Natural Language Processing of Disaster Tweets

Gita Rayung Andadari
gitarayung.andadari@studenti.unipd.it

Endrit Sveçla
endrit.svecla@studenti.unipd.it

1. Introduction

Twitter is a social media platform that allows users to send and receive information in real time. In the past several years, Twitter has shown its great potential to spread awareness quickly when a natural disaster occurred, and this is done through its *Tweet* (message sent in Twitter). Because of that, the needed help can be obtained in a shorter period of time. However, with the vast amount of tweets being exchanged daily, it is highly important to be able to classify which tweets are actually reporting about natural disaster events. Therefore, in this project, our team is building a **machine learning model that can classify which Tweets indicate true natural disasters events with greater than 77% accuracy.**

To determine the best model, we will measure the f1 score on the test set from each given model. The f1 score and computational cost trade off is also then being considered to determine the most optimal model. With only 22 variables input, the **Neural Network** model can produce 77.60% accuracy on the test.

2. Dataset

In this project, we are working with 10.876 tweets provided by the company "Figure-eight" and originally shared on their 'Data For Everyone' website. There are 2 datasets :

- 1 training set consist of 7.613 tweets
- 1 test set consists of 3.263 tweets.

Each dataset contained information about the followings :

- id - a unique identifier for each tweet
- text - the text of the tweet
- location - the location the tweet was sent from (may be blank)
- keyword - a particular keyword from the tweet (may be blank)
- target - in training set only, this denotes whether a tweet is about a real disaster (1) or not (0).

Based on our analysis, the location feature does not have a direct connection with a tweet being considered as a natural disaster or not. Therefore, our team will not consider this during the learning process. In addition, to give more emphasis on impactful words, the keyword is added on the beginning of every natural disaster tweet.

To understand the data further, distribution of the training and test set is observed. Both sets approximately share the same distribution based on the length of the tweet. Therefore, it is important to maintain this shape when splitting the training data into training and validation sets. Furthermore, the training set is observed based on the target classification. There are 57% target "1" and 43% target "0" tweets. This shows that our dataset is unbalanced which can make our model more sensitive to target 1. In general, it is also observed that natural disaster tweets tend to have shorter length of tweets.

Diving deeper, there are 1056 duplicated tweets in the training data. Those duplicated tweets consist of 325 unique tweets. Some of them have identical text but different embedded links. To avoid confusion in the model training phase, these duplicated tweets will be deleted from the training data.

3. Method

The natural disaster tweet detection is achieved by applying the text processing framework.

3.1. Data Cleaning

- Text Cleaning

Remove non-alphabetic character (ex: number, emoji, and punctuation), lowering the text, remove http link

- Remove duplicated tweets in the training set
- Fill null values of the features keyword with empty string
- Adding keyword at the beginning of the text for disaster tweet
- Removing stopwords

Stopwords are a set of commonly used words such as determiners (e.x: the, a, an), coordinating conjugations (e.x: for, an, nor) and prepositions (in, on, under). The removal step is to make the learning more focused on important keywords rather than commonly used one.

- Lemmatizing

Lemmatizing is part of Tokenization, which is a step to convert the text to a sequence of word/tokens. In this project, we used **WordNetLemmatizer**. This process is useful to reduce the amount of unique words in the data set.

- Contracting tweet

This step is needed to deal with combined words without spaces and popular abbreviations.

3.2. Vectorization of the Training and Test Data

Vectorization is a way to transform the character sets into numerical. This process is important because the targeted machine learning model only accepts numeric input. We are using a **bag-of-words** model to simplify the word representation of each tweet. Therefore, each tweet is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

In this project, we are going to experiment with 3 different vectorizers : **CountVectorizer, TF-IDF,** and **Tokenizer**. There are 2 types of experiments that will be conducted for **CountVectorizer**:

1. To find a good combination of **n-gram**
2. To find the minimum number of **nword** parameter in vectorizer. This experiment is important to minimize the computational cost while maintaining the accuracy of the model.

Each experiment will be conducted on 5 machine learning models: Perceptron, LogisticRegression (LR), DecissionTree, RandomForest, and SVC. Next, for **Tf-Idf**, the vectorizer result will become an input for the same 5 machine learning models. In the end, we will compare the end result to determine which vectorizer will be used for the final prediction.

3.3. Hyper Parameter Tuning

Using the optimum setup combination obtained from the previous experiment, we will conduct hyper parameter tuning for each machine learning model using **GridSearch**. Furthermore, to minimize the sampling bias error, we applied **StratifiedKFold** to the training set.

Finally, the tuned model then will be used to predict the test set. To decide the best model for this case, f1 score against the test set from each model will be obtained and compared. Model with the highest f1 score will be determined as the best model. F1 is calculated as follows:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Where:

$$precision = \frac{TP}{TP+FP} \qquad recall = \frac{TP}{TP+FN}$$

and:

True Positive [TP] = prediction is 1, and the ground truth is also 1

False Positive [FP] = prediction is 1, and the ground truth is 0

False Negative [FN] = prediction is 0, and the ground truth is 1

3.4. Neural Network (NN)

The model that we have been experimenting takes quite a big number of inputs (>2000 variables). Beside being computationally expensive, it also takes time. Therefore, we would like to find an alternative model that can be trained more effectively.

For this approach, we are using **Tokenizer** to tokenize the dataset. Since each tweet will have a different Tokenizer result length, we will pad it with 0 at the end for shorter tweets.

Figure 1 shows the NN layer architecture. We also applied an early stopping mechanism **ReduceLROnPlateau** which will reduce learning rate when a metric has stopped improving.

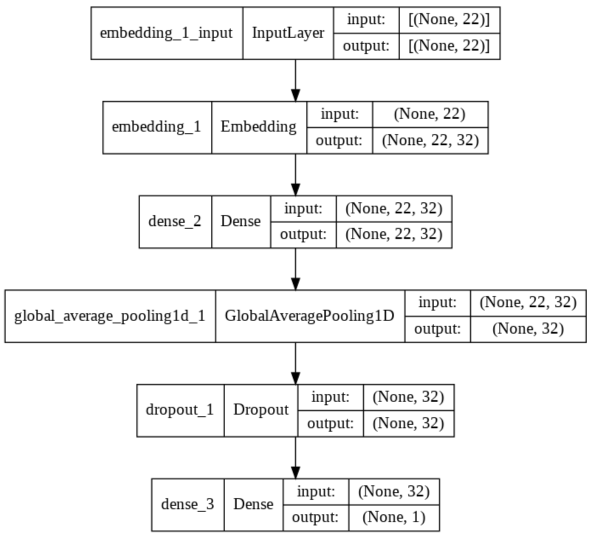

embedding_1_input | InputLayer | input: [(None, 22)] | output: [(None, 22)]
embedding_1 | Embedding | input: (None, 22) | output: (None, 22, 32)
dense_2 | Dense | input: (None, 22, 32) | output: (None, 22, 32)
global_average_pooling1d_1 | GlobalAveragePooling1D | input: (None, 22, 32) | output: (None, 32)
dropout_1 | Dropout | input: (None, 32) | output: (None, 32)
dense_3 | Dense | input: (None, 32) | output: (None, 1)

Figure 1. NN layer architecture

4. Experiments

4.1. Finding a good combination of n-gram for CountVectorizer

Experiment result is summarized in Figure 2. It can be seen that n-gram = 1 works best for all the machine learning models.

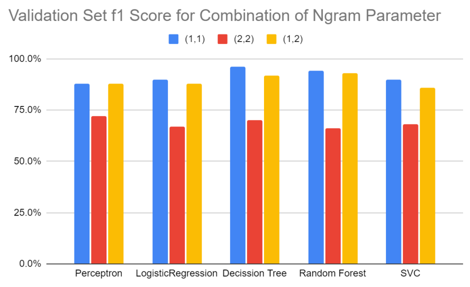


Figure 2. Validation Set f1 Score for Combination of Ngram Parameter

4.2. Finding the minimum number of nword parameter in CountVectorizer

For this experiment, we conducted multiple training sessions for each model with nword variation ranging from 250 - 12000. Experiment summary is shown in Table 1 below. It can be inferred that well trained models don't need all unique words in the training data set.

4.3. Tf-Idf

Tf-Idf vectorizer is applied then the result is used to train several machine learning models. The result is summarized in Table 1. After experimenting with both methods, for natural disaster tweet problems CountVectorizer provides better transformation based on the accuracy score yielded for the validation score. This can happen because Tf-Idf weighs down common words that often appear in the document. Therefore, important keywords may be marked with smaller scores which lead to misinterpretation by our model.

Model	#Word	F1 Val Accuracy	
		Count Vectorizer	TfIdf
Perceptron	1000	75.32%	73.25%
LR	7000	79.74%	74.58%
DecisionTree	9000	73.57%	73.25%
RandomForest	2000	78.03%	77.24%
SVC	3000	79.70%	75.41%

Table 1. F1 Accuracy for Combination of Unique Word Number

4.4. Hyper Parameter Tuning

After obtaining the optimum parameter for the vectorizer, now we are going to tune the model hyper parameters. Best parameter chosen based on the maximum validation f1 score which is summarized in Table 2. These hyper parameters then will be used to do the final prediction on the test set.

Model	Parameter
Perceptron	{'alpha': 0.01, 'max_iter': 5}
LR	{'C': 0.1, 'fit_intercept': True}
Decision Tree	{'criterion': 'gini', 'max_depth': 16}
Random Forest	{'bootstrap': True, 'max_depth': 32}
SVC	{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}

Tabel 2. Best parameter for each model

4.5. Neural Network

The goal of the neural network approach in this project is to minimize the number of input while maintaining the f1 score. With only 22 variables, this model can obtain 96.61% accuracy on the training set, 84.76% on the validation set, and **77.60%** on the test set. Figure 3 shows how the model improves over epoch for the training set, but not so much in the validation set. Thus, the early stopping mechanism stops the execution to avoid overfitting.

4.6. Analyzing Misclassified Tweets

To improve the model in the future, we have to find several possibilities why misclassification happened. Figure 4 shows the confusion matrix of the NN model on the validation set. As predicted, our model is more

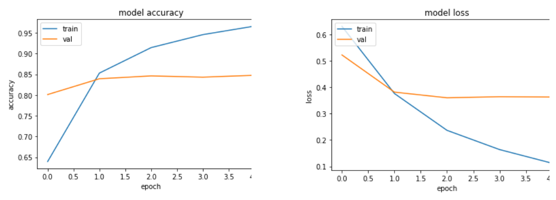


Figure 3 [left] Accuracy over epoch. [right] Loss function over epoch

likely to mark a tweet as a natural disaster. This is due to an unbalanced dataset. To overcome this issue, oversampling with SMOTE can be used.

Moreover, the quality of the labels in the training set needs to be improved. There are some tweets that do not imply a natural disaster occurrence but are labeled as so. For example, “@camilacabello97 Internally and externally screaming screaming” is marked as a natural disaster.

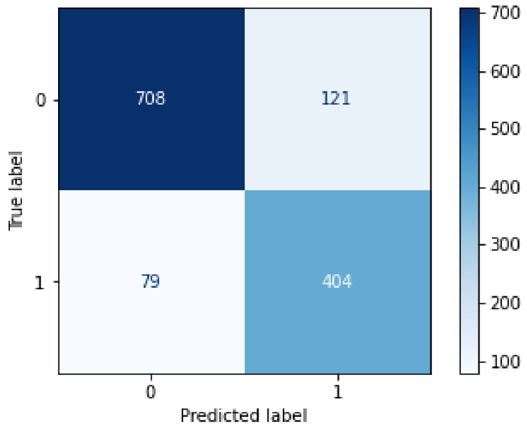


Figure 4. Confusion Matrix of NN Model on the Validation Set

5. Conclusion

Figure 5 shows the test set f1 score from all the models that we have experimented in this project. **SVC** model works best for this case with **78.73%** accuracy on the test set. Furthermore, if we have to consider the accuracy and computational cost trade-off, the **Neural Network** model is ahead of this category. With only 22 variables input, the result is only 1.13% below the best model accuracy (**SVC**) that receives 3000 variables input.

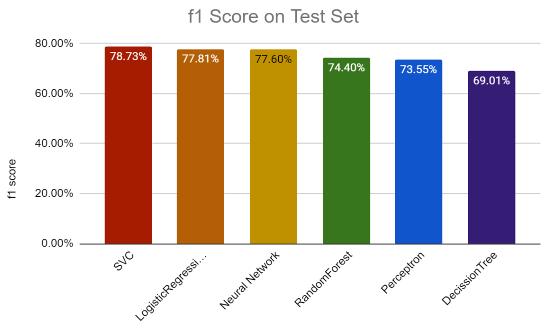
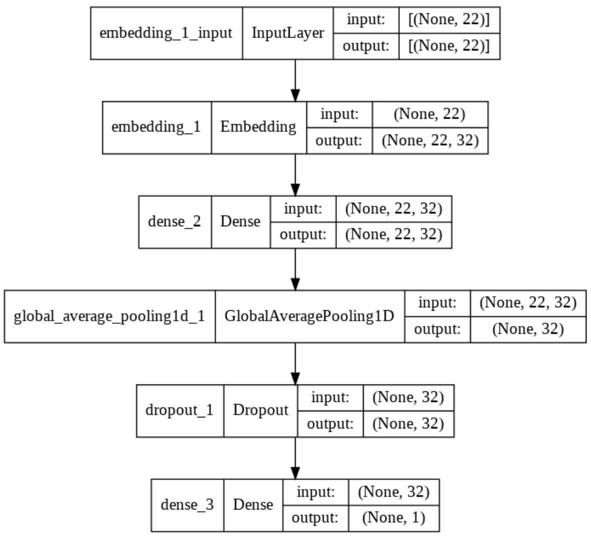


Figure 5. F1 Score on Test Set from various machine learning model

6. References

[1] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras
[2] Google Developer. Text Classification Guide. 2021. https://developers.google.com/machine-learning/guides/text-classification
[3] Koehrsen,W. Hyperparameter Tuning the Random Forest in Python. 2018. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
[4] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
[5] Qureshi, K.A. Multi-Class and Automated Tweet Categorization. arXiv cs.CL. 2021.

Neural Network Layer Explanation


embedding_1_input | InputLayer | input: [(None, 22)] | output: [(None, 22)]
embedding_1 | Embedding | input: (None, 22) | output: (None, 22, 32)
dense_2 | Dense | input: (None, 22, 32) | output: (None, 22, 32)
global_average_pooling1d_1 | GlobalAveragePooling1D | input: (None, 22, 32) | output: (None, 32)
dropout_1 | Dropout | input: (None, 32) | output: (None, 32)
dense_3 | Dense | input: (None, 32) | output: (None, 1)

Input :

22 variables each representing a word in a tweet

Embedding Layer :

- Weight :
  - model2.layers[0].get_weights()[0].shape
  - (12681,32) → this weight then is used to create a new representation of our tweets
- Output : 22 x 32 [max sequences x embedding layer]
  - Bomb = [0.87, 0.28, …, 0.37] 32 entry
  - Deeds = [0.33, 0.45, …, 0.45]
  - So if a tweet was represented with 22 numbers before, now each number will be represented by 32 numbers. So we have (22x32)
- Practically speaking, embedding layer is our input layer
- Goal of this layer : create a matrix that can represent the sentiment of each word. Therefore, we can have a group word based on its similarity
- Why use an embedding layer if you can use integer or one-hot encoding?
  - One-hot encoding is inefficient as most indices are zero -> sparse
  - Integer encoding does not reflect the relationship between words -> benefit of dense matrices
  - Embedding allows for representation of similar words wit
- Good reference to understand embedding layer :
  - https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
  - https://www.youtube.com/watch?v=nam2zR7p7Os&ab_channel=DigitalSreeni
  - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ [read until chapter 3. Example of Learning Embedding]
  - https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526 [good to read to wrap up the concept of embedding layer and the future it holds. Does not really explain how the embedding layer works in detail. At least it does not help me understand the technicality hehe]

Dense Layer :

- Goal : optimizing the weight of each word (the matrix)
- Weight: 2 vector, each with the size of 32x32
- Output: 22x32

Global Average Pooling 1D :

- Output : 32

- The GlobalAveragePooling1D layer returns a fixed-length output vector for each example by averaging over the sequence dimension. This allows the model to handle input of variable length, in the simplest way possible.
- Goal : put it in a lower dimension. From 22x32 to only 32

Drop out :
- The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.
- In human words, in our model, we have so much weight and parameters. This makes our model overfit the training set. This layer helps us to decide which layer that we can ignore to overcome the overfitting issue.
- Therefore, even though the output size is the same as input size, after passing through this layer, some parameter will be neglected
- https://keras.io/api/layers/regularization_layers/dropout/
- So if we have ratio = 0.1, means from 32 neuron, 3.2 neuron will be deactivated

Dense 3 :
- Make the final decision: natural disaster or not