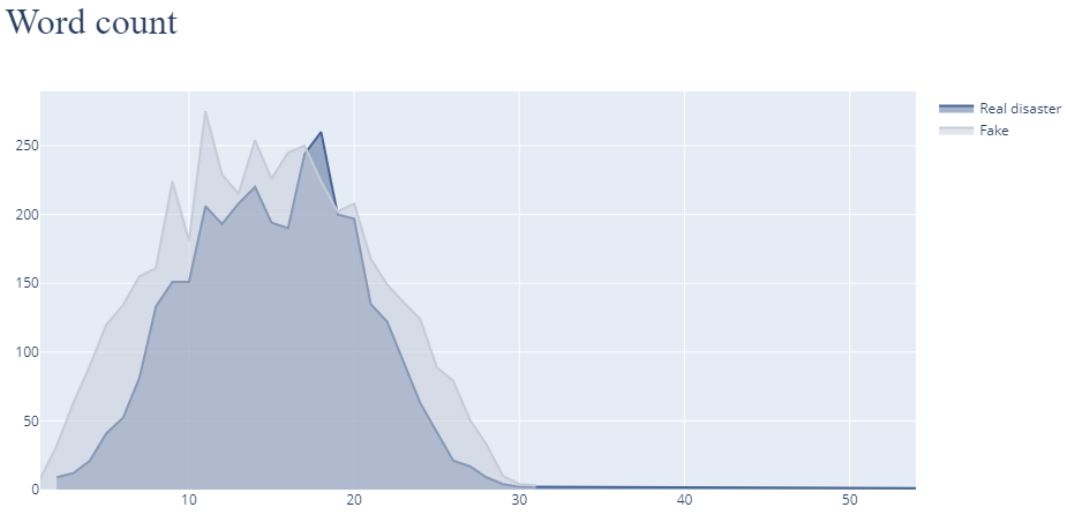
Since this is a classification task, we will be using accuracy to evaluate these models. This metric is also used by the kaggle competition themseleves to score each submission and hence this seemed like the best choice. I also wanted to plot the loss values for the NLP based approach to see how it is being trained. Existing approaches have used Google trained models such as BERT and have gotten very high accuracies in the test set. Due to time and computational constraints, I will be using much simpler architectures but will aim to get the best results using it. Once we have evaluated the test set, we can maybe understand what types of outputs have been classified wrongly and see if the model has any bias that can be removed to increase accuracy.

5 Experimental Design

My first job was to analyse the data. We had four features namely, ID, location, keyword and tweet. Also we have the target variable. From the features we firstly removed ID as it does nothing to contribute to the target variable. I then checked for missing values and found that the columns location and keyword contained alot of missing values. This would be a big problem and hence I decided to remove these features as well. The tweet alone would be good enough to predict the target variable. Now I decided to check for class imbalance. If there contained any big imbalance in the classes, then we would have to undersample the dataset which would cost our performance. Oversampling in this case would be very complicated as we are dealing with a tweet containing text and not just numbered features.



As we can see from the above graph that there are definitely more fake tweets as compared to real disaster tweets. But this distribution is still fairly balanced and we should have no problem working with this. Therefore no sampling was done to the dataset. Next I tried to visualise the word count for both the classes. My goal was to see how both the classes differed in their tweet size. This could be an interesting observation which could also be derived by our proposed model. Also for our NLP based approach, we sometimes need to specify the size of the sentence and this observation helps towards determining that.



From the above graph we can see that the fake tweets tend to be smaller compared to the real disaster tweets. Although this difference is very small and thus should not make any difference in our approach. After this I had to preprocess the dataset. I removed all the numbers, links, emojis, and punctuations from the tweets. After this I imported the list of stopwords from the NLTK library and removed the stop words from the tweets. Stopwords contain words such as prepositions etc which dont really contribute in getting the meaning of a tweet. Thus it is better to remove them for such tasks. After cleaning the datset we can now get some interesting insights from it. I decided to find the most frequently appearing words in each class. For this I removed the count of the top 10 words from each class.

```
[('fire', 266),
 ('bomb', 179),
 ('kill', 158),
 ('news', 132),
 ('via', 121),
 ('flood', 120),
 ('disast', 116),
 ('california', 115),
 ('crash', 110),
 ('suicid', 110)]
```

From the above image we can see the top 10 words appearing the real disaster tweets. These are all words which are clearly related to disasters or are even disasters themselves. These words appearing frequently help our proposed models greatly as they can find these patterns between the classes with lesser difficulty.

```
[('like', 306),
 ('get', 222),
 ('amp', 192),
 ('new', 168),
 ('go', 142),
 ('dont', 139),
 ('one', 134),
 ('bodi', 116),
 ('love', 115),
 ('bag', 108)]
```

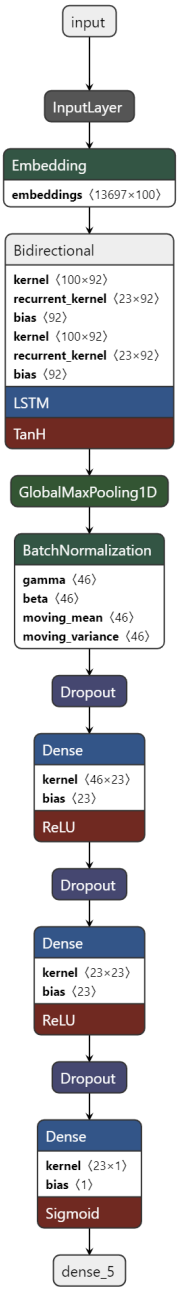
The above image shows us the top 10 words appearing in the fake tweets. Unfortunately these are just normal words and for us as humans they have no correlation to the class as a whole. Still we can hope the model can derive some relation between these to give us a better result.

Now I had to begin selecting my models. As I said above, I decided to go with two different approaches, a machine learning based approach and a more NLP based one. Since NLP problems require more complex approaches, for the machine learning based one I could use a simple model as it would just fail to find any dependencies and would end up giving poor results. Therefore for my machine learning approach I decided to use the model XGBoost.

This is an ensemble approach and one of the more complicated models in machine learning. XGBoost stands for eXtreme Gradient Boosting. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. This approach supports both regression and classification predictive modeling problems.

Naturally we cannot just pass the tweet into the XGBoost Classifier. There we had to prepare the tweet in the pipeline for implementing the classifier.For this we had to use two types of processing. The first is bag of words. This creates a frequency count for each word in the set. This was done using the function CountVectorizer. This alone wouldnt make much sense to the model. Thus our next step is converting this bag of words output into something more meaningful. For this we used the approach of TF-IDF. This was inspired from the paper described in the background section. TF-IDF, short for term frequency – inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is one of the most popular term-weighting schemes today. Converting the bag of words into TF-IDF will give the XGB Classifier the required information to train on the dataset and find relations from them. Thus using this we trained the first model which is the XGBoost Classifier.

For the second approach, i.e. the NLP based one, I decided to use a Recurrent Neural Network (RNN). RNNs are the most popular architectures for performing NLP based tasks. For this model I decided to use a Neural Network using the LSTM layer as well as Dense Layers. This is one of the more popular approaches as the LSTM model is a more commonly used one because of its ability to keep short term patterns in a memory and it is also not a very heavy model thus would not take alot of time in its training. Again for this model we cannot use the tweet as it is and thus we require some preprocessing for it. In this case it is better to create embeddings of the tweet rather than use the TF-IDF. To create embeddings, we can train our own enbeddings but due to the small train size it was better to import readily available ones. Stanford has freely available embeddings of dimensions 50,000, 100,000 and 400,000 for students and researchers to use. I decided to use the 100,000 one in this problem and imported these embeddings. After converting the tweets into embedding vectors, we can now feed them into our model.



From the above image we can see the architecture of the model. The image shows the layers present in the model as well as its parameters. The architecture follows an input layer into an embedding layer. These have the dimensions of the embedding vector which was creating during the preprocessing stage. This follows the Bidirectional LSTM layer.

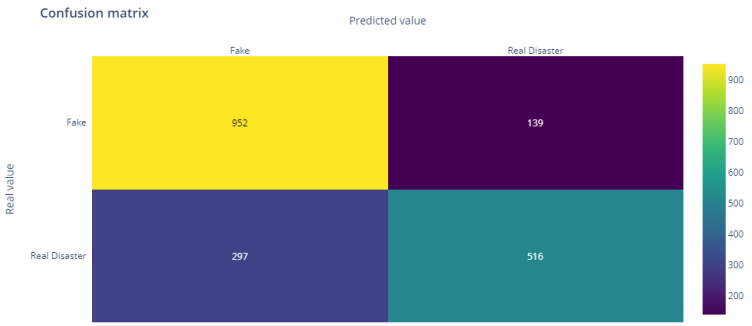
After this the GlobalMaxPooling layer helps extract the most important features and passes it on the the BatchNormalization layer which normalizes the features to process efficiently. After this are a series of Dropout and Dense layers. The Dropout layers help reduce overfitting and the Dense layers reduce the size while keeping the information gained from the above layers. These finally reach the output layer which predicts the class. The description of the model is below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 23, 100)           1369700

bidirectional_14 (Bidirecti  (None, 23, 46)            22816
onal)

global_max_pooling1d_14 (Gl  (None, 46)                0
obalMaxPooling1D)

batch_normalization_14 (Bat  (None, 46)                184
chNormalization)

dropout_42 (Dropout)         (None, 46)                0

dense_42 (Dense)             (None, 23)                1081

dropout_43 (Dropout)         (None, 23)                0

dense_43 (Dense)             (None, 23)                552

dropout_44 (Dropout)         (None, 23)                0

dense_44 (Dense)             (None, 1)                 24

=================================================================
Total params: 1,394,357
Trainable params: 1,394,265
Non-trainable params: 92
_________________________________________________________________
```

6 Experimantal Results

Now it is time to evaluate and compare the results. We ran the models on a train and test set and recorded the accuracies obtained from both.

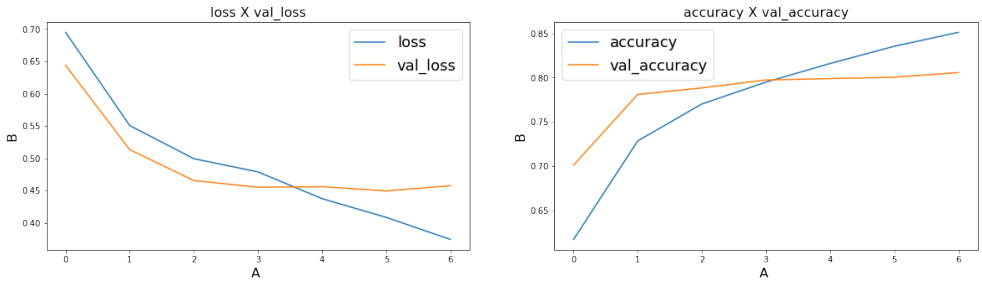
For the XGBoost model, the confusion matrix is shown below.



In this model we got an accuracy of 86 in the train set and 77 on the test set. The model performed very well on the train set but had some issues. The model clearly was overfitting on the train set and thus did not provide the best results. Reducing the number of iterations would lower the accuracy and thus give us even less desirable results.

Now for the LSTM based model, the classification report and loss graphs are below.

```
F1-score:  0.7562254259501967
Precision:  0.7105911330049262
Recall:  0.8081232492997199
Acuracy:  0.8046218487394958
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.87      0.80      0.84      1190
           1       0.71      0.81      0.76       714

    accuracy                           0.80      1904
   macro avg       0.79      0.81      0.80      1904
weighted avg       0.81      0.80      0.81      1904
```



We can see from above that the F1 score is 0.75 and the accuracy is 80. This is lower than the previous classifier. But in this case we have an accuracy of 80 as well on the test set. Thus this model did not suffer alot by overfitting and is a much more desirable model as compared to the XGBoost classifier. The F1score, precision and recall all having high values gives us confidence that the predictions were equally weighted in both classes, i.e. no specific class had a much worse prediction rate to the other and this is also another good result for us.

7 Conclusion and Future Work

Thus comparing the results the NLP based approach worked better as it gave us a better test set result but the XGBoost classifier was poor and was a decent classifier. Between the two, the LSTM model would be a better model to deploy to use for actual scenarios. This shows that machine learning models can give decent results for NLP based tasks but Recurrent Neural Networs still perform better and are rightly so much more popular in performing such tasks. Still in the future we may discover some machine learning based approaches which could compete toe to toe to the advances made in NLP architectures.

As for future approaches we could use some refining in certain areas. Since our dataset is not very large, this caused our XGB classifier to overfit and we could obtain much better results with a much bigger corpus. One approach towards this is collecting more data in other languages and running a language translation model to obtain a larger training set. Also with enough resources we could deploy state of the art architectures such as variations of BERT or GPT which are developed by the leaders in this technologies namely Google and OpenAI. For this we could used cloud services as well as transfer learning to train these architectures according to our problem.

Since this is such an open challenge on kaggle, there are submissions posted daily for this challenge and students as well as researchers are working daily to optimize as well as provide a better solution for this task. The future for this problem looks very bright with the pace of advancements made in state of the art architectures and we are very close to a day we deploy models for this task for real world use making this a huge benefit to society.

References

1) https://www.kaggle.com/c/nlp-getting-started/overview/description

2) https://publish.tntech.edu/index.php/PSRCI/article/view/686

3) https://ai.plainenglish.io/nlp-my-solution-to-kaggles-disaster-tweet-competition-9973ddf48235

4) https://appen.com/open-source-datasets/