Analysis and Prediction of Tweets for Disaster Identification

Kenil Shah
kshah9

Shahil Shah
sshah29

Urvish Vasani
uvasani

Chintan Gandhi
cagandhi

1 INTRODUCTION AND BACKGROUND

1.1 Problem Statement

Since the past few years, social media giants such as Twitter and Facebook have invested a lot of effort to weed out fake news on their platform. Spread of fake news can have negative and long lasting consequences. One such incident occurred during the last presidential elections where the opposing parties purposely vilified each other by spreading certain rumours on the social media. In this project, we aim to deal with a subset of this problem related to disaster handling measures.

The goal of this project is to predict whether a tweet talks about a real disaster or not. This is a very relevant challenge as people are actively using social media platforms, primarily Twitter to announce emergencies in real-time and such tweets are monitored by news agencies and disaster relief organizations. Hence, there is a need to segregate real and fake disaster tweets so that timely action can be taken to address the disaster.

1.2 Related Work

Over the past decade, there have been major strides in the task of text classification. An unsupervised feature generating model proposed by Google [3] utilizes the strength of transfer learning and multi-layer bidirectional Transformer encoder model to generate the vector embedding for words. To achieve good results on general text classification tasks, another semi-supervised Sequence Learning algorithm was proposed by Andrew et al. [2] that uses autoencoders as a first step to finetune the parameters of the model and leverages LSTM's finetuned model to classify text. Another approach proposed by Zhang et al. [6] uses Word2Vec [7] approach for obtaining text embeddings followed by a Convolutional Neural Network and a Softmax binary classifier to categorize the sentences. Unfortunately, a downside to CNN based models is that even simple ones require practitioners to specify the exact model architecture to be used and set the accompanying hyperparameters. One crucial step in order to achieve good results on text classification task is to learn vector space representation of the words. This vector space is necessary as it specifies the correlation among words and thus, generating a vector space in which these words are projected nearby. For this purpose, we have used Global Vectors for Word Representation [9] as it outperforms other word embedding techniques such as Word2Vec [7].

2 METHOD

2.1 Approach

To classify whether a tweet is related to a real disaster or not, we need to learn the representations between words in tweets and the labels given to these tweets. Since the tweets contain words, we need a way to represent these words by numbers. We apply different techniques for vectorization such as Bag of Words Count which counts the number of occurrence of each word in the tweet and TF-IDF vectorization approaches where each word receives a score based on its Term Frequency and Document Frequency. Both these approaches generate a vector of the length of the vocabulary. We also look for combination of words whose occurrence can be noted with its TF-IDF score. Using these generated vectors, we perform Monte-Carlo simulations for training on various machine learning algorithms such as Logistic Regression, Support Vector Machine, Multinomial Naive Bayes, Decision Tree and Random Forests which is discussed more in section 3.3.

The detailed illustration of our final approach is shown in Figure 1. From the machine learning experiments described in the above paragraph, we observed that we will need to incorporate the sentence structure for better performance. To exploit the sequential nature of a tweet and learn the temporal correlation among the words, we use a deep learning architecture known to perform better on textual data. Firstly, we preprocess and clean the tweet sentences to remove unnecessary information from the training and testing tweets. Secondly, we use Global Vectors for Word Representation [9] in order to project the words into a vector space of dimension 100. Thirdly, we use the vector embedding of the entire sentence and pass it through LSTM to preserve the sequential format of the data. The LSTM architecture is followed by several fully connected layers and a sigmoid activation function which provides us the probability score of the sentence which is either 1 (Real disaster tweet) or 0 (Fake disaster tweet). The above-mentioned procedure is discussed in detail in the following subsections.

2.1.1 ***Data Preprocessing.*** Raw tweets in the dataset had several discrepancies. First was the presence of tags("@") and hyperlinks in the tweets. Since a tag is used to mention people and a hyperlink to navigate people to other sources, we filter them out. Moreover, we remove punctuations and convert the tweet into lowercase as the difference in casing of the words can lead to generation of different word embeddings. We also remove all stop words, i.e. extremely common words such as the, of, for, etc. Further more, we choose lemmatization to convert all inflectional forms of a word to its base form. For example, upon applying lemmatization the word "geese" is transformed to its base "goose". Further more, twitter tweets can be filled with the set of emojis which should be removed before it is passed on to our LSTM network. We primarily use the libraries "re" and "nltk" to preprocess all tweets. Each process has been explained with respect to an original sentence as following:

1. **Original text:** Battle of the GOATS!!! https://t.co/sdjfv <a href=img.asp>
2. **Lowercase:** battle of the goats!!! https://t.co/sdjfv <a href=img.asp>
3. **Removing URL:** battle of the goats!!! <a href=img.asp>
4. **Removing HTML:** battle of goats!!!
5. **Removing Punctuations:** battle of the goats
6. **Lemmatization:** battle of the goat

2.1.2 ***Data Augmentation.*** As mentioned in the section[3.1], we have a total of 7613 tweets in the dataset. This amount of dataset is not enough for training a deep neural network because of certain issues like overfitting. To counter this issue and increase the variance in the dataset we propose below a "novel" strategy to augment data.

Our augmentation strategy involved replacing certain words in the tweet by their closest synonyms. We used GloVe and WordNet in order to replace the words in the tweets. From the original tweet, we generated two new tweets with different synonyms using both GloVe and Wordnet. Thus, after augmentation, our dataset increased 4 fold and was used to train the LSTM model.

Examples of the sentences generated by substituting synonyms are as mentioned in the below table:-

	Example 1	**Example2**
Original Tweet	I was in a horrible car accident this past Sunday	Usama bin Ladins family dead in airplane crash. Naturally no accident
Tweet1 using GloVe	I was in a sick car accident this past sunday	Usama bin Ladins kinsperson dead in airplane crash. Naturally no fortuinity
Tweet2 using Glove	I was in a horrific car accident this Sunday	Usama bin Ladins family dead out landed crash . Naturally no accident
Tweet3 using Wordnet	I was in a atrocious car accident this past Sunday	Usama bin Ladins day dead in engine crash . Naturally hay accident

Table 1: Data Augmentation

2.1.3 ***Global Vectors for Word Representation.*** After preprocessing the tweets, we need to have a vector space representation of the words present in the tweets. To project them onto a vector space, we use a word transformation technique known as Global Vectors for Word Representation [9]. This method transforms every word into a vector of length 100. The output of this network would have a dimension of N x 100 where N stands for number of words in the network. As the number of states in LSTM is predefined, we pad the output with a vector generated from zeros and convert the dimension from N x 100 to 32 x 100. To overcome the lack of training data and the issue of overfitting, we use pre-trained word vectors to avoid overfitting. A pre-trained GloVe model on Twitter dataset [8] itself containing 27B tokens and 1.2M words in its vocabulary is used. While transformation, if we come across a word in our training dataset that isn't present in the vocabulary, then we assign a specific random vector of length 100 to that unknown word. Upon this transformation, words with similar meaning are closer to each other in the generated vector space.

2.1.4 ***Long Short Term Memory (LSTM) [4].*** Subsequently, we use a LSTM network to sequentially process every word in the sentence. A simple Neural Network is not capable of processing temporal data as the network does not have any memory of previous states and the past state cannot influence the current state output. On the other side, Recurrent Neural Networks (RNN) are good at handling temporal data as it remembers the things learnt from prior inputs while generating the output. It does so by adding a feedback loop from the output to the input for the next timestamp and passing the hidden state learnt at every timestep. So, the same input could produce a variety of outputs depending on the previous inputs in the series. But, the main downside of using a RNN architecture in long sequences is the problem of vanishing gradient. As more and more layers are stacked in the network, the gradients of earlier layers in the network approaches zero which slows down the training process of the network.

In order to solve this problem of vanishing gradient, we use a modified version of RNN which is known as LSTM. Long Short Term Memory units include a 'memory cell' that can maintain information in memory for long periods of time. It is possible due to a set of gates used to control when information enters the memory, when it is output, and when it is forgotten. This architecture lets these units learn long-term dependencies.

Upon achieving the best performance with the model described in the prior paragraphs, we looked to modify this approach by replacing LSTM cells by a different recurrent cell structure called Gated Recurrent Units [1] which is a simpler cell with only 2 gates: the update gate and reset gate. Due to its simpler structure, GRU layers are fast to train and are known to perform equally good as LSTM units. Another approach involved averaging the LSTM prediction with the probability of a tweet with that keyword being real. Finally, we also implemented the state-of-the-art model BERT and tuned it to find optimal performance. We compare the results obtained from all these approaches in the section 4.2. However, the model where we achieve the best performance is the pretrained GloVe embedding + LSTM layer model.

2.2 Rationale

The first step in the pipeline was to choose the attributes to be incorporated in the model. We observed the distribution of real and fake tweets grouped by each "keyword". For obvious disaster related keywords such as terrorism, sandstorm, debris, etc., there were more real tweets than fake tweets. However for other disaster-related keywords such as avalanche, blizzard, catastrophe, etc., there are more tweets classified as fake than real. Given this observation for our training dataset, it seemed that ignoring the "keyword" attribute altogether rather than including it in text and confusing our model would yield a better performance. For the location attribute, 33% of rows have missing values and the values present are not geo-tagged and have been manually entered. Due to this, these values are not extremely specific and do have junk values as well

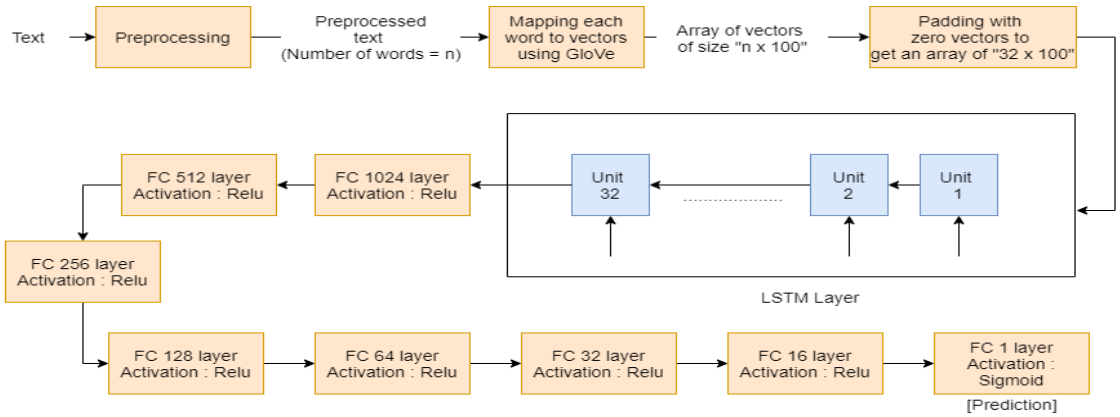

Figure 1: Final Approach.

such as *Everywhere, (21.462446,-158.022017), btwn a rock and a hard place*, etc. Coupled with the high occurrence of missing values and such junk values, we decided to ignore the "location" attribute as well.

The next step was data preprocessing. We followed the standard steps of converting the text to lowercase, removing punctuations and stopwords. For our tweet dataset, we also needed to remove emojis, hyperlinks and tags since they are not useful in learning the tweet representation. To convert the words into their base forms, we chose lemmatization over stemming as certain words such as 'coming' are simply chopped off to get 'com', a word which doesn't exist in the pretrained GloVe dictionary and the false representation of such stemmed words leads to a major drop in performance.

The next step was deciding the word embedding technique to be used to represent words in a vector space. The two popular techniques are Word2Vec or GloVe. Both the approaches are very similar in that they both try to capture the maximum semantics of analogy. We chose to proceed with the GloVe embedding approach since it has a embedding model specifically pre-trained on Twitter dataset of 2B tweets which made it more suitable to our problem as it might have better representation for some words used specifically on Twitter. Also, it is relatively simple to understand and still delivers the same performance as Word2Vec.

3 EXPERIMENT

3.1 Dataset

The dataset for this particular task comes from the competition *Real or Not? NLP with Disaster Tweets* on Kaggle [5]. The dataset has been split into training and test data CSV files. Each row of the train and test CSV file contains the following information:

- Tweet from the user
- Keyword from the tweet describing the disaster type
- Location from where the tweet was sent
- Target class (in training data only)

This information will be used by our model to determine if the disaster tweet is real or fake. There are exactly 7613 total tweets for training out of which around 4.4k are fake and 3.2k are real disaster tweets. The testing data contains another 3263 unidentified disaster tweets.

Both the training and testing data have almost same ratios of missing values for columns *keyword* and *location*. 0.8% of rows has 'keyword' attribute missing whereas the 'location' attribute has approximately 33% of rows where there are missing values.

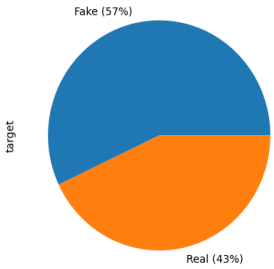

Figure 2: Class distribution of tweets

The target distribution in tweets is more or less equal as can be seen in Figure 2. Both the classes, disaster(real) and not-disaster(fake) have roughly equal amount of tweets and hence our training data doesn't suffer from the problem of class imbalance. The raw tweet text is full of stopwords and various tenses of words. We need to pre-process this text so that our machine learning model learns the proper representation. Upon preprocessing the tweets, we analyze the lengths of the Real and Fake Disaster tweets in Figure 5. We found that the average length of a real disaster tweet is 108 characters and that for the fake ones is 96 character with no tweet having more than 33 words in it.

The given training dataset also contains 7552 tweets with their corresponding keywords. We calculated the percentage of tweets that were real for each keyword. Figure 3 shows the top 5 keywords

for having highest percentage of real disaster tweets. Figure 4 shows top 5 keywords for having lowest percentage of real disaster tweets. From Figure 3 we can see that the tweets associated with keywords 'wreckage', 'derailment', 'debris', 'outbreak' and 'typhoon' have a majority of tweets that are real disaster tweets. From Figure 4 we can see that the tweets associated with keywords 'aftershock','body bags','ruin','blazing' and 'body bag' have a majority of tweets that are fake disaster tweets.

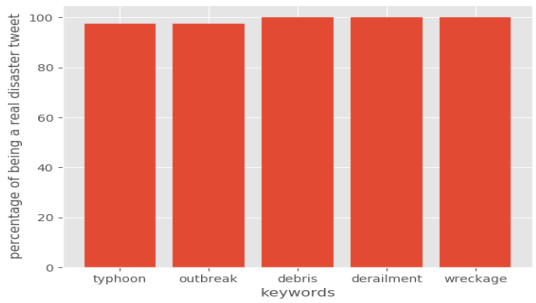

Figure 3: Keywords having maximum real disaster tweets

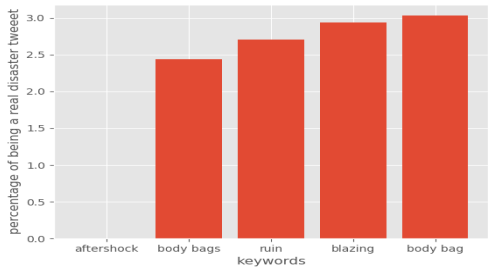

Figure 4: Keywords having least real disaster tweets

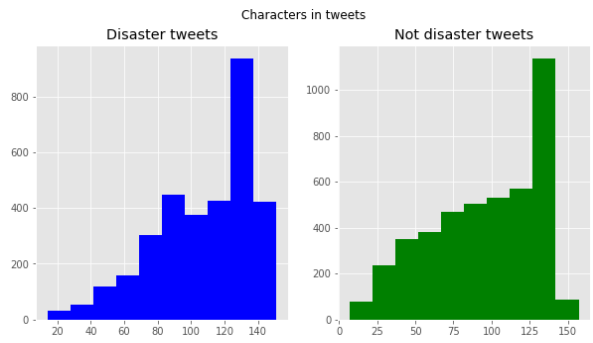

Figure 5: No. of tweets vs Character Length

3.2 Hypothesis

The primary hypothesis we have tried to investigate in this project is whether a tweet can be classified as real or fake. The other question that could be asked is given some fixed categories of disaster, find out the type of disaster the tweet is talking about so that proper safety measures can be taken by the first responders. The relevance of this problem has been already mentioned in the problem statement section.

3.3 Experimental Design

The dataset available consists of binary labels (real/fake) assigned to the tweets and hence we move forward with the hypothesis where we build a model that classifies a tweet as real or fake.

Since the available data does not incorporate the fixed categories of disaster, the primary problem would have become that of gathering tweets and annotating them. This would drastically hamper our ability to analyze the data at hand and compare different techniques and models and hence we do not consider this hypothesis for further experiments.

The basic pipeline to build a model that classifies the tweet as talking about a real disaster or not is as follows. (1) Pre-process the tweet text as described in section 2.1.1. (2) Represent the text as a vector through simple models such as Bag of words (Count/TF-IDF) or complex word vector representations such as GloVe. (3) Pass these vectors as input to a machine learning or deep learning algorithm that outputs the label (real/fake) for the tweets. (4) Evaluate various classifiers on validation data (5) Generate labels for testing data tweets.

For the Bag of Words vectorization approaches, the only difference lies in the generation of vectors of tweet sentences. Each sentence is converted to a vector of the length of vocabulary size where each word has a corresponding count or TF-IDF score. Upon generation of these vectors, we use them as input for different machine learning classifiers such as Logistic regression, Linear Support Vector Machine, Multinomial Naive Bayes, Decision trees and Random forests. The training data is split in a 80/20 stratified fashion. We perform a Monte-Carlo simulation for 5 iterations on these classifiers, average the F-1 scores and report them in section 4.1.

3.3.1 ***Using CountVec and Traditional M.L. Algorithms.*** We use simple count vectorization to represent the preprocessed text. Each tweet is represented in the form of a vector with the entry being the number of times that word appears in the tweet. These vectors are then fed to different classification models. Based on the performance table, hyperparameters were tuned for the best models. We report the validation accuracy for each method and then checked the mean F-1 score of the best performing method by uploading the predictions of test data on Kaggle. The results are shown in section 4.1 Table 2.

3.3.2 ***Using TF-IDF and Traditional M.L. Algorithms.*** We use TF-IDF vectorization to represent the preprocessed text. Each tweet is represented in the form of a vector with the entry being the TF-IDF score of that word derived from all the tweets. TF-IDF

score reflects how relevant a word is in dataset. We trained the same models with these vectors. Based on the performance table, hyperparameters were tuned for the best models. We report the validation accuracy for each method and then checked the mean F-1 score of the best performing method by uploading the predictions of test data on Kaggle. The results are shown in section 4.1 Table 3.

3.3.3 ***Using TF-IDF and N-gram modeling.*** We have also applied N-gram modeling with TF-IDF vectorizer to consider the combination of adjacent words instead of single words to utilize the positional information present among the tweet text. After trying different ranges of N for N-gram, [1,3] worked best for this data which means that the occurrences of 1-3 adjacent words are used to transform the tweets to a vector. Similar models as described in previous two sections were trained using these vectors and parameters were tuned for the models with better performance. We report the validation accuracy for each method and then checked the mean F-1 score of the best performing method by uploading the predictions of test data on Kaggle. The results are shown in section 4.1 Table 4.

3.3.4 ***Using trainable embedding layer and LSTM.*** We tried with simple vectorization approaches because they are relatively simple and provide good results for a lot of problems. In order to improve the performance on different kinds of tweets, it is evident that we need to generate vectors intelligently and also understand the long term dependencies between the words and hence, we try a LSTM based model with a trainable embedding layer. However, upon observing the train/test loss curve in Figure 6, we found that after around 3 epochs, training loss kept on decreasing whereas test loss increased dramatically, a conclusive proof of over-fitting. One possible reason for this could be the absence of sufficient data to train both the embedding layer and LSTM model. To alleviate this problem of over-fitting, we replace the trainable embedding layer with pre-trained GloVe embeddings in order to decrease the number of weights to be trained as described in our main approach.

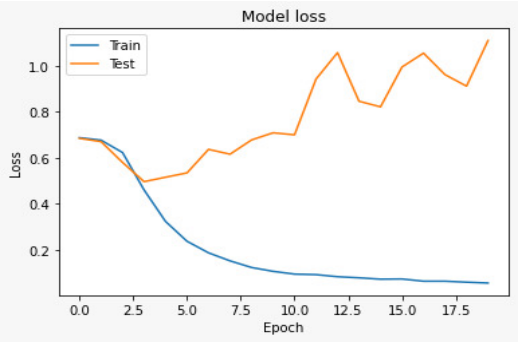

Figure 6: Overfitted Curve

3.3.5 ***Using pre-trained GloVe embeddings and LSTM.*** We divided the training data with a split of 80-20 for training and validation respectively. The text was pre-processed and augmented using the method shown in sections 2.1.1, 2.1.2. We used two pre-trained GloVe embeddings, one trained on Wikipedia text and other on Twitter's tweets. The base network for this experiment is the same as shown in Figure 1. We tuned various hyperparameters like number of epochs, batch size, learning rate and dropout rate for each FC layer. To test this method we used validation accuracy on our validation set and mean F-1 score on the test set using Kaggle.

3.3.6 ***Using GloVe embedding layer and GRU.*** We kept our base network architecture the same as that described in section 3.3.5 with one small change. We replaced the LSTM cells by another variant of Recurrent Neural Networks, called Gated Recurrent Unit (GRU) that has a similar structure but is known to be computationally more efficient and achieve a performance on par with that of LSTMs. We did hyper-parameter tuning for parameters such as no. of units in GRU, batch size, etc. To test this method we used mean F-1 score on test data by using Kaggle.

3.3.7 ***Weighted Probability Scores.*** We tried taking into account the percentage of tweets that were real disaster tweets for each keyword calulated from the training data as seen in section 3.1. Out of the total 3264 test tweets, 3238 tweets had a corresponding keyword. We calculated the probability of being a real disaster tweet for the test data using the LSTM network with pre-trained GloVe embeddings. Then for each tweet that had an associated keyword, we took the average of the probability calculated by the LSTM network and the probability obtained by dividing the percentage of tweets that were real for the keyword by 100. To test this method we used mean F-1 score on test data obtained using Kaggle. The results are shown in section 4.2.

3.3.8 ***BERT.*** Bidirectional Encoder Representation of Transformers (BERT) is a self-supervised technique for NLP tasks. The model is usually pre-trained on large datasets like Wiki and then finetuned for various different tasks in natural language processing. It differs from the traditional sequential model, because instead of reading the text in a certain manner, it reads the entire text at once. In order to classify the tweets as real or fake, we have added a classification layer on top of the pre-trained BERT architecture.

Figure 7: BERT Loss curve

From the figure above, we can see that the validation loss keeps on increasing just after an epoch while training loss is decreasing considerably which is a sign of overfitting.

4 RESULTS

4.1 Results

We have been provided with training and test data on Kaggle [5]. We don't use the provided testing data for anything other than generating the final predictions.

4.1.1 ***Results on Traditional M.L. algorithms.*** Table 2 shows the summary of model performances using CountVec. It can be seen that Multinomial Naive Bayes and Logistic Regression perform similarly. We tune their hyper-parameters using Grid Search and receive a best mean F-1 score of 0.75 on the Kaggle challenge for Multinomial Naive Bayes classifier with a Laplace smoothing parameter of 1.

Index	Algorithm	Validation data Accuracy
1	Logistic Regression	0.75
2	Linear SVM	0.73
3	Multinomial Naive Bayes	0.7597
4	Decision Tree	0.703
5	Random Forest	0.701

Table 2: Validation Accuracy for CountVec + ML algorithms

Table 3 summarizes performance of different models using TF-IDF vectorizer. It can be seen that Multinomial Naive Bayes and Linear SVM perform similarly. We tune their hyper-parameters using Grid Search and receive a best mean F-1 score of 0.79 on the Kaggle challenge for Multinomial Naive Bayes classifier with a Laplace smoothing parameter of 1.

Index	Algorithm	Validation data Accuracy
1	Logistic Regression	0.7396
2	Linear SVM	0.7442
3	Multinomial Naive Bayes	0.7449
4	Decision Tree	0.6936
5	Random Forest	0.6967

Table 3: Validation Accuracy for TFIDF + ML algorithms

Summary of model performances using TF-IDF with N-gram modeling is shown in the table 4. It is clear that there is no significant improvement on the performance from the simple TF-IDF approach. Multinomial Naive Bayes and Logistic Regression perform similarly. We tune their hyper-parameters using Grid Search and receive a best mean F-1 score of 0.78 on the Kaggle challenge for Logistic Regression classifier with a regularization parameter of 10.

4.1.2 ***Results on Deep learning algorithms.*** We have implemented various different deep learning architectures to evaluate the performance on the validation dataset. We developed a arhictecture with pre-trained GloVe embeddings followed by LSTM cell layer and certain fully connected layers for fine-tuning. The results of this approach are specified in Table 5.

Followed by this, we simply replaced the LSTM cells with another recurrent cell called the GRU and kept the rest of the architecture same. This approach did not perform well for this task and ended up getting a validation accuracy of 77.2% and a mean F1 score of just 0.6732.

Index	Algorithm	Validation data Accuracy
1	Logistic Regression	0.7376
2	Linear SVM	0.7254
3	Multinomial Naive Bayes	0.7342
4	Decision Tree	0.6780
5	Random Forest	0.6311

Table 4: Validation Accuracy for TFIDF + N-Gram

We leverage the power of ensembling by combining the GloVe + LSTM model with the probability scores derived from keywords present in the dataset. Using weighted probabilities, the following changes were observed for predictions on the test dataset:

(1) The number of tweets whose class changed from being real disaster tweet to fake disaster tweet : 178
(2) The number of tweets whose class changed from being fake disaster tweet to real disaster tweet : 63

The mean F-1 score obtained on test data was '0.830' which was not an improvement from just using the LSTM with pre-trained GloVe embeddings.

We also use a pre-trained BERT model to more accurately learn the semantic structure of tweets followed by a classification layer that outputs the probability of tweet being that of a real disaster. The result has been tabulated in Table 5.

Results of the above mentioned techniques have been mentioned in the table below. By observing these values, we see that the GloVe (Twitter) + LSTM network outperformed all other approaches.

Architecture	Test F1-score
GloVe (Twitter dataset) +LSTM	***0.832***
GloVe (Wiki dataset) + LSTM	0.817
Weighted probability scores + GloVe + LSTM	0.830
BERT + 2FC layers	0.826
GloVe (Twitter dataset) + GRU	0.6732

Table 5: Results

The following hyperparamters were used in order to obtained the results:-

(1) Batch Size = 64
(2) No. of Epochs = 25
(3) Dropout Rate for F.C. layers = 0.2
(4) Learning Rate = 0.01

(5) Optimizer = Adam
(6) Loss Function = Binary Crossentropy

Moreover, the training accuracy and training loss curves are displayed in Figure 8.

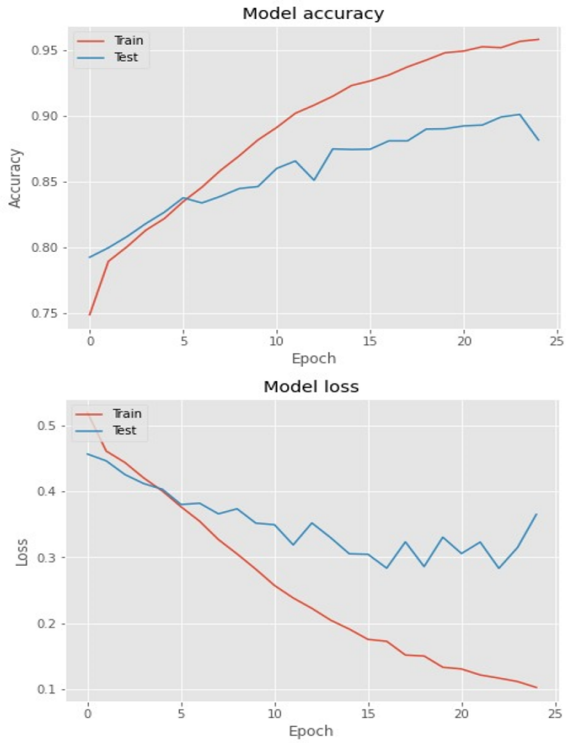

Figure 8: Final model Loss and Accuracy curves

Since we did not have access to the labels of testing dataset, we have used the validation set to obtain the classification report of our final GloVe (twitter) + LSTM model. The classification report looks like:

	precision	recall	f1-score	support
0	0.85	0.95	0.90	2623
1	0.93	0.77	0.84	1945
accuracy			0.88	4568
macro avg	0.89	0.86	0.67	4568
weighted avg	0.78	0.77	0.77	4568

Table 6: Classification Report :-LSTM(twitter)+GloVe

4.2 Discussion

Based on the results described in the earlier section, the key aspects of our discussion can be divided into 3 broad questions.

4.2.1 *Why are the traditional machine learning algorithms not capable of achieving good results on this dataset?*

Traditional ML algorithms work where the input is a vector representing a sentencce. Methods like n-gram might be able to encode partial sequential information in the sentence. But complete sequential information still gets lost in these vector representations. To accurately learn the nature of the sentence, algorithms need to capture the full sequential information in the sentence. Since traditional ML algorithms are not able to do so, they do not achieve a high mean F-1 score on the test data.

4.2.2 *Why does the proposed approach work better with Twitter embeddings than with Wiki embeddings?*

The structure of language and words used in the given tweets will align more strongly with GloVe embeddings from Twitter data due to the similarity between the language style and certain Twitter-specific jargon used like the abundance of acronyms. The language style in Wikipedia GloVe embeddings may be more formal and professional in style.

4.2.3 *Why does a lesser complex model (LSTM) work better than a much more complex model (BERT)?.*

The final approach (GloVe + LSTM) proposed by us had an overall of 1M trainable paramters as compared to 335M trainable parameters present in the BERT architecture. From Table 5, we note that with proper hyper-parameter tuning, the GloVe + LSTM approach performs a little bit better than BERT. An explanation for this result could be the extremely small size of the training dataset given. A simpler architecture is able to fit the data properly whereas it is evident, from Figure 7, that a complex architecture such as BERT overfits the dataset in the initial stage of training itself as validation loss keeps on increasing after that point.

5 CONCLUSIONS

- We learnt that exploring and analyzing data can help to weed out unnecessary information. For our dataset, we saw that *keyword* attribute and *location* attribute had a lot of missing values with *location* containing a lot of junk values. As a result, we were able to remove those attributes.
- We also saw that the pre-processing step varies with the type of dataset. Majority of the resources that dealt with text pre-processing did not include removal of links and tags ("@"). However, the removal of these tokens was important to the dataset as they served no purpose in classifying tweets as real or fake.
- A complex model or a state of the art model does not always guarantee the best performance. A complex model like BERT could not outperform a simpler tuned architecture of LSTM probably because of lack of training data.
- One thing that we observed was that a lot of these feature representation techniques and complex architectures have already been pre-trained on gigantic datasets by the creators and can directly be used with some fine tuning for custom datasets with lesser number of training data points.

There were certain experiments that could have lead to improvement in performance had they been tested. We would have liked to replace the acronyms in tweets such as IMHO, fab, etc. with the actual full form so that model might benefit by learning the actual tone of the sentence. We also hoped to test these models by providing a weighted loss function which enforces a larger penalty when

the positive class (real disaster tweets) are wrongly classified. We also came across this concept of attention that basically provides weights to different words in the sentence in order to better learn the true semantic structure of the tweet.

6 MEETING ATTENDANCE

Date	Time	Members
April 4^{th}	1:00pm - 2:30pm	cagandhi, uvasani, kshah9, sshah29
April 10^{th}	12:00pm - 2:30pm	cagandhi, uvasani, kshah9, sshah29
April 16^{th}	5:00pm - 6:30pm	cagandhi, uvasani, kshah9, sshah29
April 21^{st}	5:00pm - 6:30pm	cagandhi, uvasani, kshah9, sshah29

Table 7: Group Meetings

REFERENCES

[1] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[2] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*. 3079–3087.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[5] Kaggle. Ongoing. *Real or Not? NLP with Disaster Tweets. Predict which Tweets are about real disasters and which ones are not.* https://www.kaggle.com/c/nlp-getting-started/

[6] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[8] Stanford NLP. 2020. GloVe. https://github.com/stanfordnlp/GloVe

[9] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

GITHUB LINK

https://github.com/kenil-shah/CSC522-FinalProject