Machine Learning Engineer Nanodegree

Capstone Project

Topic: Real or Not? NLP with Disaster Tweets[1]

Faraz Alam Ansari
March 17, 2020

I. Definition

Project Overview

Twitter is undoubtably one of the most popular social media platforms. It is used globally not only to express opinions, ideas and excitement but also as a means of effective communication. There is an increasing use of Twitter these days as a means of communication during emergency situations like natural calamities or other disasters. High accessibility to smartphones almost round the clock enables and encourages people to share or announce the state of emergency they are observing in real time. As a result, many disaster relief agencies are eager to programmatically monitor Twitter so as to get alerts in a real time during disaster scenarios. That's why an efficient algorithm to pick disaster tweets from the mountains of tweets every day is definitely a necessity in order to detect and identify disaster outbreak in no time, which could definitely reduce losses to human life or property by several folds.

Problem Statement

The objective of this project is to develop an algorithm to predict which tweets are about real disasters and which ones are not.

When provided with a new unseen tweet the algorithm should be able to correctly identify if it as a real Disaster tweet or not. This algorithm should ideally be able to provide a Boolean output (0 = Not-real disaster tweet, 1= real disaster tweet). This is basically a Natural Language Processing (NLP) problem and is picked from a current Kaggle competition[1].

I plan to use a set of different machine learning classifiers for this task e.g. Linear Regression, SVC, Multinomial NB, Decision Tree, Random Forest classifier, etc and then identify which algorithm is working best on our dataset.

Metrics

For evaluating my models, I plan to use 4 metrics that are widely used in machine learning classification models, namely - **accuracy, precision, recall and F1-score**. Mathematical representation for these metrices are as below.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positive, TN = True Negatives, FP = False Positives and FN = False Negative

Similarly, precision and recall can be calculated as

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

F-1 score, on the other hand can be calculated from precision

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

These are universally accepted metrics for validating the performance of machine learning classifiers and perform well for binary classification problems like ours. That's why I chose these matrices and believe they served the purpose well.

II. Analysis

Data Exploration

As suggested before, this problem is picked from a Kaggle competition[1]. The dataset too is picked from Kaggle directly.

Given dataset contains 3 files:

- train.csv - the training set
- test.csv - the test set
- sample_submission.csv - a sample submission file in the correct format

Each sample in the *train* and *test set* has the following information:

- The text of a tweet
- A keyword from that tweet (although this may be blank!)
- The location the tweet was sent from (may also be blank)

Features in the files:

- 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 train.csv only, this denotes whether a tweet is about a real disaster (1) or not (0)

Exploratory data analysis (EDA) on test and train datasets

For our study, we uploaded the test and train datasets as pandas data frames and these data frames was used for all further analysis and training.

```
Number of records in train data set: 7613
Number of records in test data set: 3263
```

As mentioned above in dataset features details, ***location*** and ***keywords*** fields may be blank in the dataset records. A quick analysis on these fields of the dataset reviels below:

For train data set:

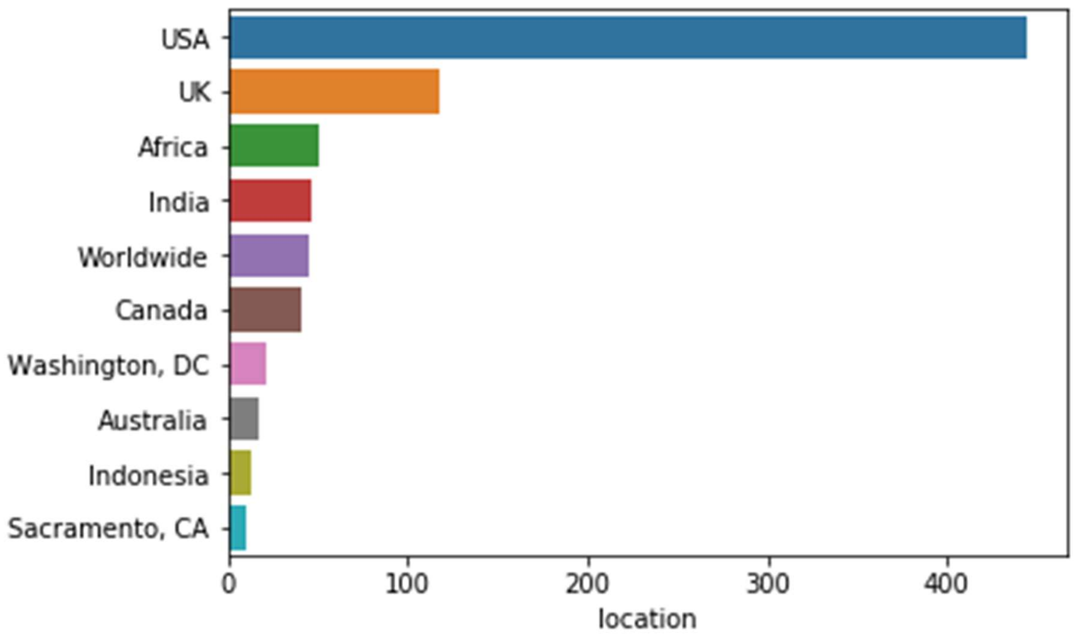
```
Number of records with missing location: 2533
Number of records with missing keywords: 61
33.0% of location values are missing from Total Number of Records.
1.0% of keywords values are missing from Total Number of Records.
```

For train data set:

```
Number of records with missing location: 1105
Number of records with missing keywords: 26
34.0% of location values are missing from Total Number of Records.
1.0% of keywords values are missing from Total Number of Records.
```

Even though the column 'location' has lots of missing values, let's check the top 20 locations present in the dataset. Since some of the locations are duplicated, lets clean the duplicate locations and plot them.

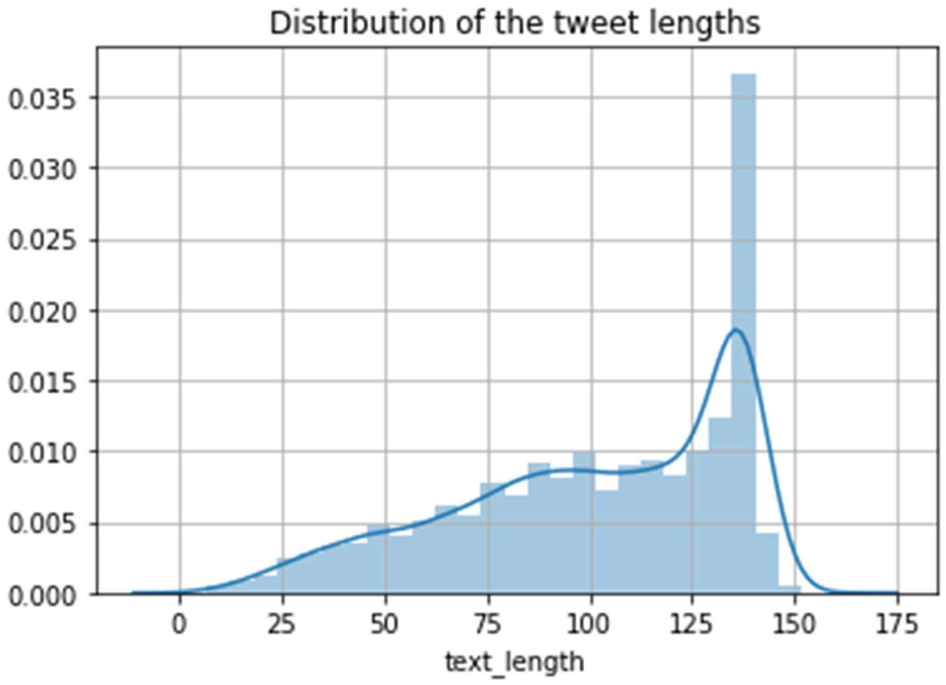
In the plot below, we are Plotting the location names along Y-axis and the number of records with given location on X-axis.



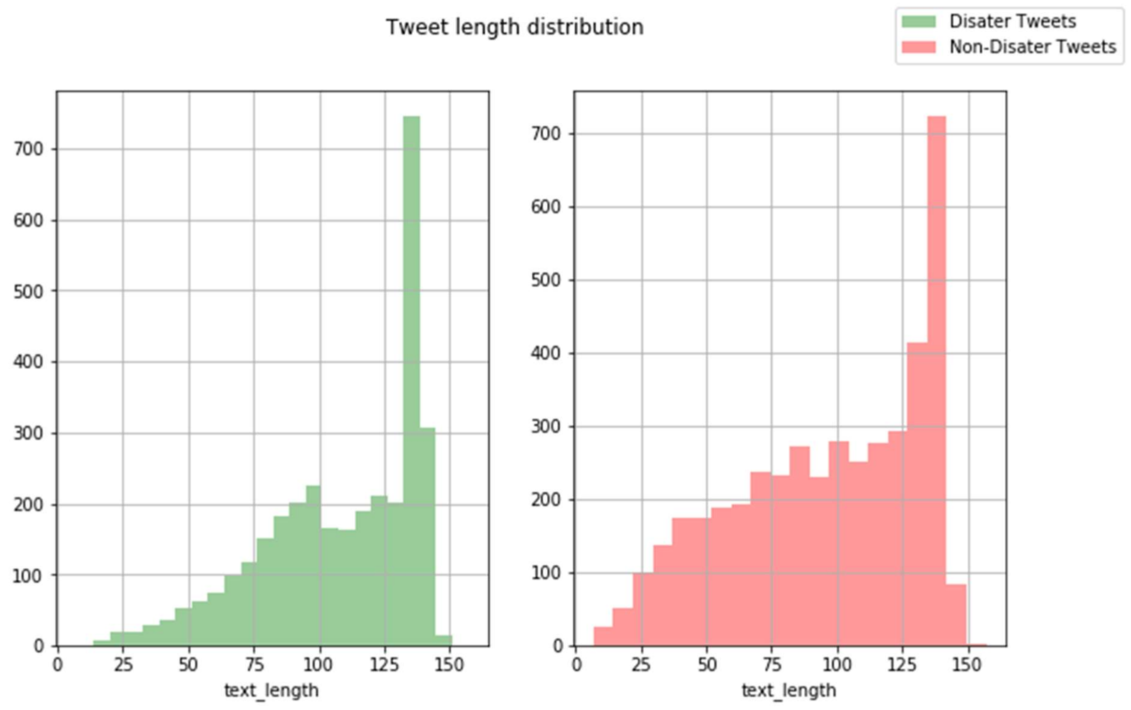
As we can see from above, highest number of location tags are for USA. Though it may be informative, it doesn't seem that location feature adds any value in our analysis or training and could be dropped from the data frame.

We further explored some of the characteristics of our datasets, that may help us in choosing the algorithm we should use for the classification of this data.

Below is the bar graph for distribution of tweet lengths

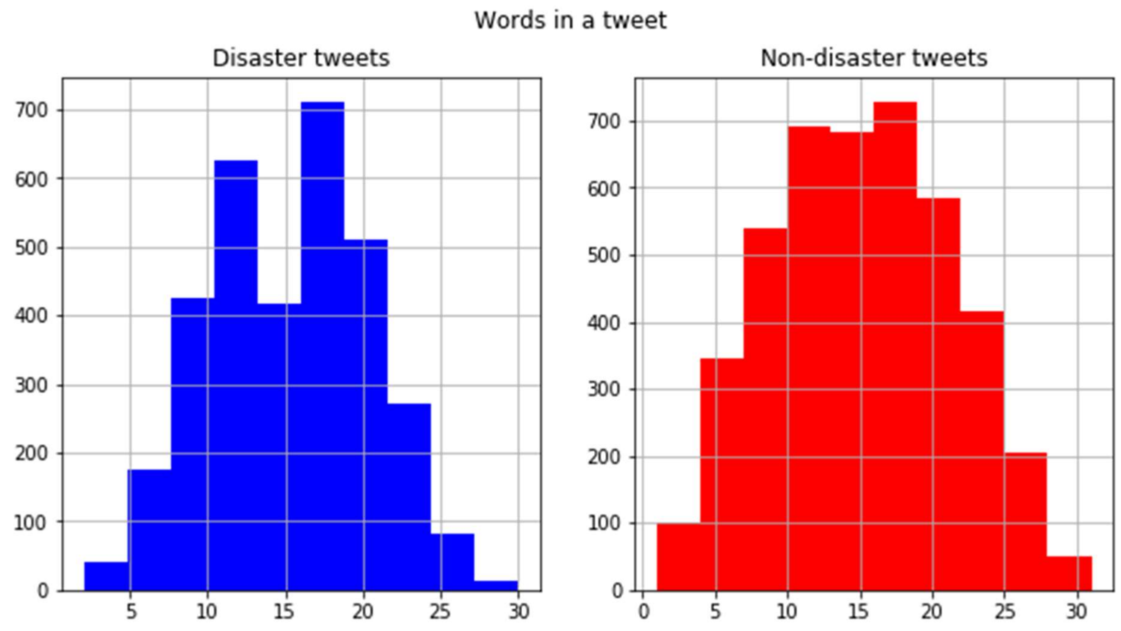


Also, the visualization of the distribution of text length in comparison to target feature (i.e. text length distribution in Non-disaster V/s disaster associated tweets).

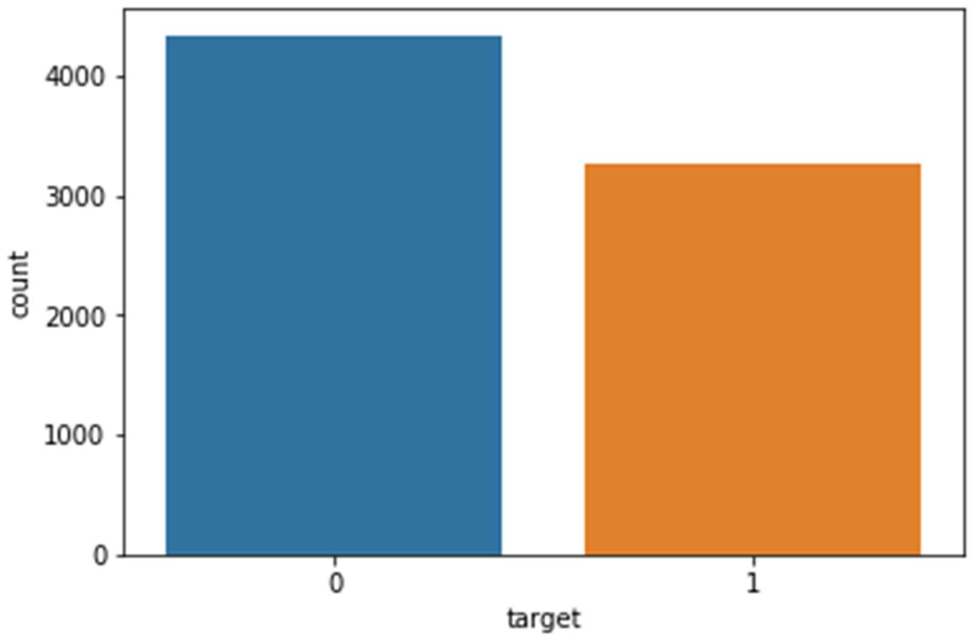


As we can see from above, the distribution of both kind of tweets seems to be almost same. About 120 to 140 characters in a tweet are the most common among both.

Distribution of word counts in Non-disaster V/s disaster associated tweets is shown below:



Finally, we would also like to explore how many tweets of each target kind (non-disaster v/s disaster related) are present in the ***training*** dataset.



```
Target of 0 is 57.0 % of total
Target of 1 is 43.0 % of total
```

From all above stats we can comfortably say that the ***training*** data set we have is a fairly balanced data set and it does not have any class imbalance, in terms of the two target classes we need to predict.

Algorithms and Techniques

The method I will be using will be a simple pipeline generally used to tackle NLP problem.

1. Text pre-processing and converting the tweets into numerical ***feature-vectors*** that could be understood by a machine learning algorithm and could be used for model training.

2. Build Machine Learning models - Lets create simple classification models using commonly used classification algorithms from ***sklearn*** and check how the model performs. We will see which algorithm performs best on our dataset and the same would be chosen for our classification task.

Benchmark

A similar type of work has been conducted and published by ***Guoqin Ma*** at Stanford [3], though he used a different dataset altogether for his study. He is using a combination of BERT and several Neural networks (LSTM, CNN, etc) and have obtained different accuracies and other metric scores with different combinations. The paper describes 5 different algorithms with differing evaluation metric scores. I plan to use the algorithms outlined in this paper as benchmark model for my project.

III. Methodology

Data Pre-processing

Before starting any NLP project, text data needs to be pre-processed to convert it into in a consistent format. Text will be cleaned, tokenized and converted into a matrix. Some of the basic text per-processing techniques includes:

1. Make text all lower or uppercase

Algorithms does not treat the same word different in different cases.

2. Removing Noise

Everything in the text that isn’t a standard number or letter i.e. Punctuation, Numerical values, etc.

3. Tokenization

Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e. words. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings.

4. Stop word Removal

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words.

5. Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemmtization”, the result would be a single word “stem”.

6. Lemmatization

A slight variant of stemming is lemmatization. The major difference between these is, that, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that ”run” is a base form for words like ”running” or ”ran” or that the word ”better” and ”good” are in the same lemma so they are considered the same.

Vectorization of pre-processed text

Pre-processed text needs to be transformed into a vector matrix of numbers before a machine learning model can understand it and learn from it. This can be done by a number of techniques:

1. Bag of Words

The bag of words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Why is it is called a “bag” of words?

It’s called bag of words because any information about the order or structure of words in the document is discarded and the model is only concerned with whether the known words occur in the document, not where they occur in the document.

2. Bag of Words — Countvectorizer Features

Countvectorizer converts a collection of text documents to a matrix of token counts. It is important to note that CountVectorizer comes with a lot of options to automatically do pre-processing, tokenization, and stop word removal. However, all the pre-processing of the text has already been performed by creating a function. Only vanilla version of Countvectorizer will be used.

3. TFIDF Features

A problem with the bag of words approach is that highly frequent words start to dominate in the document (e.g. larger score) but may not contain as much "informational content" this will lead to more weight to longer documents than shorter documents.

To avoid that, one approach is to re-scale the frequency of words by how often they appear in all documents so that the scores for frequent words like "the" that are also frequent across all documents are penalized. This approach to scoring is called "Term Frequency-Inverse Document Frequency", or TF-IDF for short, where:

- **Term Frequency**: is a scoring of the frequency of the word in the current document.

```
TF = (Number of times term t appears in a document)/(Number of terms in the
document)
```

- **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

```
IDF = 1+log(N/n), where, N is the number of documents and n is the number of
documents a term t has appeared in.
```

Implementation

1. Data pre-processing and vectorization

All the Text pre-processing steps described above were applied on the train and test data frames we have got in order to clean the training/test data before converting them to feature vectors. The cleaned data set were then converted to numerical feature vectors using two methods – i) Count-vectorizer & ii) TF-IDF.

Hence, we had two sets of feature vectors data frames ready as our training sets based on two different Vectorization methods. We will use both of them in training the Models in order to identify which vectorization method performs better.

2. Build a Machine Learning model

We will create simple classification models using commonly used classification algorithms in **sklearn** and check how the model performs.

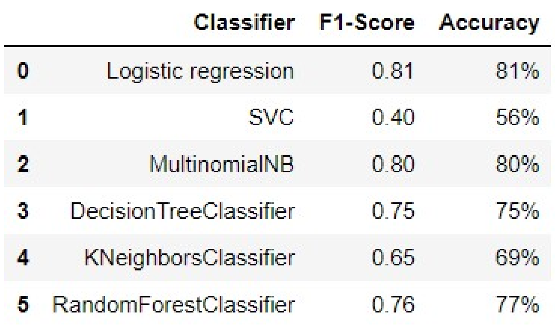
We split the 'train' data set into train and 'test' data for fitting the model using Count Vectorizer and TFIDF and train a set of models based on below machine learning algorithms:

1. Linear regression
2. Support Vector Clustering (SVC)
3. Multinomial Naïve Bayes
4. Decision Tree Classifier
5. K-Neighbors Classifier
6. Random Forest Classifier

IV. Results

Model Evaluation and Validation

Based on the training and predictions made by various classifier models , lets print the F1-score and accuracy metrics of the models to summarize the results.



	Classifier	F1-Score	Accuracy
0	Logistic regression	0.81	81%
1	SVC	0.40	56%
2	MultinomialNB	0.80	80%
3	DecisionTreeClassifier	0.75	75%
4	KNeighborsClassifier	0.65	69%
5	RandomForestClassifier	0.76	77%

As clear from above metrics, It seems from all the classifiers tested, LogisticRegression classifier performed best with CountVectorizer.

Fine tune the final classifier model

The result from LogisticRegression() classifier seems good enough than rest of the classifiers to be used as a final model. Based on the above results, it seems this classifier is not over fitting however, I tried to improve the model by tuning some hyper parameters of the classifier, but the GridSearch result with different hyper parameter suggested that the original classifier with default parameters performed best hence that will be used as the final model for test data predictions.

Reflection

Working on this data science capstone project went pretty smoothly without many issues. Even though every step in developing this project was smooth, following complications were faced:

1. The training dataset is a bit small with only about 7000 data records due to which the accuracy of the model suffers.

2. NLP is a complex field in itself and was new to me. Initially I planned to use **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)[3] which has recently gained much popularity in the area of Natural Language Processing to solve our Disaster tweets classification problem, but I faced a lot of problems in installing and then training BERT and that's why had to switch to using a combination of models in sklearn and evaluate the same.

3. Wasted much time experimenting on BERT which was little difficult as it is new and complex. Since, I had little time left to complete my capstone project, had to switch the strategy and use sklearn which was simpler to use.

4. I definitely plan to work on BERT for the same problem after completing this capstone.

Conclusion

In this capstone project, I took a Kaggle challenge to classify tweets into disaster tweets or non-disaster ones. First, I have analysed and explored all the provided tweets data to visualize the statistical and other properties of the presented data.

Next, I performed some exploratory analysis of the data to check type of the data, whether there are unwanted features and if features have missing data. Based on the analysis, I decided to drop 'location' column since it has most missing data and really had no effect in classification of tweets. The 'text' column consisted of all text data along with alpha numeric, special characters and embedded URLs and it required to to be cleaned, pre-processed and vectorized before it can be used with a machine learning algorithm for the classification of the tweets.

After pre-processing the train and test data, the data was vectorized using ***CountVectorizer*** and ***TFIDF*** features and then it was split into training and test data. Various classifiers were fit on the data and predictions were made. Out of all classifiers tested, LogisticRegression performed the best with the test accuracy of 81%. An effort was made to tune some hyper parameters of the final classifier to see if accuracy can be improved. It turned out that classifier with default parameters preformed little better than tuned model .

Finally, LogisticRegression() classifier with default parameters was selected as final model for this NLP classifications challenge and predictions were made on the test data. Kaggle submission[3] of the predictions resulted in an accuracy of 79%.

Improvements

I believe there are definitely opportunities to try out various machine learning models to see if there are better performing models that could be used to solve this problem with higher than 81% test accuracy.

Trying BERT, deep neural networks models such as RNN (Recurrent Neural Network), XBoost might result in higher accuracy and better performance but may be very costly in terms of compute resources used. Another way to improve the accuracy of the models would be to have more training data than given

References

[1] https://www.kaggle.com/c/nlp-getting-started.

[2] https://www.figure-eight.com/data-for-everyone/

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proc. of NAACL, 2018

[4] Guoquin Ma. Tweets Classification with BERT in the Field of Disaster Management. Stanford Univ. https://web.stanford.edu/class/cs224n/reports/custom/15785631.pdf