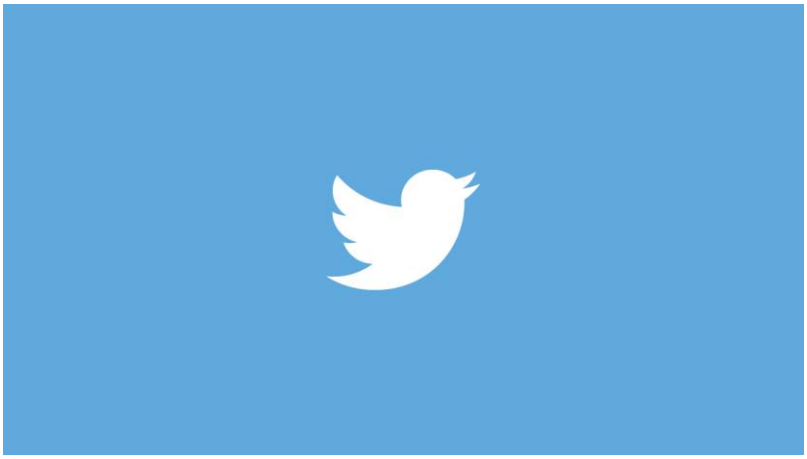


REAL OR NOT? NLP WITH DISASTER TWEETS

TERM PROJECT REPORT

SCS 3253_022

Vaishali Bhat,
Olga Senko

Table of Contents

1. Frame the problem and look at the big picture

Twitter, is a microblogging communication technology, was founded by Jack Dorsey and associates in San Francisco in 2006. It was originally developed to be an urban lifestyle tool for friends to provide each other with updates of their whereabouts and activities. However, with its changed tagline from 'What are you doing?' to 'What's happening?' it has developed into a reporting and communication medium useful in many fields including emergency management

Twitter allows users to distribute short messages (tweets) on the internet through smartphone apps or web browser. Over the years, various additional features have been included in the backend and the interface e.g. facilities for picture upload and display, automatic shortening of URLs to save characters in tweets. Through an API (Application Programming Interface), third-party applications which offer additional functionalities can be connected to the service

In the Global picture, We Are Social report (2020) estimated that **3.8 billion** and the latest trends suggest that **more than half** of the world's total population will use social media by the middle of this year. As shown in Picture 1 below, approximately 64 per cent of social media users accessed Facebook, while about 9 per cent (340 million) used Twitter. Twitter growth is slower than the average global growth rate for social media.

There is a difference in spread of the use of social media usage around the world. For example, in North America 82 per cent of the population use social media, while only 30 per cent in Central Asia use social media (Picture 2). There are very low levels of Twitter use in some countries such as Russia that have their own language social networks, while other countries have usage rates above the global average, such as the Saudi Arabia (53 per cent), Australia (30 per cent), United Kingdom (29 per cent), United States of America (21 per cent), and Malaysia (15 per cent) (Picture 3). In Canada, 67 per cent of the population used social media in 2020 with 79 per cent using Facebook and 40 per cent using Twitter (Picture 4). According to the recent research, more males than females use Twitter (Picture 5).

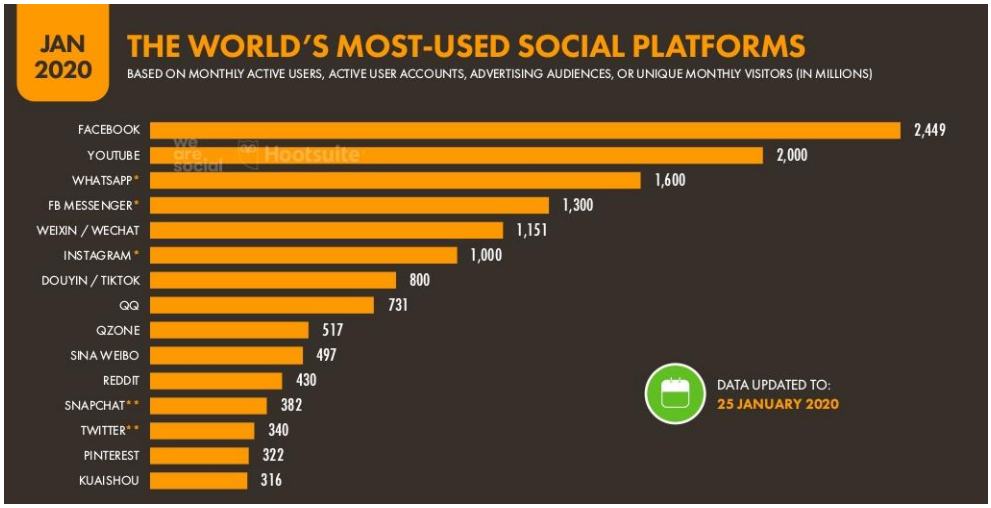
During disasters, affected individuals often turn to social media platforms, such as Twitter and Facebook, to find the latest updates from government and response organizations, to request help or to post information that can be used to enhance situational awareness. A large proportion of the research into the use of social media in emergency management has focused on Twitter, even though the global uptake of Facebook is substantially higher, and despite Facebook being used more extensively in disasters to date (Irons et al. 2014). This is largely because Twitter has some unique characteristics that are, at this stage, more useful to disaster management and research.

At the same time, the value of the information posted on social media platforms during disasters is highly unexploited, in part due to the lack of tools that can help filter relevant, informative, and actionable messages. According to one of the recent reports more than 5, 200 rescue requests made on social media were missed by the first responders, while about 46% of the critical damage information posted on social media during Harvey Hurricane was missed by FEMA in their original damage estimates (that is almost half of the total costs of $125 billion estimated for this hurricane). As an official explained: "It's very labor intensive to watch [social media] and because of the thousand different ways people can hashtag something or keyword something, trying to sort out what's relevant and what's not and what's actionable is very, very difficult" (Silverman 2017)

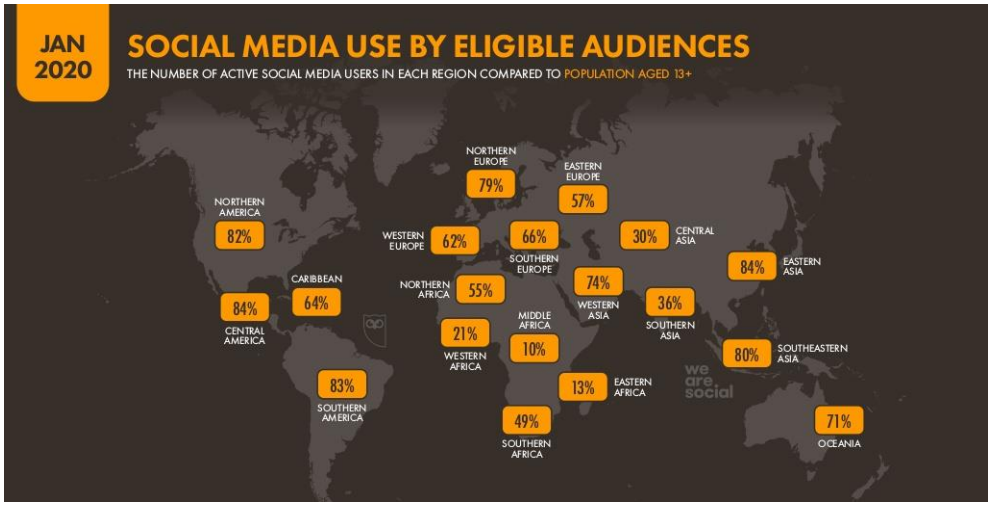
In the chosen Kaggle open competition, we're challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. We had access to a dataset of 10,000 tweets that were hand classified

The best performing model achieves an F1-score as high as 84.04%. The dataset, code, and other resources from this work are made available on Github.

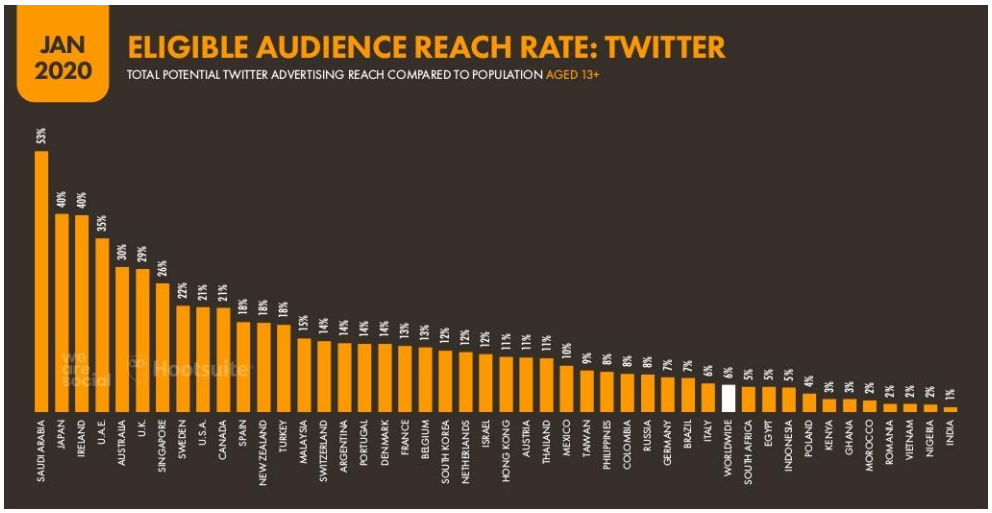
Picture 1.
Source: We are Social 2020. Jan 2020



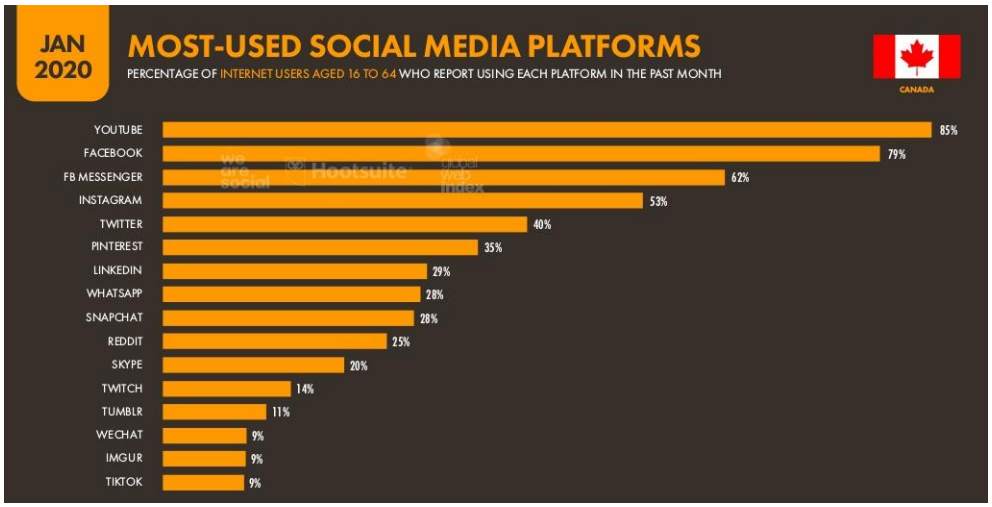
Picture 2.
Source: We are Social 2020. Jan 2020



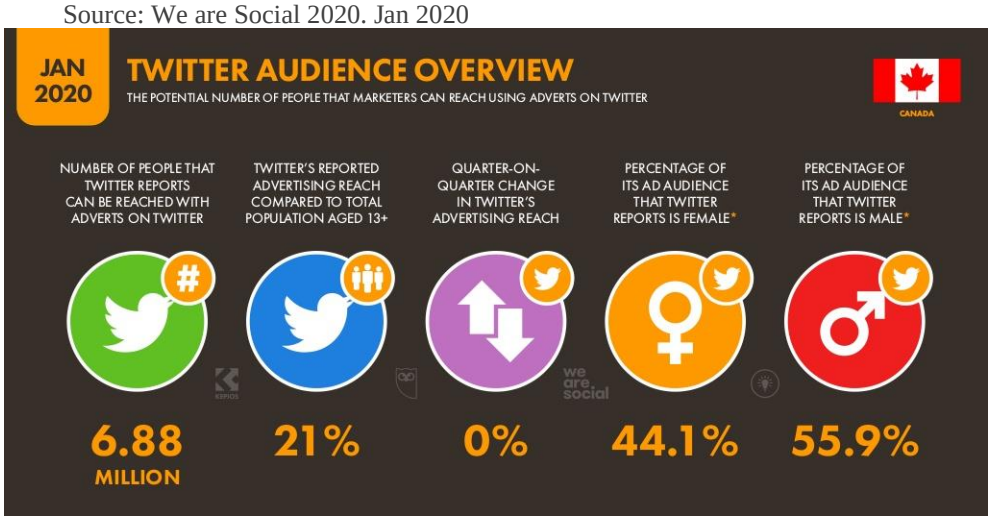
Picture 3.
Source: We are Social 2020. Jan 2020



Picture 4.
Source: We are Social 2020. Jan 2020



Picture 5.
Source: We are Social 2020. Jan 2020



Business Objective

Based on the information we gathered through the initial stages of our analysis for social media usage, we identified potentially growing demand for new solution in the market. News agencies as well as disaster relief organizations are interested in the solution that can help decrease labor intensity of social listening and increase speed of reaction for providing relief where needed.

2. Get the data

To further investigate, we analyzed the given dataset which comprises of 10,000 tweets that were hand classified. The challenge is to build a binary classifier that predicts which Tweets are about real disasters and which one's aren't. I.e the output is a whether a given tweet is about a real disaster or not. If so, predict a 1. If not, predict a 0.

Data include 5 columns:

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

Data were available on Kaggle platform: https://www.kaggle.com/c/nlp-getting-started/data were we found 3 files:

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

3. Explore the data

As train and test sets were pre-set we would start with checking on their split and investigate on details for train set.

```
df_train = pd.read_csv('./train.csv', dtype={'id': np.int16, 'target': np.int8})
df_test = pd.read_csv('./test.csv', dtype={'id': np.int16})

print('Training Set Shape = {}'.format(df_train.shape))
print('Test Set Shape = {}'.format(df_test.shape))
```

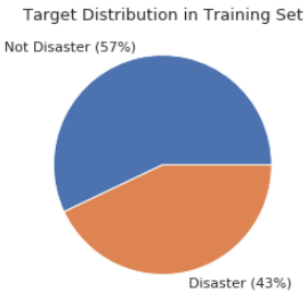
```
Training Set Shape = (7613, 5)
Test Set Shape = (3263, 4)
```

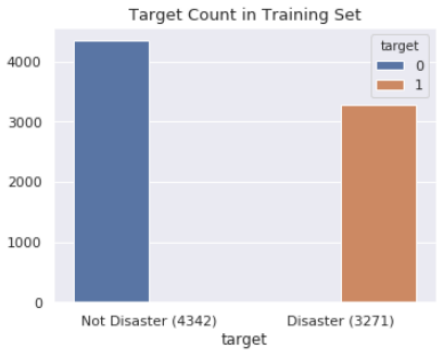
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
id          7613 non-null int16
keyword     7552 non-null object
location    5080 non-null object
text        7613 non-null object
target      7613 non-null int8
dtypes: int16(1), int8(1), object(3)
memory usage: 200.9+ KB
```

From the above code we see that the Train Data has 7613 rows and 5 columns & The Test Data has 3263 rows and 4 columns. We also understand that Text is all non-null. Only a small percentage of tweets have no keyword whereas Location has much more null values.

In this section, we will first find out the distribution of Target/Predictor variable i.e. Target

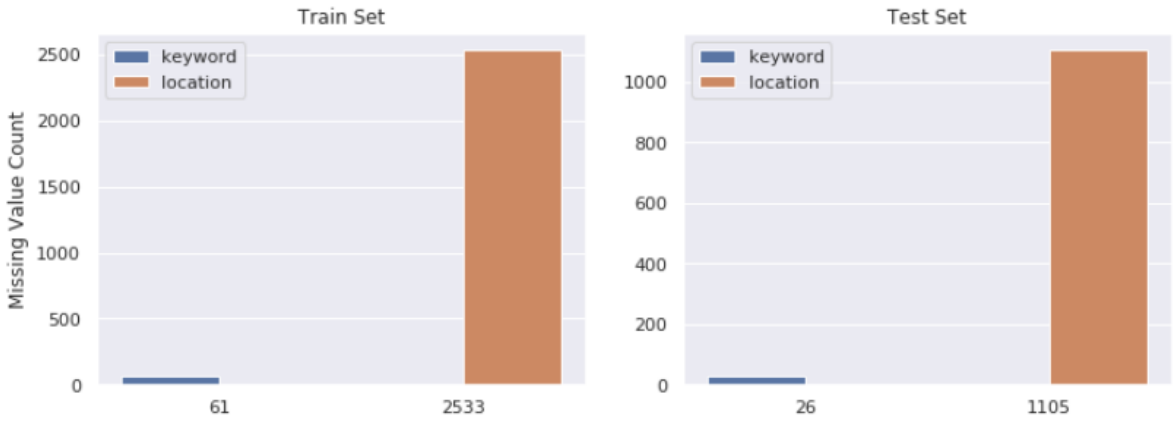




	Sample Disaster tweets	Sample non-Disaster tweets
0	Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all	What's up man?
1	Forest fire near La Ronge Sask. Canada	I love fruits
2	All residents asked to 'shelter in place' are being notified by officers. No other evacuation or shelter in place orders are expected	Summer is lovely

From the above plots it is clear that the classes are not skewed but are quite balanced. Class distributions are 57% for 0 (Not Disaster) and 43% for 1 (Disaster) and hence, they don't require any stratification by target in cross-validation.

Now we will move on to look into the Features-Location and Keyword on both Train and Test.



Percentage of missing attribute-keywords on Test=0.8 %
Percentage of missing attribute-keywords on Train=0.8 %

Percentage of missing attribute-location on Test=33.86 %
Percentage of missing attribute-location on Train=33.27 %

From the above percentages it is clear that, both training and test set have same ratio of missing values in keyword and location.

- 0.8% of keyword is missing in both training and test set
- 33% of location is missing in both training and test set

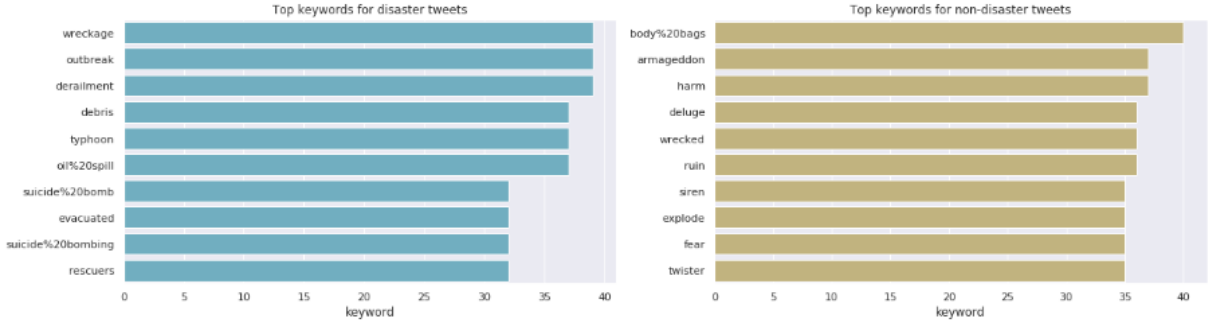
Since missing value ratios between training and test set are too close, they are most probably taken from the same sample.

Next, we will check the number of unique values for both keyword and location attributes.

Number of unique values in keyword = 221 in Train & 221 in Test
Number of unique values in location = 3341 in Train & 1602 in Test
From the above numbers, we can see that

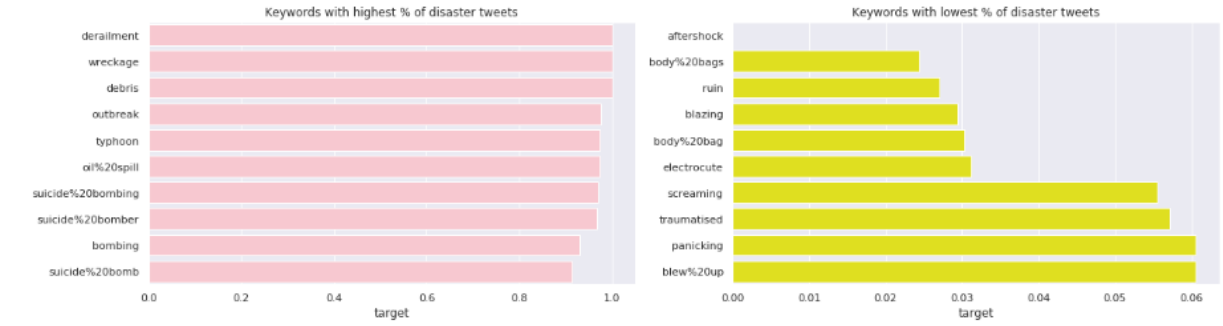
- Train and test have the same set of keyword.
- Locations are not automatically generated, they are user inputs. That's why location is very dirty and there are too many unique values in it. Hence, we would not used it as a feature.

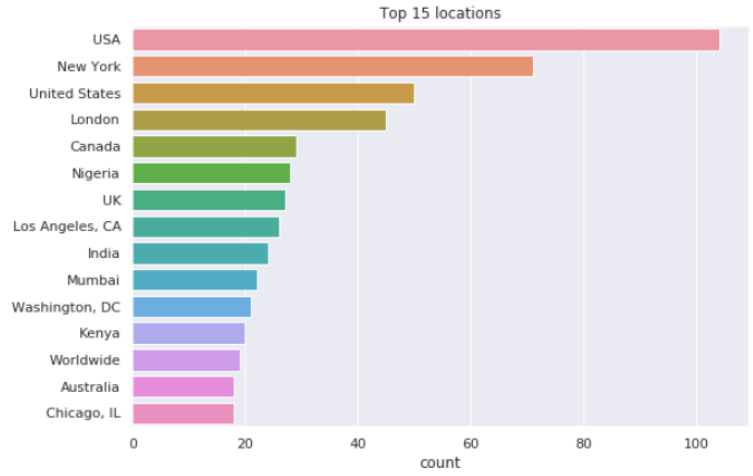
Now let us next find out the most common Keyword attribute.



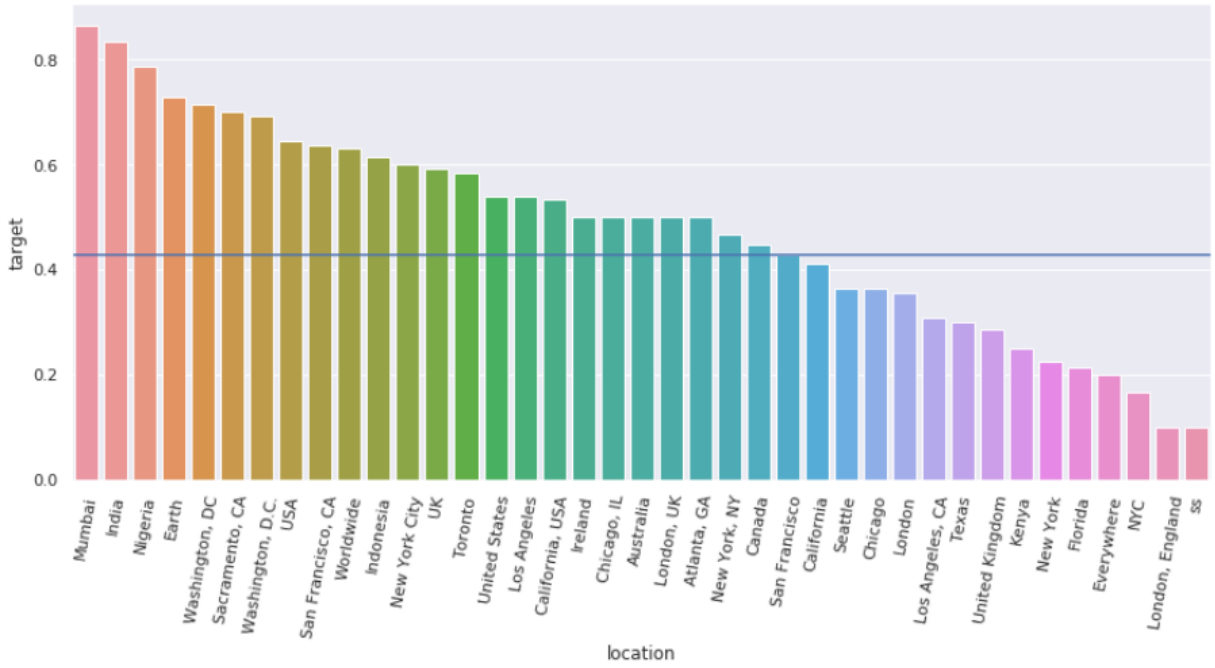
We can see from the above graphs that there is no common top 10 keywords between disaster and non-disaster tweets.

Let us now see, the keywords with highest number of disaster tweets and non-disaster tweets.





As location is free text, the data is not clean, you can see both 'USA' and 'United States' in top locations. We will now have a look at % of disaster tweets for common locations.

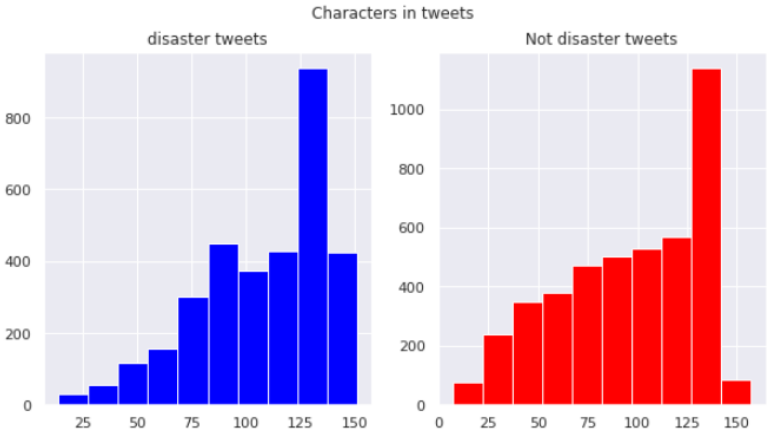


Below are our findings:

1. The top 3 locations with highest % of disaster tweets are Mumbai, India, and Nigeria.

2. As the location data is not clean, we see some interesting cases, such as 'London, UK' saw a higher-than-average % of disaster tweets, but 'London' is below average. We will try to clean up the location and see if there is any difference

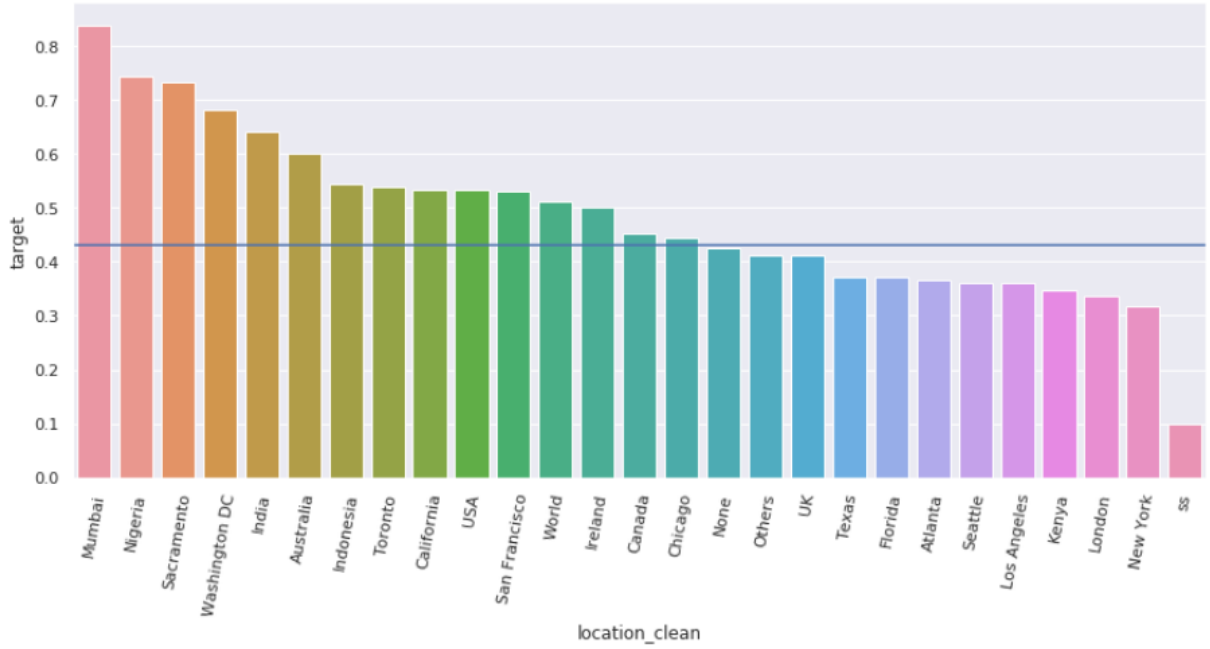
3.1 Analyse Text Feature



From the above distributions we can see that The distribution of both seems to be almost same. 120 to 140 characters in a tweet are the most common among both.

4. Prepare the data

Cleaning Free Text of Location attribute



We noticed that after cleaning the Location column, Mumbai and Nigeria are still on the top. Other than the strange 'ss', London and New York made the bottom of % of disaster tweets.

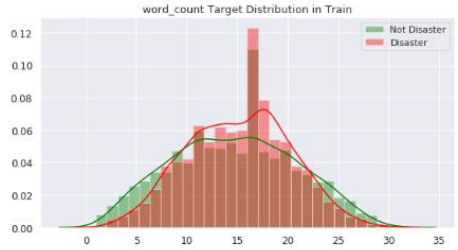
Feature Engineering

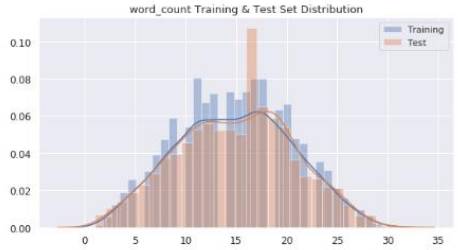
We will be creating the following list of features for Text attribute for the analysis

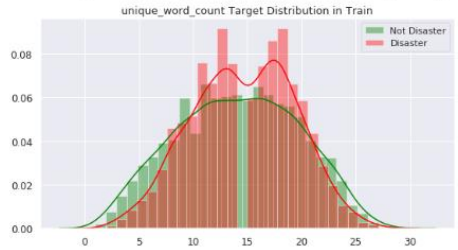
- word_count - number of words in text
- unique_word_count - number of unique words in text
- stop_word_count - number of stop words in text
- url_count - number of urls in text
- mean_word_length - average character count in words
- char_count - number of characters in text
- punctuation_count - number of punctuations in text
- hashtag_count - number of hashtags (#) in text
- mention_count - number of mentions (@) in text

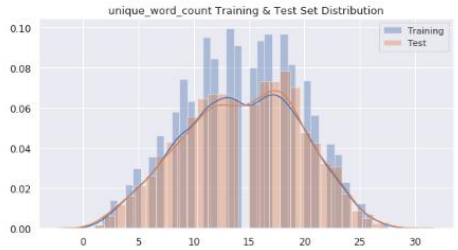
```
stop_word_count     -0.111250
mention_count       -0.103343
word_count           0.039966
unique_word_count    0.053326
hashtag_count        0.058115
id                   0.060781
punctuation_count    0.132606
mean_word_length     0.176855
char_count           0.181817
url_count            0.195455
Name: target, dtype: float64
```

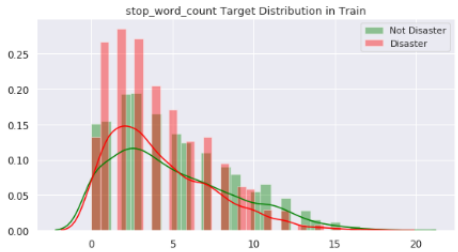
From the above correlation statistics we infer that these meta-features have very low correlation with the target variable.

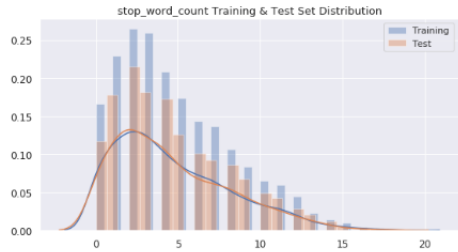

word_count Target Distribution in Train
Not Disaster
Disaster

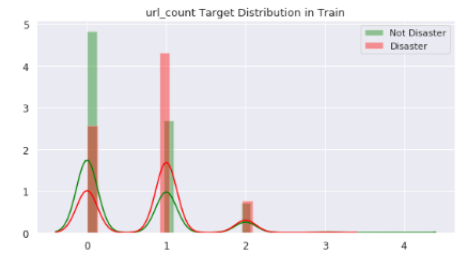

word_count Training & Test Set Distribution
Training
Test

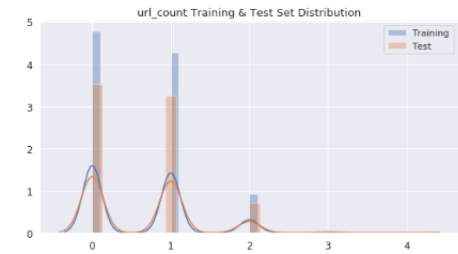

unique_word_count Target Distribution in Train
Not Disaster
Disaster

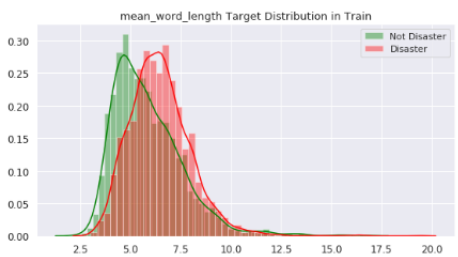

unique_word_count Training & Test Set Distribution
Training
Test

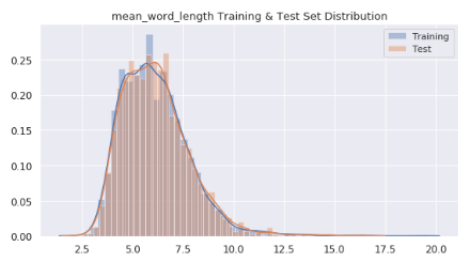

stop_word_count Target Distribution in Train
Not Disaster
Disaster

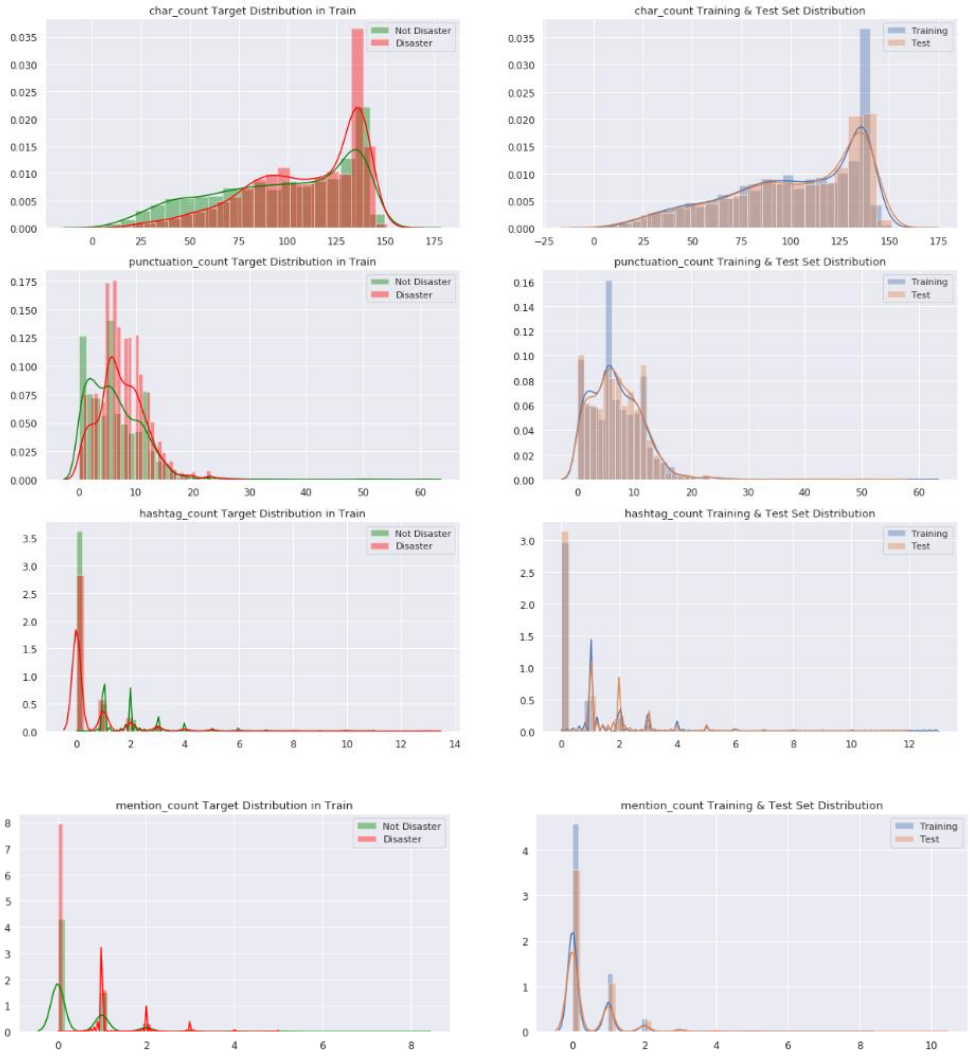

stop_word_count Training & Test Set Distribution
Training
Test


url_count Target Distribution in Train
Not Disaster
Disaster


url_count Training & Test Set Distribution
Training
Test


mean_word_length Target Distribution in Train
Not Disaster
Disaster


mean_word_length Training & Test Set Distribution
Training
Test



Below is our findings from the above plots:

1. All of the meta features have very similar distributions in training and test set which also proves that training and test set are taken from the same sample.
2. All of the meta features have information about target as well, but some of them are not good enough such as url_count, hashtag_count and mention_count.
3. On the other hand, word_count, unique_word_count, stop_word_count, mean_word_length, char_count, punctuation_count have very different distributions for disaster and non-disaster tweets. Those features might be useful in models.

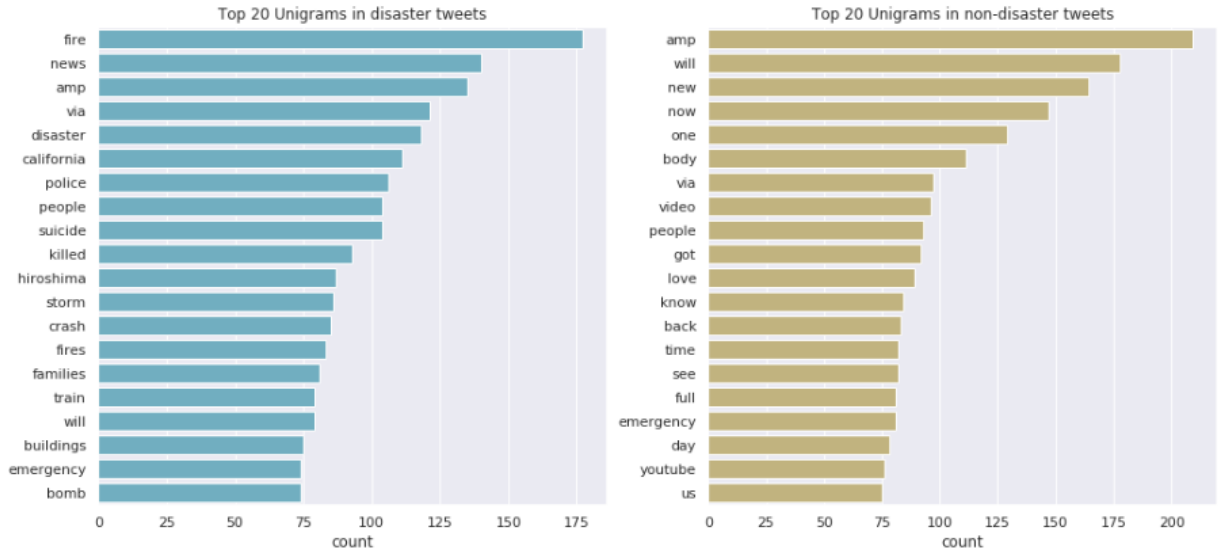
Clean Text feature

Here we clean up the text column by:

1. Making a 'clean' text column, removing links and unnecessary white spaces
2. Creating separate columns containing lists of hashtags, mentions, and links

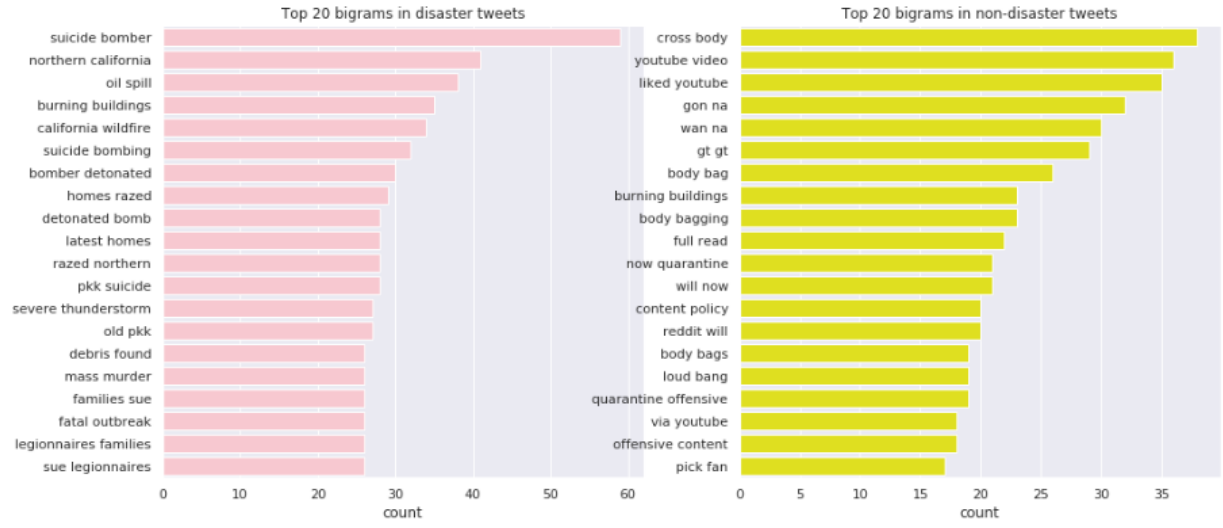
Unigrams

Most common unigrams exist in both classes are mostly punctuations, stop words or numbers. So, it is better to clean them before analyzing, since they don't give much information about target.



Below are our findings:
Top two words in disaster tweets: 'fire' and 'news', don't make the top 20 on unreal disaster tweets.
Words are more specific for real disaster tweets (e.g. 'califonia', 'hiroshima', 'fire', 'police', 'suicide', 'bomb').

Bigrams



Following are our findings:

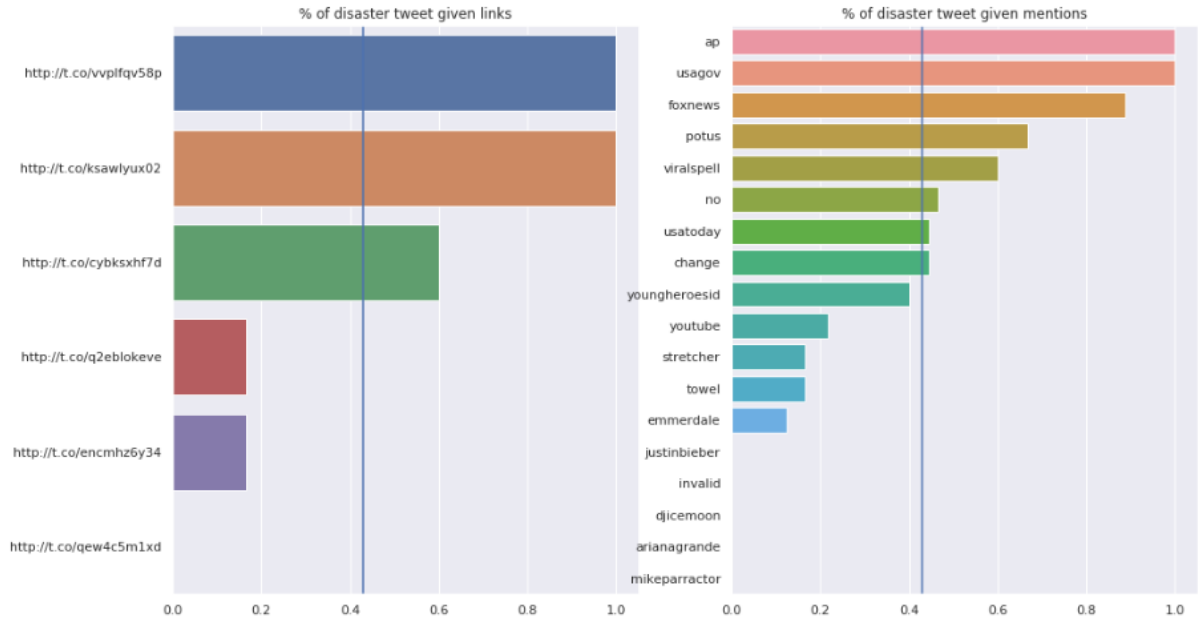
- There are no common bigrams that exist in both classes.
- Most top bigrams in disaster tweets show certain kinds of catestrophe (e.g. suicide bomber, oil spill, california wildfire); for non-disaster tweets, only 'burning buildings' as top bigram look like a disaster;

- Top bigrams in disaster tweets have a more casual tone;
- 'youtube' appears in three of the twenty bigrams for non-disaster tweets; none in disaster tweets

.

Count Hashtags, Mentions and Urls in Text feature

The below results of code uses Count Vectorizer on cleaned text to count the number of links, hashtags and mentions columns



Analysis using Pre-trained Embeddings

When you have pre-trained embeddings, doing standard preprocessing steps might not be a good idea because some of the valuable information can be lost. So, it is better to get vocabulary as close to embeddings as possible. In order to do that, train vocab and test vocab are created by counting the words in tweets.

Text cleaning is based on the embeddings below:

1. GloVe-300d-840B
2. FastText-Crawl-300d-2M

GloVe Embeddings cover 52.06% of vocabulary and 82.68% of text in Training Set
GloVe Embeddings cover 57.21% of vocabulary and 81.85% of text in Test Set

FastText Embeddings cover 51.52% of vocabulary and 81.84% of text in Training Set
FastText Embeddings cover 56.55% of vocabulary and 81.12% of text in Test Set

And after some cleaning that involves removing Special characters, Contractions, Character entity references, Typos, slang and informal abbreviations, Hashtags and usernames, Urls, Words with punctuations and special characters, # ... and .., Acronyms, Grouping same words without embeddings
GloVe Embeddings cover 82.89% of vocabulary and 97.14% of text in Training Set
GloVe Embeddings cover 88.09% of vocabulary and 97.32% of text in Test Set

FastText Embeddings cover 82.88% of vocabulary and 97.12% of text in Training Set
FastText Embeddings cover 87.80% of vocabulary and 97.25% of text in Test Set

5. Explore different models

5.1 XGBoost Model

We have started with building t the a XGBoost Model and tuned the hyperparameters using GridSearch
C word embedding- transformation from words to vectors.
The challenge with textual data is that it needs to be represented in a format that can be mathematically used in solving some problem. In simple words, we need to get an integer representation of a word.

One of the most commonly used technique to vectorize text by computing TD-IDF. Let us first cover the theoretical part of this before implementing this.

TF-IDF stands for term frequency-inverse document frequency which assigns weights to the word based on the number of occurrences in the document and even takes into consideration the frequency of the word in all the documents. TF-IDF is an acronym that stands for "Term Frequency – Inverse Document" Frequency which is a product of the below scores assigned to each word.

Term Frequency: This summarizes how often a given word appears within a document.

$$TF(t) = (Number_of_times_term\ appears_in\ document) / (Total_number_of_terms_in_the_document)$$

Inverse Document Frequency: This downscales words that appear a lot across documents.

$$IDF(t) = log_e(Total_number_of_documents / Number_of_documents_with_term\ in\ it)$$

This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

The scores are normalized to values between 0 and 1 and the encoded document vectors can then be used directly with most machine learning algorithms. This approach is better than BOW(Bag of Words) which just counts the occurrences of words because TF-IDF lowers the weight of the words that occur too often in all the sentences like 'a', 'the', 'as' etc and increases the weight of the words that can be important in a sentence. This is useful in the scenarios where we want to get the important words from all the documents and is commonly used in topic modelling.

The logic of the below piece of code is explained here.

1. Tokenize the each english sentence in "Text" feature of Train into seperate words.

2. Use python implementation of **Word2Vec** from **gensim** library to generate vector representations of words that carry semantic meanings for further NLP tasks. Each word vector is typically represented using several hundred dimensions and each unique word in the our Train(also called corpus) is assigned a vector in the space. Below is the list of Hyperparameters for Word2Vec chosen:
   'size' which is dimensions of word embeddings = 50

'min_count' which indicate the all the words with frequency less than this value = 5
'workers' which indicates worker threads to train the model = 4

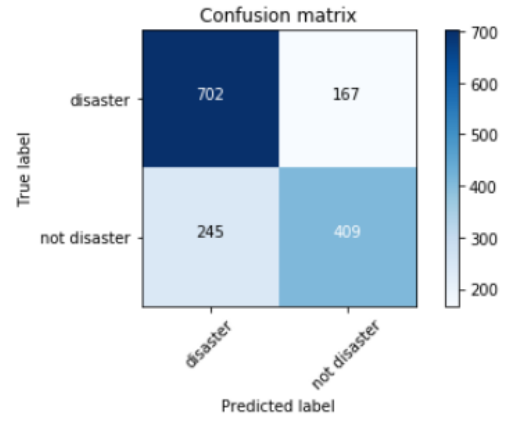
3. Calculate TF-IDF . Following is the list of hyperparameters of TF-IDF chosen:
   'max_features' build a vocabulary that only consider the top max_features ordered by term frequency across the corpus =2500
   'min_df' hich indicates to ignore terms that have frequency strictly lower than the given threshold=7
   'max_df' which indicates to ignore terms that have frequency strictly higher than the given threshold = 0.8

4. Loop through every text Feature in Train to multiply its' TF-IDF value with its word2Vec weight.

We do the above steps for Test as well. Finally, split Train into Train and Val in the ratio 80:20

Confusion Matrix and Classification Report¶

```
Confusion matrix, without normalization
```



	precision	recall	f1-score	support
0	0.74	0.80	0.77	869
1	0.70	0.63	0.66	654
accuracy			0.72	1523
macro avg	0.72	0.71	0.71	1523
weighted avg	0.72	0.72	0.72	1523

```
0.7235718975705844
```

5.2 Neural Network

We were training a Neural Network on our dataset. Below is a list of that we have try.

1. Word Embeddings using Bag of Words
2. Word Embeddings using TF IDF
3. Word Embeddings using GloVe

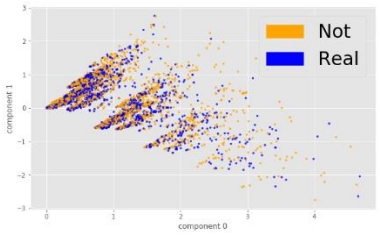
Word embedding: transformation from words to vectors.¶

The challenge with textual data is that it needs to be represented in a format that can be mathematically used in solving some problem. In simple words, we need to get an integer representation of a word.

1. Bag of Words

One of the common techniques used is Bag of Words Counts which is implemented in Scikit-learn's CountVectorizer. This is used to transform a corpora of text to a vector of term / token counts. The output of the CountVectorizer is a Sparse Matrix. Each column in the matrix represents a unique word in the vocabulary, while each row represents the sample(or document) in our dataset.

Later, we will visualize the embeddings. We are using truncated SVD for Dimensionality reduction and plotting the components.



These embeddings shown above don't look very cleanly separated. Let's see if we can use a different Vectorizer.

2. TF IDF

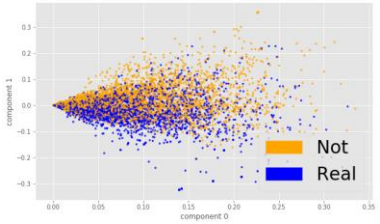
TF-IDF converts a collection of raw documents to a matrix of TF-IDF features. This is an acronym that stands for "Term Frequency – Inverse Document" Frequency which is a product of the below scores assigned to each word.

Term Frequency: This summarizes how often a given word appears within a document.
Inverse Document Frequency: This downscales words that appear a lot across documents.

This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

The scores are normalized to values between 0 and 1 and the encoded document vectors can then be used directly with most machine learning algorithms.

Later, we will visualize the embeddings. We are using truncated SVD for Dimensionality reduction and plotting the components.



3. GloVe Embeddings

GloVe is an acronym for Global Vectors for Word Representation. This allows us to take a corpus of text, and intuitively transform each word in that corpus into a position in a high-dimensional space which means that similar words will be placed together.

The first task is to download pre-trained word vectors that is available in 3 varieties : 50D, 100D and 200 Dimensional. We will try 100D here. Before we load the vectors in code, we have to understand how the text file is formatted. Each line of the text file contains a word, followed by N numbers. The N numbers describe the vector of the word's position. N may vary depending on which vectors is downloaded, for us, N is 100, since we are using glove.6B.100d.

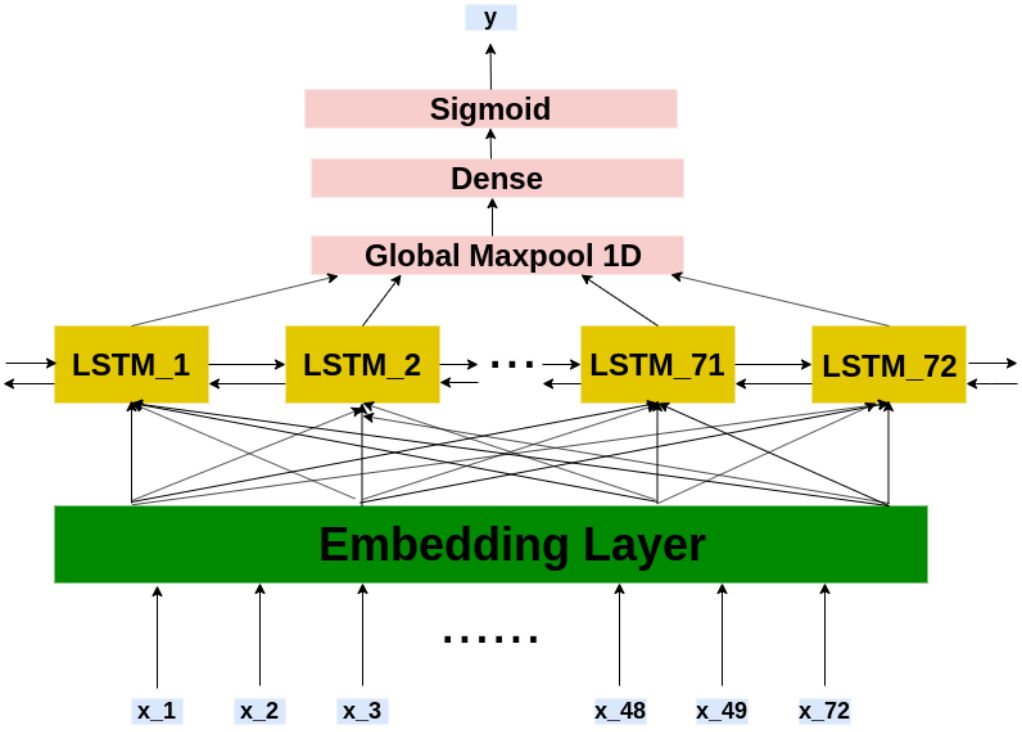
Below is the summary of tasks if the below code:

1. parse the GloVe vectors file and build the dictionary of it's vectors.
2. text is split into words using word_tokenize to build the corpus.
3. build the dictionary consisting of unique words from the corpus.
4. create embedding vectors for each sample in Train with maximum length of 72 words. We also tried an option with maximum 50 words

Notice the Train shape printed below. It has changed from 5 features to 72 features

Train Bidirectional LSTM Model with pre-trained GloVe word embeddings

In the below section we will build 5 networks and train it using GloVe features as the inputs. The architecture of all the networks is same and below is it's pictorial representation



Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 72, 200)	4540200
bidirectional_2 (Bidirection	(None, 72, 144)	157248
global_max_pooling1d_2 (Glob	(None, 144)	0
batch_normalization_2 (Batch	(None, 144)	576
dropout_4 (Dropout)	(None, 144)	0
dense_4 (Dense)	(None, 72)	10440
dropout_5 (Dropout)	(None, 72)	0
dense_5 (Dense)	(None, 72)	5256
dropout_6 (Dropout)	(None, 72)	0
dense_6 (Dense)	(None, 1)	73

Total params: 4,713,793
Trainable params: 173,305
Non-trainable params: 4,540,488

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 72, 200)	4540200
bidirectional_3 (Bidirection	(None, 72, 144)	157248
global_max_pooling1d_3 (Glob	(None, 144)	0
batch_normalization_3 (Batch	(None, 144)	576
dropout_7 (Dropout)	(None, 144)	0
dense_7 (Dense)	(None, 72)	10440
dropout_8 (Dropout)	(None, 72)	0
dense_8 (Dense)	(None, 72)	5256
dropout_9 (Dropout)	(None, 72)	0
dense_9 (Dense)	(None, 1)	73

Total params: 4,713,793
Trainable params: 173,305
Non-trainable params: 4,540,488

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 72, 200)	4540200
bidirectional_4 (Bidirection	(None, 72, 144)	157248
global_max_pooling1d_4 (Glob	(None, 144)	0
batch_normalization_4 (Batch	(None, 144)	576
dropout_10 (Dropout)	(None, 144)	0
dense_10 (Dense)	(None, 72)	10440
dropout_11 (Dropout)	(None, 72)	0
dense_11 (Dense)	(None, 72)	5256
dropout_12 (Dropout)	(None, 72)	0
dense_12 (Dense)	(None, 1)	73

Total params: 4,713,793
Trainable params: 173,305
Non-trainable params: 4,540,488

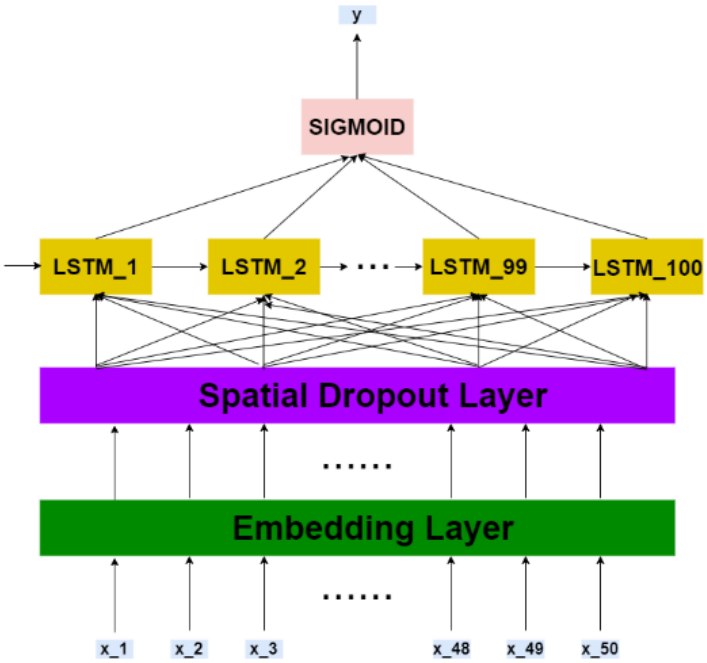
Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 72, 200)	4540200
bidirectional_5 (Bidirection	(None, 72, 144)	157248
global_max_pooling1d_5 (Glob	(None, 144)	0
batch_normalization_5 (Batch	(None, 144)	576
dropout_13 (Dropout)	(None, 144)	0
dense_13 (Dense)	(None, 72)	10440
dropout_14 (Dropout)	(None, 72)	0
dense_14 (Dense)	(None, 72)	5256
dropout_15 (Dropout)	(None, 72)	0
dense_15 (Dense)	(None, 1)	73

Total params: 4,713,793
Trainable params: 173,305
Non-trainable params: 4,540,488

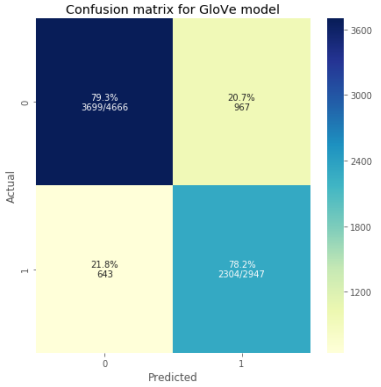
After that we run prediction on test and create submission file that will be uploaded to kaggle.

Baseline Model with GloVe results for limitation of 50 words

In the below section we will build a network and train it using GloVe features as the inputs. The architecture of the network is shown below.



And the confusion matrix for GloVe model (50 words limitation)



5.3 BERT by Google AI language

In the field of computer vision, researchers have repeatedly shown the value of transfer learning — pre-training a neural network model on a known task, for instance ImageNet, and then performing fine-tuning i.e. using the trained neural network as the basis of a new purpose-specific model. In recent years, researchers have been showing that a similar technique can be useful in many natural language tasks. We will show how a pre-trained neural network produces word embeddings which are then used as features in NLP models.

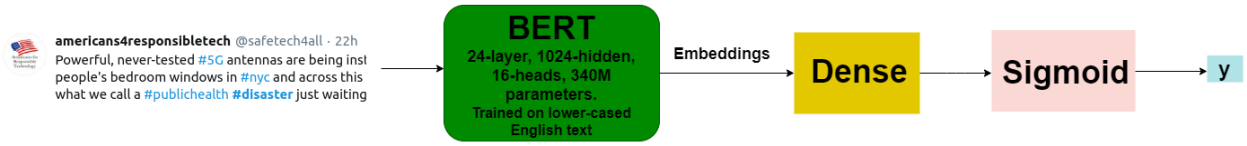
Now, we are trying a different model called BERT. BERT, which stands for Bidirectional Encoder Representations from Transformers and one of it's applications is text classification. BERT is a text representation technique like Word Embeddings. Like word embeddings, BERT is also a text representation technique which is a fusion of variety of state-of-the-art deep learning algorithms, such as bidirectional encoder LSTM and Transformers. BERT was developed by researchers at Google AI Language in 2018 and has been proven to be state-of-the-art for a variety of natural language processing tasks such text classification, text summarization, text generation, etc. One of the mechanisms of the model is an Transformer Encoder that reads the text input. The input to Transformer Encoder is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index. BERT can be used for a wide variety of language tasks, while only adding a small layer to the core model. Classification task is done by adding a "classification layer" on top of the Transformer output for the token.

We are going to download the model using a url, where we can find all the prebuilt and pretrained models developed in TensorFlow. We will use the official tokenization script created by the Google team that is upload on github.

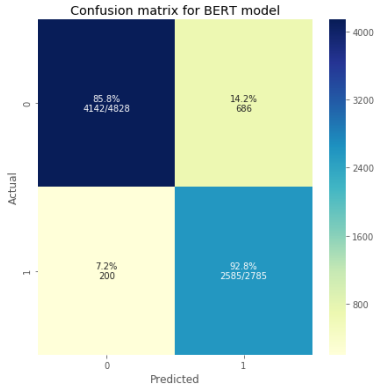
As a part of text cleaning we will be removing links and non-ASCII characters, emoji, punctuations and also convert abbreviations such as ppl, omg, fyi, etc.

Following is the logic of the code in the next few cells-

1. Load BERT model from the Tensorflow Hub(tfhub)
2. Load tokenizer from the bert layer
3. Encode the text into tokens, masks, and segment flags
4. Modify the output layer of the pre-trained BERT model as follows and train-
   input-text ===> Encoding for bert ==> BERT ===> Classifier(FeedForward-Network with 'softmax'-output-layer)
   Below is the pictorial representation of the architecture.



And the confusion matrix for BERT model



BERT by Google AI language and SVM

We prepared submission based on an ensemble of the predictions of the following four models:

- BERT pretrained word embeddings fed to a Dense Layer
- pretrained smaller dimensional word embeddings by Universal Sentence Encoder fed to SVM
- pretrained higher dimensional word embeddings by Universal Sentence Encoder fed to a MLP
- pretrained higher dimensional word embeddings by Universal Sentence Encoder fed to SVM

SVC with embeddings from Universal-Sentence-Encoder

While embedding a sentence, along with words the context of the whole sentence needs to be captured in that vector. This is where the "Universal Sentence Encoder" comes into the picture. If you recall the GloVe word embeddings vectors which turns a word to 50-dimensional vector, the Universal Sentence Encoder is much more powerful, and it is able to embed not only words but phrases and sentences. The **Universal Sentence Encoder (USE)** developed by researchers at **Google AI** encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks. The pre-trained Universal Sentence Encoder is publicly available in **Tensorflow-hub**. It comes with two variations i.e. one trained with Transformer encoder and other trained with Deep Averaging Network (DAN).

Following is the logic of the code in the next few cells-

1. Load USE model from the Tensorflow Hub(tfhub)
2. Create the embeddings for Train and Test
3. Feed the embeddings to Support Vector Classifier(SVC) model and train-
   input-text ===> Embeddings from USE ==> SupportVectorClassifier
   Below is the pictorial representation of the architecture.

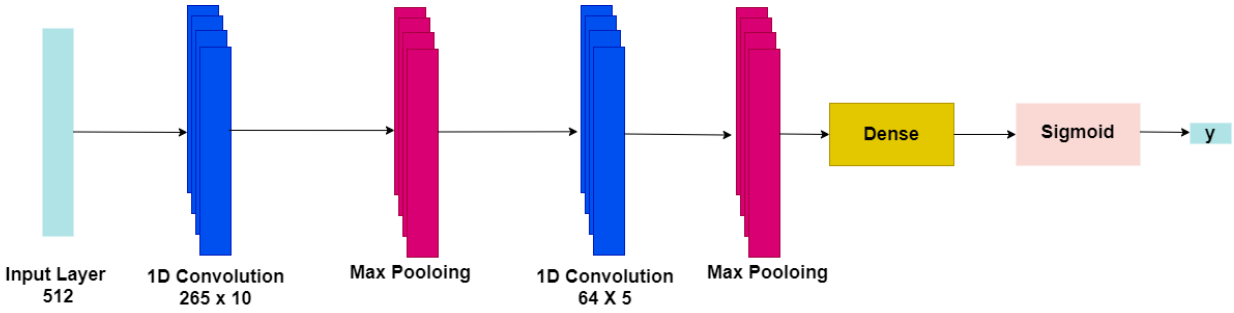
MLP with large dimensional embeddings from Universal-Sentence-Encoder

Following is the logic of the 3rd model-

1. Load USE model from the Tensorflow Hub(tfhub)
2. Create the higher dimensional embeddings for Train and Test

3. Feed the embeddings to a MultiLayerPercepton and train

Below is the pictorial representation of the architecture.



SVR with Large Dimensional Embeddings from Universal Sentence Encoder

Following is the logic of the 4th model-

1. Load USE model from the Tensorflow Hub(tfhub)
2. Create the higher dimensional embeddings for Train and Test
3. Feed the embeddings to a MultiLayerPercepton and train

Below is the pictorial representation of the architecture.

Weighted Voting of Predictions

Below are the weights for predictions from the above 4 models:

(0.5 Predictions from BERT model with Embeddings from BERT) +
(0.5 Predictions from SVM with Lower Dimensional Embeddings from Universal Sentence Encoder) +
(0.1 Predictions from Perceptron with Large Dimensional Embeddings from Universal Sentence Encoder) +
(0.3 Predictions from SVM with Higher Dimensional Embeddings from Universal Sentence Encoder)

6. CONCLUSIONS

- We have shown different Data Cleaning tasks related to text such as removing urls, non ascii characters, replacing abbreviations.
- We trained both classical machine learning models such as XGBoost and also Deep Learning based models with pre-trained word embeddings from GloVe.
- We also explored "BERT" a State of the art language model published by researchers at Google AI Language that outperformed all other models and pushed our Kaggle Leaderboard score to 0.84 whereas the highest score using classical ML approach was 0.43. That allowed us to be in 80 percentile in the competition with over 2 thousand team