AI for Disaster Tweet Classification

ECSE 526 Final Project

VRAJ PATEL
(261022581)
vraj.patel@mail.mcgill.ca

ROMAIN BAZIN
(261087142)
romain.bazin@mail.mcgill.ca

***Abstract*—This report describes the results of an AI which is useful to classify disaster-related tweets with the help of Natural Language Processing (NLP). The project was conducted to evaluate the feasibility as well as effectiveness of using AI for disaster response, with the aim of not only increasing speed but also an accuracy of disaster-related information dissemination. The AI was trained on a dataset of tweets labelled as either disaster-related or not disaster-related, with hyper parameter tuning on a validation dataset. The performance of the trained AI was evaluated on a separate dataset of tweets.**

I. Introduction

Disasters such as earthquakes, tsunamis, wildfires, flood, hurricanes, can cause significant loss of many lives. It is limited not only to lives, due to disasters the government has to bear billion of dollars in economic loss. A survey shows, in 2021, due to calamities nations has to bear more than 300 billions of dollars in loss. Authorities around the globe spend large chunk of capital to mitigate these events especially in technology. Effective disaster response depends on timely and accurate information dissemination, which can be challenging during those time since, it is mostly filled with chaos and confusion. Social media platforms, such as Twitter which has more than 200 million active users, is widely used by individuals to share information and updates about disasters. Given the fact that, there are more than 500 million Tweets on a daily basis, finding right information in the emergency is an important and time-consuming task. In this project, we explore the feasibility and effectiveness of using AI to automatically classify disaster-related tweets.

For the classification task, we explore two NLP models which are **BERT** (Bidirectional Encoder Representations from Transformers) and **LSTM** (Long Short Term Memory). We used ensemble learning to improve the overall model performance on a classification task. BERT allows us to understand the context and meaning of words in a sentence, which is important for accurately classifying disaster-related tweets. Meanwhile, LSTM is better for time-series data which in the end help model to comprehend better, which is crucial for identifying the type of disaster being discussed. Ensemble learning allows us to combine the predictions of multiple models, which can improve the overall performance of the classification. Combining BERT and LSTM for ensemble learning gives a powerful toolkit for accurately and efficiently classifying disaster-related tweets.

This project was also part of a Kaggle challenge on **disaster tweet classification**. The Kaggle competition provided a good source of data for training and evaluating for our model, it give us a way to measure the performance of our model on a test dataset. The competition also provided a leaderboard where we could compare our model's performance with other participants in the challenge.

The code is available here.

II. Data

A. *Train, validation and test sets*

The train dataset consisted of 7613 labelled tweets, we split the dataset into training and validation sets. In order to ensure that our model achieves more generalization we used stratified split to ensure that distribution of the labels was preserved in both training as well as validation sets. Our models BERT and LSTM were trained on training set, validation set was used to evaluate the performance of the models and fine-tune the hyper-parameters. Splitting the dataset into training and validation sets allowed us to train our models on a large portion of the data (80% of labelled tweets) and evaluate their performance on a held-out set (20% of the labelled tweets), which is a common practice in machine learning during training.

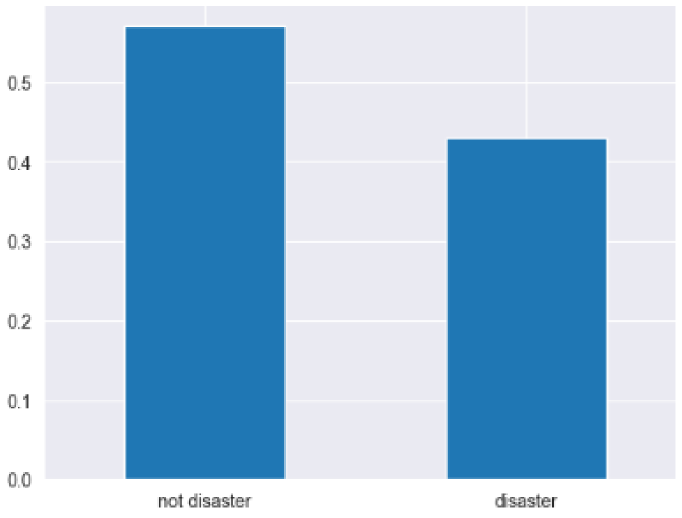


Fig. 1. Distribution of labels for full train dataset

The test set consisted of 3263 unlabelled tweets. We used the trained BERT and LSTM models to classify those tweets and submitted that on Kaggle platform. The platform then evaluated the predictions which were made by our model and give us the results. As a result, Kaggle gave us an accuracy score and a leader-board rank so that we can compare our results with other participants. The accuracy score provided a measure of the overall performance of our model on unseen data.

$$Accuracy = \frac{Corrrect\ predictions}{Total\ number\ of\ samples} \tag{1}$$

III. Data preprocessing

A. *Data-cleaning*

For Deep-Neural Networks to perform better at generalization, we need to train it on a clean data so that we can achieve higher accuracy and lower loss. Preprocessing step helps to clean and normalize the tweets, making them easier to analyse and understand for natural language preprocessing(NLP) models.

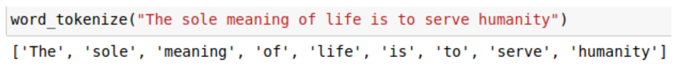
Preprocessing part of the tweets can improve the performance of language models like BERT and LSTM in several ways. Firstly, preprocessing helps to make the data more consistent and easier to work with, which can make model more stable and reliable. Secondly, preprocessing helps to reduce the amount of noise and irrelevant information in the tweets, which can improve accuracy, precision, recall and F1 score of the model. Lastly, preprocessing can help to standardize the tweets, which can make it easier for the language models to learn from the data. For the given reasons, we implemented a preprocessing pipeline with the following transformation:

- Remove html escapes like **&amp**
- Remove html tags
- Substitute markdown urls by their inner text
- Remove text in brackets [ ], { }
- Remove standalone sequences of special characters like **& #**
- Remove sequence of white spaces
- Remove # in #hastags but keep the word
- Remove email addresses
- Remove phone numbers
- Remove dates
- Remove timestamps
- Remove accents
- Remove user mentions **@user**
- Remove currency symbols **$**
- Remove emojis
- Lowercase the whole Tweet

At first, we implemented these transformation with an automated tool from the library tweet-preprocessor. However, the tool wasn't able to do all these transformation out of the box, and even the basic functionality of removing the urls wasn't working all the time. So we decided to implement these transformations ourselves using a combinations of regular expressions and automated tool with the libraries re and textacy.

B. *Tokenization*

Deep-Learning models can only take numbers as an input, as a result, we need to convert texts into numbers so that we can use that for training and testing of the model. Tokenizer is in charge of preparing the inputs for a model, it converts string of input data into list of numbers so that it can used. For this task, we used a pretrained tokenizer which was trained specifically on Tweets and it has 30000 words in its vocabulary. This model is based on BPE (Byte-Pair Encoding) tokenization algorithm, which tokenizes strings based on information theory and compression. It uses Huffman encoding for tokenization meaning, it makes dictionary of more frequent words and whenever less frequent words appear in the string it break that into smaller parts which are made of more frequent words. In addition to this, it also add some special tokens so that the model can understand start and end of sentence.



```
word_tokenize("The sole meaning of life is to serve humanity")
['The', 'sole', 'meaning', 'of', 'life', 'is', 'to', 'serve', 'humanity']
```

Fig. 2. Tokenization of a sentence

C. *Embedding*

Embedding layer is a very important integral part of a Natural Language Processing model, Embedding is svectorization of tokens. It puts the tokens in a higher dimension space, where distance between two vectors illustrates the relation between different token.

For our project, we used pre-trained BERT in which first layer of the model is Embedding layer, this model is then hyper tuned as per our specific task.

For LSTM, we used TensorFlow's embedding layer as a first layer of our model. All the weights for this layer were initailized as zero and later we trained and tuned our model on train as well as validation dataset respectively.

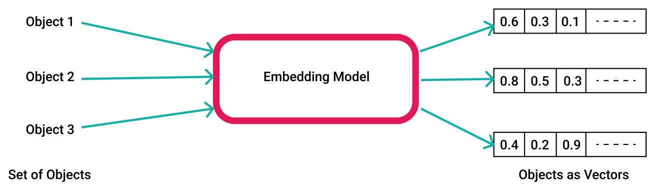


Fig. 3. Embedding layer

IV. Feature Extractor

Embedding layer converts words into vectors, as a result there are vectors for every words in the given dataset, which can passed to our feature extractor. In our project we've used two architectures for feature extraction first one isBERT and another is LSTM.

- **BERT:**
  We've used pre-trained $BERT_{base}$ which has 12 encoders with 12 bidirectional self-attention heads consequently, it has more than 110 million parameters. We've tuned this model specifically for our dataset. From the

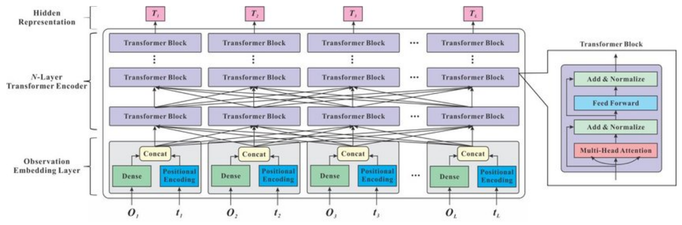


Fig. 4. Architecture of BERT

  picture, there are 3 parts of BERT model. First one is embedding layer which we discussed previously, second is **Transformer Encoder**, and the last one is **Transformer decoder**.
- **LSTM:**
  It is a type of Recurrent Neural Network (RNN) which overcomes the problem of diminishing gradients by having long term memory, this memory is useful for time-series data and understanding the sentences more deeply. For our classification problem we've used simple LSTM architecture with more than 5 million parameters to learn. We've trained and tuned our model on trained dataset.

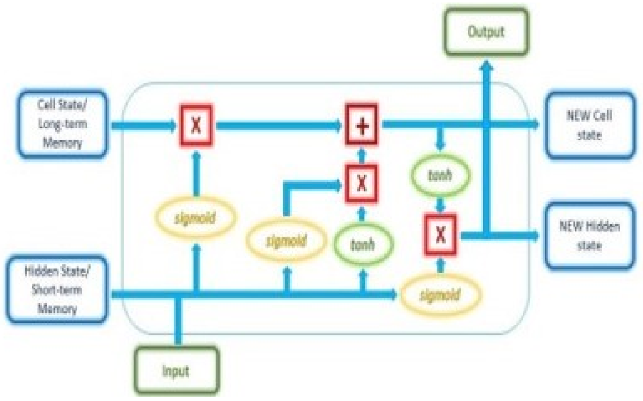


Fig. 5. Architecture of LSTM

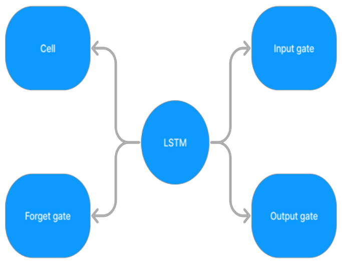


Fig. 6. Components of LSTM

  This photo illustrates the architecture of LSTM, there are four components which makes LSTM architecture. Cell is the most important part of the LSTM, it is useful for context formation of sentences. The other three gates are important for flow of information inside the model.

V. Classifier

The last layer for our models is classifier, it gives probabilities as an output for a given input belonging to a particular class. We've used linear as well as Sigmoid for the classification of our tweeter dataset.

- Linear classifier:

$$y(x) = Wx + C \tag{2}$$

- Sigmoid classifier:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

VI. Results

A. Result Metrics

- Precision: It shows what proportion of positive identification was actually correct.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{4}$$

- Recall: It gives information about how many actual positives was identified correctly.

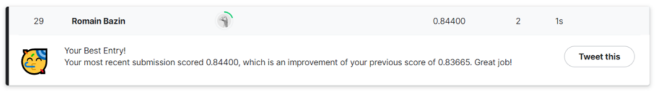
$$Recall = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{5}$$

- F1 score:

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

B. BERT

We tuned pre-trained BERT model for our specific task which was classification of tweets, as a result we got around 85.1% accuracy with 88.9% , 73.8% , 80.6% as a precision, recall and F1-score respectively on validation set.
We also uploaded our results for test-set on Kaggle which helped us better to understand the performance of our model compared to others. From the figure, we got the accuracy of


29 Romain Bazin 0.84400 2 1s

Your Best Entry!
Your most recent submission scored 0.84400, which is an improvement of your previous score of 0.83665. Great job!

Fig. 7. BERT results

84.4% for test set with a rank of 29 out of more than 800 people.

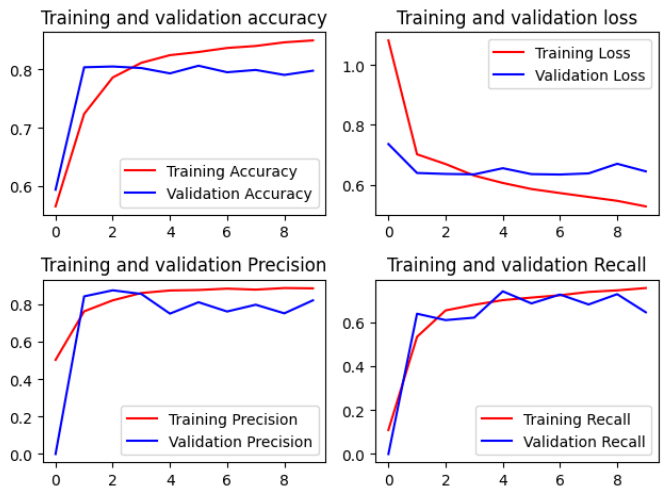


Fig. 8. Tuning of LSTM with best hyper-parameters

```
514   thevp_007   0.78792   1   1m
```

Fig. 9. LSTM results

C. *LSTM*

We trained our LSTM model from scratch with more than 5 million parameters to learn, compared to BERT the training of LSTM was less costly in terms of computing power. As a result, we got accuracy of 80.43% with 80.92%, 67.85%, 73.81% as a precision, recall and F1-score respectively on validation set.

We uploaded our output from our model which was trained on training set as well as tuned on validation set. As a result, we got accuracy of 78.792% on test set with a rank of 514 out of more than 800 people.

D. *Ensemble Learning*

The accuracy of machine learning models can be increased with the help of bagging, a powerful, yet simple ensemble learning technique. It functions by using distinct subsets of the training data to train several models, and then average the individual models' predictions to arrive at the final forecast. This method can help to decrease over fitting in individual models while simultaneously improving forecast stability. Additionally, since each model is trained on a different subset of the data, it is less probable that any biases in the training data will have an impact on it.

We settled for this method because it was simple to put in place but other methods could have been used like training a meta model.

VII. Hyper-parameters optimization

Tuning hyperparameters is a critical stage for any Machine Learning model. It entails modifying the values of a model's hyperparameters to improve its performance on a particular dataset. This is significant because a machine learning model's performance is strongly influenced by the values of its hyperparameters. The model may perform poorly on new, unforeseen data if the hyperparameters are not properly configured, making it unable to learn from the training data. Tuning hyperparameters can assist enhance a model's accuracy, and stability.

We optimized the commonly most important hyper-parameters, which are the *learning rate*, the *number of epochs* and the *batch size* using two libraries :

- **Optuna** (Optimization Framework) : it's capable of pruning non promising runs very early by using Bayesian inference.
- **Wandb** (Visualization framework) : we used it to visualize the impacts of each hyper-parameter on the validation loss and accuracy.

We ran for each model one trial of 100 runs, where each run has a different set of hyper-parameters. What came out of this is that one epoch is definitely the most important parameter, followed by the learning rate (figure 11 and figure 10)

The overall goal of the optimization was to reduce the validation loss. With this process, we found the best parameters summarized in table I.

Model	learning rate	epoch	warm up steps	Batch size
BERT	3.588e-5	1	8	32
LSTM	1e-3	10	5	32

TABLE I
BERT HYPER-PARAMETERS

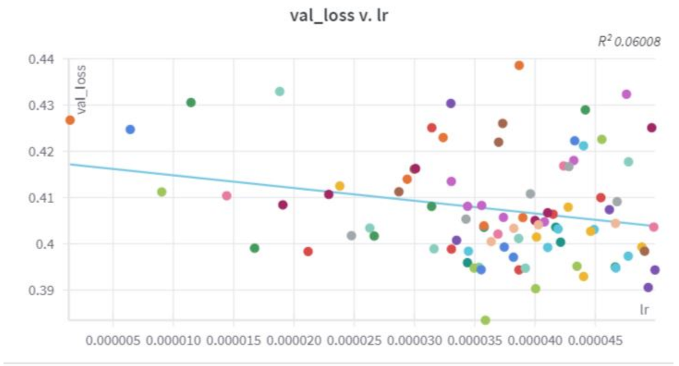


Fig. 10. Validation loss against learning rate for 100 runs

VIII. Results

In table VIII you can find the accuracy score and the leaderboard rank of the different models on the test set.

The best performing model is definitely BERT fine-tuned using the hyper-parameters from the hyper-parameters search. The LSTM network doesn't perform as good but this was expected as BERT is much better at understanding the full context of a sentence.

One hope we had was that the ensemble learning model would perform better than the two other models by taking

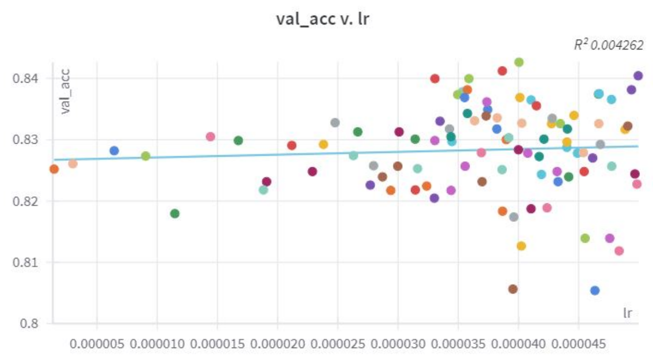


Fig. 11. Validation accuracy against learning rate for 100 runs

	Accuracy score	Leader-board rank (out of 869)
BERT	0.8412	29
BERT*	0.8440	27
LSTM	0.7879	500
Ensemble	0.80	312

TABLE II
RESULTS OF DIFFERENT MODELS ON TEST SET. BERT* MEANS THE MODEL WAS FINE-TUNED USING THE BEST HYPER-PARAMETERS FOUND.

advantage of the strong points of each model. However, we observe that it was not the case at all, the ensemble model is strongly impacted by the lower performance of the LSTM network, this is due to the fact our ensemble model is a simple average of the predictions of the other models. We tried to artificially change the importance of each model in the ensemble model but we found the optimal proportion to be to never use LSTM. Nonetheless, it's important to keep in mind that ensemble models shine when they combine a lot of other models. Here, by using only two models, we don't benefit much from the ensemble's capabilities which usually allow for better generalization.

IX. CONCLUSION

The goal of this project was to classify disaster tweets from Kaggle's challenge dataset using different NLP architectures, After exploring different models, turns out that BERT performs well compared to LSTM since it was trained on which represents a very diverse dataset. In addition to this, BERT has a wider range of applications such as sentence completion, question-answering, grammar checking, etc. However, LSTM in comparison is a simple Bi-directional neural network, with only 6 million parameters compared to BERT's 110 million parameters. To improve the performance of our classification task, we also explored the path of ensemble learning however, we didn't get any satisfactory results from that. While doing this project we came to know that, the most important thing for any NLP task is to pre-process data properly, cleaner data leads to higher accuracy. We also gave a great deal of efforts towards optimizing the hyper-parameters of our models. With all our efforts combined, we were able to rank 27 out of 871 (top 3%) in the Kaggle competition.

REFERENCES

[1] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[2] Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling." Thirteenth annual conference of the international speech communication association. 2012.