

# Real or Not? NLP with Disaster Tweets

<https://www.kaggle.com/c/nlp-getting-started>

## Authors:

Chandan Mannem

Mahalavanya Sriram

Aparajitha Sriram

**Current date:** 12-21-2020

## ABSTRACT

With today's technology, each person's online footprint opens the door for a large treasure trove of information that can be used for many purposes that varies from analyzing market trends to understanding the general emotion of a group of people. Twitter data is especially very useful for a variety of purposes when it comes to the latter use case, mainly because there are more than 6000 new tweets every second. With the advancement of technology and Natural Language Processing methodologies, the process of text and sentiment analysis has become much easier than a few years earlier. If in case, a person tweets a message which was about an emergency or an impending disaster and this was recognized immediately by our NLP models, we would be able to react quicker than normal which would help save lives. This is pretty much the crux of our project. The main aim of our project is to distinguish if a tweet talks about a real disaster or not. This is for a competition hosted by Kaggle and the dataset provided consists of a training set of 10,000 hand classified tweets on which we built our models. For the purpose of identifying if a tweet is pertaining to a disaster or not, we tried out a variety of different models like bag of words, TF-IDF features, count vectorizer followed by a ridge classifier, a naive bayes approach, SVM and LSTM models, GloVe vectorization, BERT and k-fold cross validation on our BERT model. Out of all these, we found that the BERT model with a K-fold cross validation worked best for this dataset and gave us an f1 score of 0.83573.

## 1. INTRODUCTION

The main aim of our project is to distinguish if a tweet talks about a real disaster or not. This is for a competition hosted by Kaggle [5] and the dataset provided consists of a training set of 10,000 hand classified tweets on which we built our models. For the purpose of identifying if a tweet is pertaining to a disaster or not, we tried out a variety of different models like bag of words, TF-IDF features, count vectorizer followed by a ridge classifier, SVM and LSTM models, GloVe vectorization, BERT and k-fold

cross validation on our BERT model. Out of all these, we found that the BERT model with a K-fold cross validation worked best for this dataset and gave us an f1 score of 0.83573.

We first started with data cleaning which included tokenization, removal of stop-words followed by stemming and lemmatization. Following this we made a few visualizations such as Bar plots and Word cloud. We also tried the bag of words approach for better understanding of our dataset followed by count vectorization. Then we built a baseline model using a count vectorizer followed by a ridge classifier and cross validation. This yielded a score of 0.78026. Then we tried a naive bayes approach using a Countvectorizer and a TfidfTransformer which yielded an F1 score of 0.79344.

Following this we worked on a SVM (Support Vector Machine) model and an LSTM (Long short-term memory) model which gave us an F1 score of 0.79528. After a detailed error analysis from the LSTM output, we made a few changes to the existing data processing steps and implemented a BERT (Bidirectional Encoder Representations from Transformers) model, as the existing bidirectional LSTM did not improve the accuracy even after fine tuning. This increased our F1 score drastically to 0.83573. Then we tried to improve the BERT model by using GloVe vectorization and then K-fold cross validation. However, our best score stays at 0.83573.

## **2. TEXT CLASSIFICATION WITH NLP**

The problem at hand deals with classifying tweets if they are related to an actual disaster or not. So, this problem is basically a text classification problem that might require sentiment analysis as well. One research that is very close to our problem is the research [1] “Safety Information Mining — What can NLP do in a disaster” by Graham Neubig, Yuichiroh Matsubayashi, Masato Hagiwara and Koji Murakami. Their research used NLP to aid relief efforts during the 2011 East Japan Earthquake. Their aim was to create an effective system for word segmentation, named entity recognition and tweet classification. They used a default SVM solver of LIBLINEAR with a 10-fold cross validation on the test set. This gave them an accuracy of 86.13%.

A very common problem that deals with text classification and is very close to our problem is the classification of tweets if they are spam or not. One research that deals with this problem is [2] which used a Support Vector Machine model that got a 95-97% accuracy. Another research paper [3] that deals with the problem of detecting sarcasm in customer tweets used a naive bayes approach that gave a very good accuracy.

## **3. TWEETS ABOUT REAL DISASTERS**

### **About the Dataset**

The dataset contains around 10,000 tweets that were hand classified. They have been divided into a training and a test set. This dataset was created by the company figure-eight and originally shared on

their 'Data For Everyone' website [4]. The training set contains a target column identifying whether the tweet pertains to a real disaster or not.

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

```
['Just happened a terrible car crash', 'Heard about #earthquake is different cities, stay safe everyone.', 'there is a forest fire at spot pond, geese are fleeing across the street, I cannot save them all', 'Apocalypse lighting. #Spokane #wildfires', 'Typhoon Soudelor kills 28 in China and Taiwan', 'We're shaking...It's an earthquake', 'They'd probably still show more life than Arsenal did yesterday, eh? EH?', 'Hey! How are you?', 'What a nice hat?', 'Fuck off!']
```

Our job is to create a ML model to predict whether the test set tweets belong to a disaster or not, in the form of 1 or 0. This is a classic case of a Binary Classification problem.

The dataset used contains

1. train.csv that has instances (texts from Twitter) 7,613 rows and attributes
  - id: a unique identifier of user
  - keyword: a particular word from the tweet
  - location: the location the tweet was sent from
  - text: the text of the tweet
  - target: this denotes whether a tweet is about a real disaster (1) or not (0)
2. test.csv that has instances (texts from Twitter) 3,263 rows and attributes
  - id: a unique identifier of user
  - keyword: a particular word from the tweet
  - location: the location the tweet was sent from
  - text: the text of the tweet

3. `submission.csv`: This will collect the id and target that indicates whether the message at the id is a disaster-related message or not.

In keywords and location columns, there are a lot of missed values, but we don't need to be bothered with them. We will need to clean up noises in the text feature to make our life simpler. We have actually changed the NaNs to `no_keyword` and `no_location` to avoid missing value errors. Both training and test sets have the 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

## 4. OUR APPROACH AND EXPERIMENTS

### 4.1. Motivating your approach

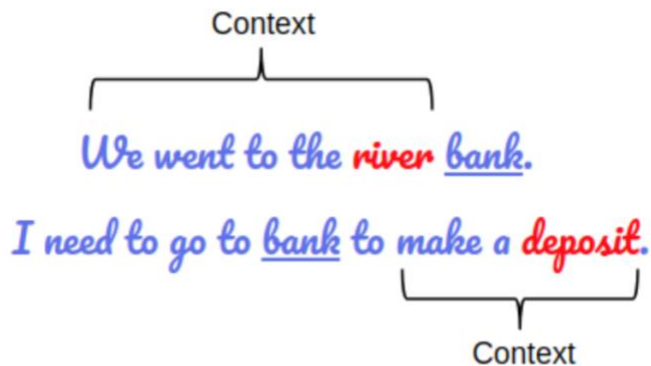
It is important to filter out noisy information from the vast volume of data flow in order to efficiently exploit social media data so that these data will help predict catastrophe damage. Many researchers turn to machine learning for the solution, not happy with simple keyword-based filtration. In this project, We use deep learning techniques to solve the issue of classification of tweets in the field of disaster management. Tweet labels represent various forms of disaster-related details that have different potential applications for emergency response purposes. For transfer learning, in particular, BERT is used. Compared to the baseline bidirectional LSTM with pre-trained Glove Twitter embedding, the standard BERT classification architecture and many other specialized BERT architectures are trained. The findings reveal that the best results are obtained by BERT, outperforming the baseline model in terms of F-1 score by 3.29 percent on average. Ambiguity and subjectivity affect the performance of these models considerably.

### **BERT (Bidirectional Encoder Representations from Transformers)**

BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications.

BERT is a deeply bidirectional model. Bidirectional means that BERT learns information from both the left and the right side of a token's context during the training phase.

The bidirectionality of a model is important for truly understanding the meaning of a language. For example, if there are two sentences in this example and both of them involve the word "bank":



If we try to predict the nature of the word “bank” by only taking either the left or the right context, then we will be making an error in at least one of the two given examples. One way to deal with this is to consider both the left and the right context before making a prediction. That’s exactly what BERT does.

## BERT’s Architecture

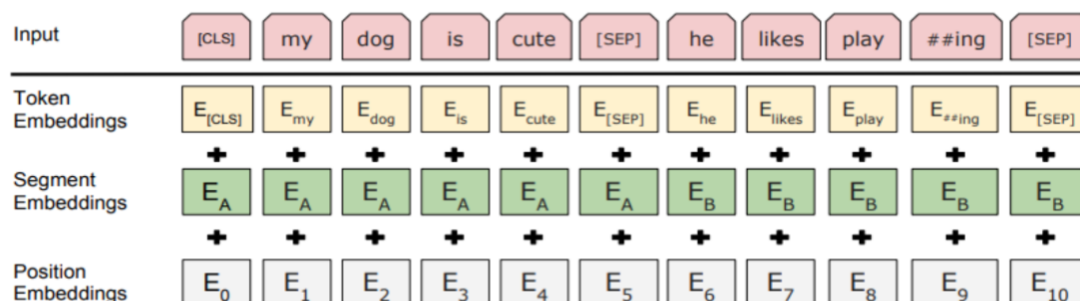
The BERT architecture builds on top of the Transformer. We currently have two variants available:

- BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
- BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters

All of these Transformer layers are Encoder-only blocks.

Now that we know the overall architecture of BERT, let’s see what kind of text processing steps are required before we get to the model building phase.

## Text processing for BERT



For Bert, every input embedding is a combination of 3 embeddings:

**Position Embeddings:** BERT learns and uses positional embeddings to express the position of words in a sentence. These are added to overcome the limitation of Transformer which, unlike an RNN, is not able to capture “sequence” or “order” information

**Segment Embeddings:** BERT can also take sentence pairs as inputs for tasks (Question-Answering). That’s why it learns a unique embedding for the first and the second sentences to help the model distinguish between them. In the above example, all the tokens marked as EA belong to sentence A (and similarly for EB)

**Token Embeddings:** These are the embeddings learned for the specific token from the WordPiece token vocabulary

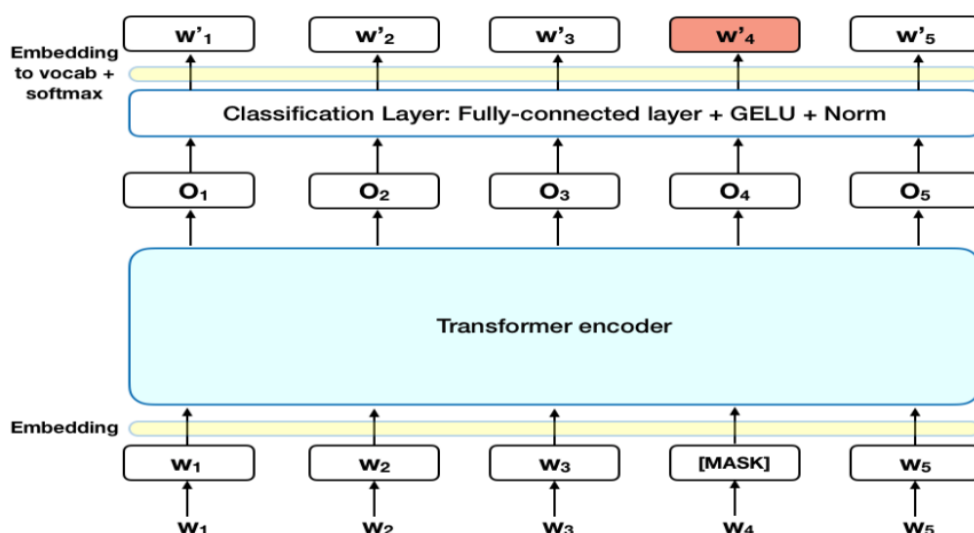
For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. Such a comprehensive embedding scheme contains a lot of useful information for the model. These combinations of preprocessing steps make BERT so versatile.

When training language models, there is a challenge of defining a prediction goal. Many models predict the next word in a sequence, a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:

## Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with softmax.



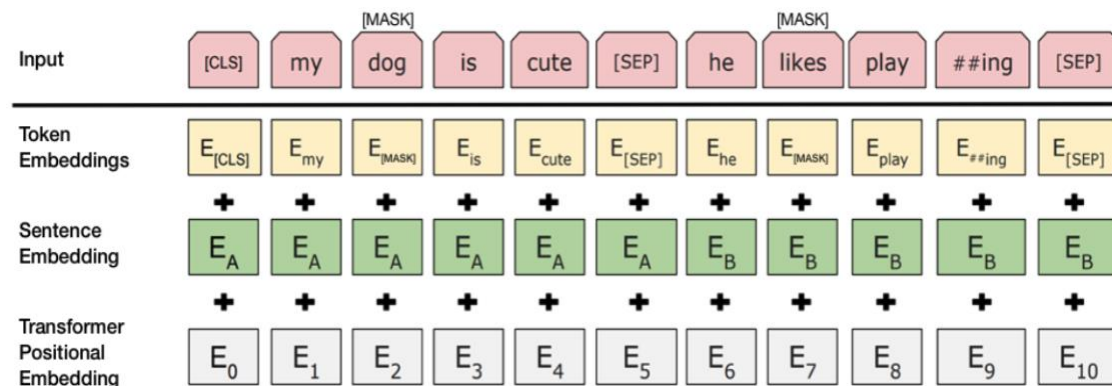
The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

## Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.



To predict if the second sentence is indeed connected to the first, the following steps are performed:

1. The entire input sequence goes through the Transformer model.
2. The output of the [CLS] token is transformed into a  $2 \times 1$  shaped vector, using a simple classification layer (learned matrices of weights and biases).
3. Calculating the probability of IsNextSequence with softmax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

## **4.2. Our Methods**

### **4.2.1. Data Cleaning and Pre-Processing**

Before we start, we pre-processed the data to get it all in a consistent format. We cleaned, tokenized and converted our data into a matrix. Some of the basic text pre-processing techniques includes:

Make text all lowercase or uppercase so that the algorithm does not treat the same words in different cases as different.

Removing Noise i.e., everything that isn't in a standard number or letter i.e. Punctuation, Numerical values, common nonsensical text (like /n)

#### **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.

#### **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 “Stigmatization”, the result would be a single word “stem”.

#### **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.

#### **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 that we actually want. 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. We used various tokenizer and word embeddings such as count vectors, TF-IDF vectorization, Continuous Bag of words, GloVe, Fast text.

The next step is to normalize the count matrix using tf-idf representation. The main reason behind normalizing frequencies instead of using the raw ones is to decrease the effect of tokens that occur several times in the text, and there's not as informative as those presented a few times. For instance, the word



“document” occurs a thousand times in a given corpus, and “awesome” happens two times. tf-idf works in this situation, like preprocessing data to change raw feature vectors into a representation that is more suitable for machine learning algorithms.

### Transforming tokens to a vector

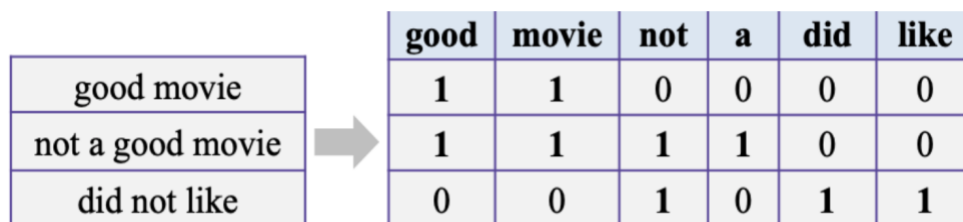
After the initial preprocessing phase, we need to transform text into a meaningful vector (or array) of numbers. This can be done by a number of techniques:

#### 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 called a “bag” of words? That is 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.



	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1

We can do this using scikit-learn's CountVectorizer, where every row will represent a different tweet and every column will represent a different word.

#### Bag of Words - Countvectorizer Features

Countvectorizer converts a collection of text documents to a matrix of token counts. It is important to note here that CountVectorizer comes with a lot of options to automatically do preprocessing, tokenization, and stop word removal. However, all the processes were done manually to just get a better understanding.

#### TF-IDF 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”. Also, it will give more weight to longer documents than shorter documents.

One approach is to rescale 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.

$TF = (\text{Number of times term } t \text{ appears in a document}) / (\text{Number of terms in the document})$

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.

## **Text Embedding**

Fine-grained classification was much harder. Lexical features (bigrams and key terms) were useful for most categories, with other features providing minor benefits. Word embeddings greatly improved performance across all categories, while most features had mixed results. We have used a few pre-trained embeddings, hence doing standard preprocessing steps might not be a good idea because some of the valuable information can be lost. It is better to get vocabulary as close to embeddings as possible. In order to do that, we train vocab and test vocab are created by counting the words in tweets.

Text cleaning is based on the embeddings below:

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

## **Uni- Grams and Bi- Grams**

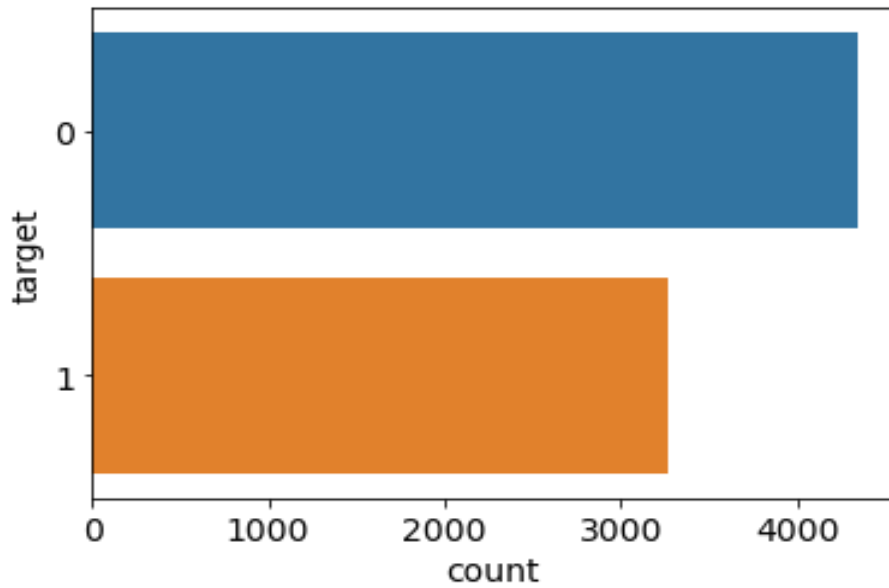
We also created bi-grams and trigrams. To find out what a word's most common neighbors are, it is useful to make a list of bigrams (or trigrams, 4-grams, etc.)-i.e. to list every cluster of two (or three, four, etc.) consecutive words in a text.

For most of the above operations we have used NLTK and SpaCy.

## **4.2.2. Data Visualizations**

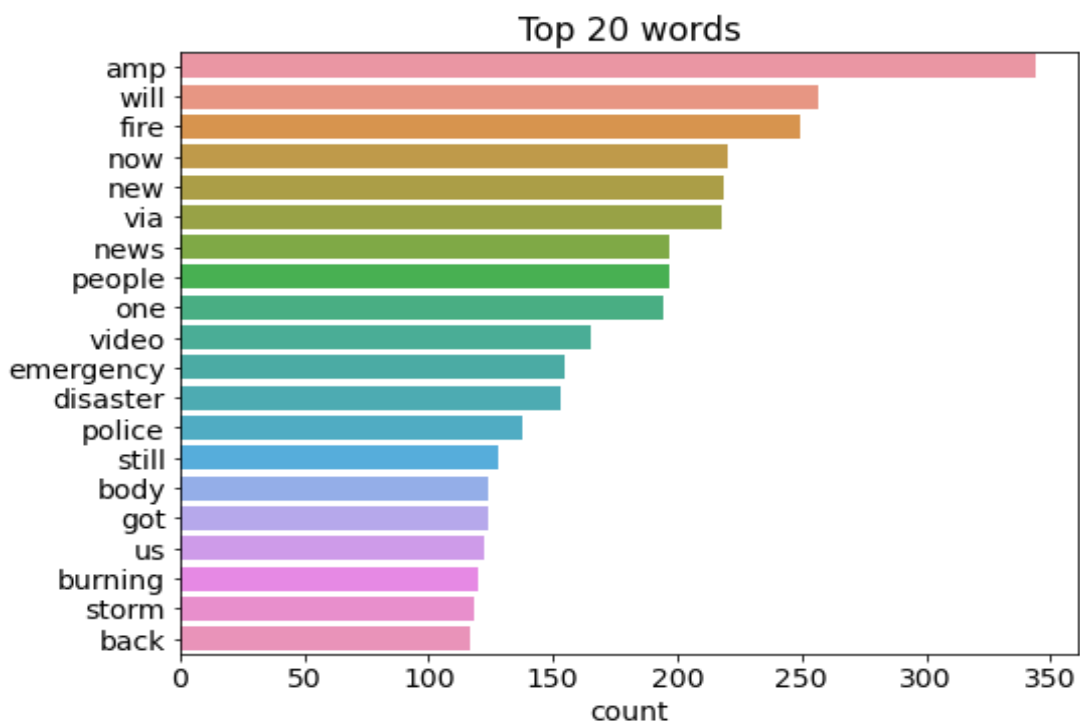
Few visualizations from our EDA mentioned below

## **Target Distribution**



There are more tweets with class 0 (No disaster) than class 1 ( disaster tweets)

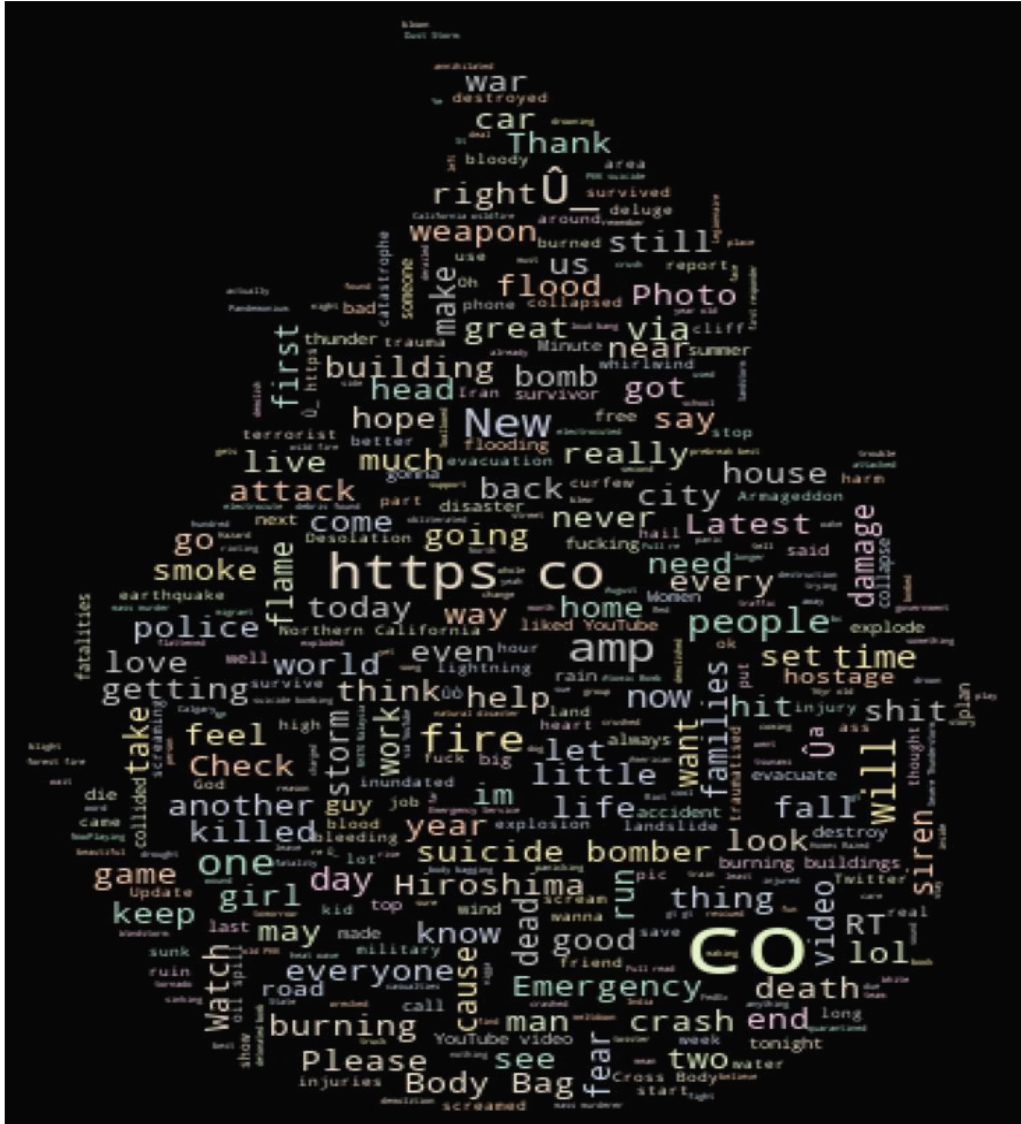
### Top 20 Keywords



## Word Cloud

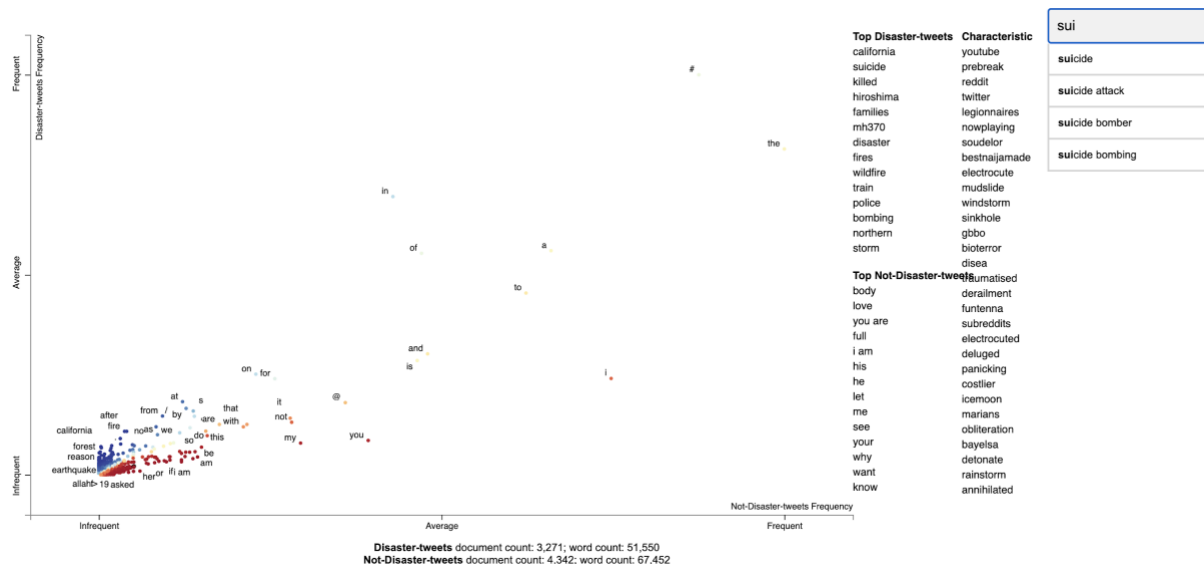
Analyzed the data by generating simple statistics such word frequencies over the different authors as well as plotting some word clouds (with image masks).

Words before cleaning



## ScatterText

A tool for finding distinguishing terms in corpora, and presenting them in an interactive, HTML scatter plot. Points corresponding to terms are selectively labeled so that they don't overlap with other labels or points. This is a tool that's intended for visualizing what words and phrases are more characteristic of a category than others.



Looking at this seems overwhelming. Each dot corresponds to a word or phrase mentioned by Tweets. The closer a dot is to the top of the plot, the more frequently it was used by Disaster tweets. The further right a dot, the more that word or phrase was used by not disaster tweets. Words frequently used by both the category tweets, like "in" and "of" tend to occur in the upper-right-hand corner. Although very low frequency words have been hidden to preserve computing resources, a word that neither tweets use, like "giraffe" would be in the bottom-left-hand corner.

The interesting things happen close to the upper-left and lower-right corners. In the upper-left corner, words like "fires" and "storm" are frequently used by Disaster tweets but infrequently or never used by non-disaster tweets. Likewise, terms frequently used by non-disaster tweets and infrequently by disaster occupy the bottom-right corner. These include "think" and "YouTube". Terms are colored by their association. Those that are more associated with Disaster are blue, and those more associated with Non disaster red.

The positions shown correspond to the dense ranks of frequencies in a particular class. Means that the first most frequently used term in a particular category is plotted next to the second most frequently used term, even if one term is used many times more than its neighbor. This prevents large gaps from appearing in the plot and makes it more readable as a whole.

Finally, an algorithm unique to Scatter Text is used to determine which points are labeled and which points aren't. This labeling happens client-side, and users can interactively see which terms correspond to which points if they mouse-over them. Clicking reveals snippets from documents showing how each term was used in context.

Terms (only unigrams for now) that are most characteristic of the both sets of documents are displayed on the far-right of the visualization.

## Topic Generation

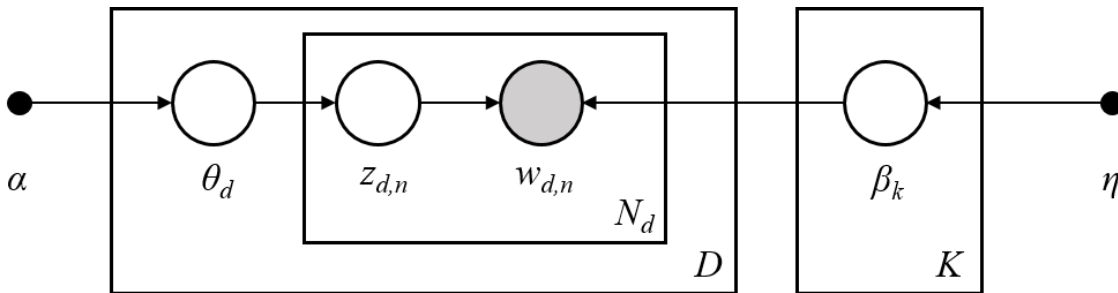
In LDA, the modelling process revolves around three things: the text corpus, its collection of documents,  $D$  and the words  $W$  in the documents. Therefore, the algorithm attempts to uncover  $K$  topics from this corpus via the following way (illustrated by the diagram)

Three Level Bayesian Model

Model each topic,  $k$  via a Dirichlet prior distribution given by  $\beta_k$ :

Model each document  $d$  by another Dirichlet distribution parameterized by  $\alpha$  :

Subsequently for document  $d$ , we generate a topic via a multinomial distribution which we then backtrack and use to generate the corresponding words related to that topic via another multinomial distribution:



(Image source: <http://scikit-learn.org/stable/modules/decomposition.html#latentdirichletallocation>)

The LDA algorithm first models document via a mixture model of topics. From these topics, words are then assigned weights based on the probability distribution of these topics. It is this probabilistic assignment over words that allow a user of LDA to say how likely a particular word falls into a topic. Subsequently from the collection of words assigned to a particular topic, are we thus able to gain an insight as to what that topic may actually represent from a lexical point of view.

From a standard LDA model, there are really a few key parameters that we have to keep in mind and consider programmatically tuning before we invoke the model:

$n\_components$ : The number of topics that you specify to the model

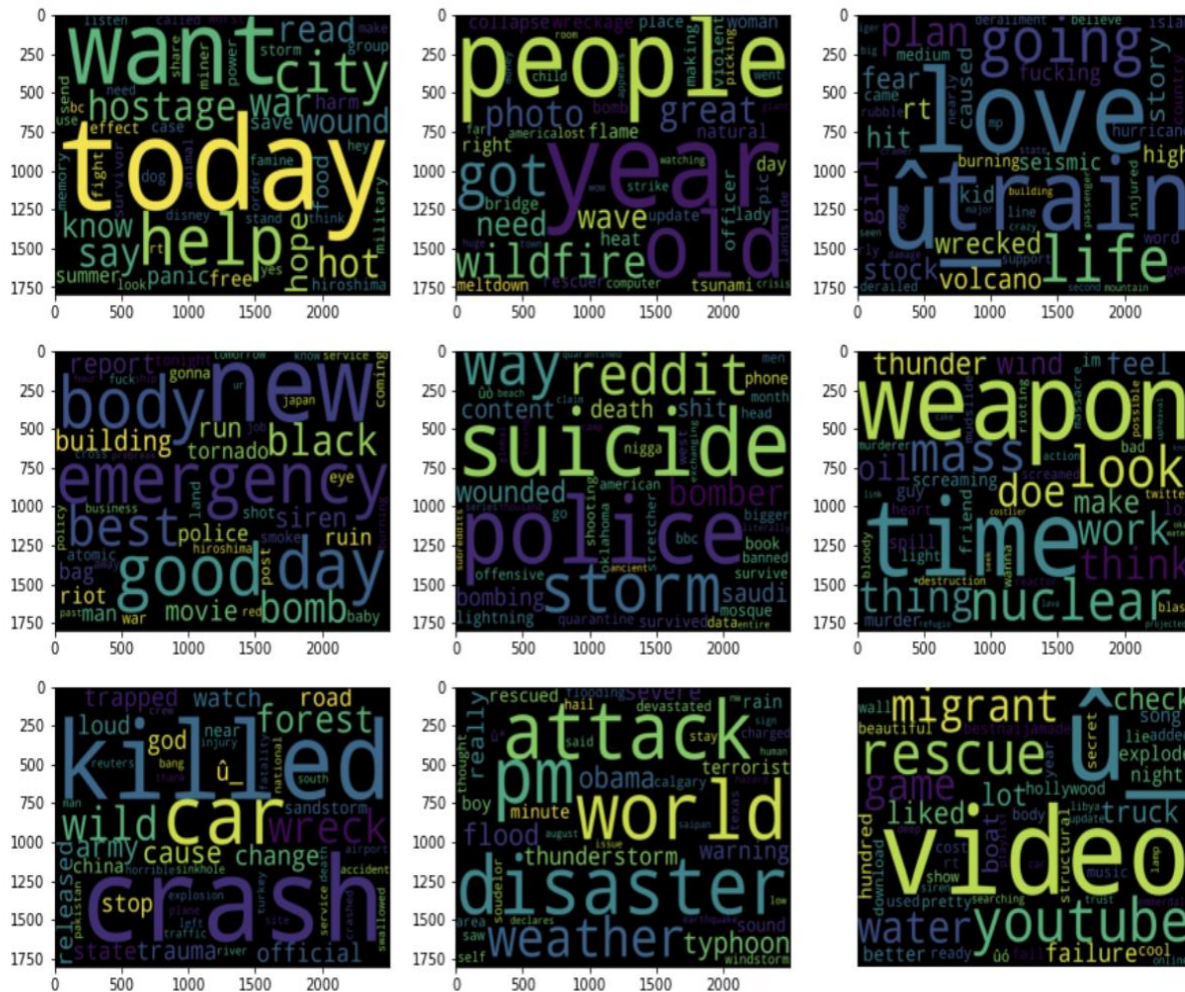
$\alpha$  parameter: This is the dirichlet parameter that can be linked to the document topic prior

$\beta$  parameter: This is the dirichlet parameter linked to the topic word prior



To invoke the algorithm, we simply create an LDA instance through Sklearn. LatentDirichletAllocation function. The various parameters would ideally have been obtained through some sort of validation scheme. In this instance, the optimal value of n\_components (or topic number) were found by conducting a KMeans + Latent Semantic Analysis Scheme (as shown in this paper here) whereby the number of K Means clusters and number of LSA dimensions were iterated through and the best silhouette mean score.

Then we pulled the top 10 topics and constructed the word cloud for the same



### 4.2.3. Models implemented

The following are the models we implemented on our dataset.

## **Simple Linear Classification - Logistic Regression**

Logistic Regression is one of many machine learning methods that works by taking input and multiplying the input value with weight value. It is a classifier that learns what features from the input that are the most useful to discriminate between the different possible classes.

## **Multinomial naïve-bayes**

To begin our analysis, we considered Naive Bayes as an initial model, a popular classifier for unstructured text with proven ability in the field.

Naive Bayes performs well with a small amount of data and uses prior knowledge to calculate a posterior probability, represented by a probability distribution which reflects the likelihood that a specific instance belongs to a particular class. With our data prepared we can instantiate a Naive Bayes classifier by invoking an SKLearn class, initially maintaining default parameters to establish a baseline performance. We will use this baseline performance as a means of deciding whether to further explore the parameter space of Naive Bayes or consider a different model for our problem.

## **Support Vector Machine**

An SVM is a kind of large-margin classifier: it is a vector space-based machine learning method where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data (possibly discounting some points as outliers or noise). We have used LinearSVC from Sklearn

## **Bidirectional LSTM**

The Long-Short Term Memory (LSTM) is a specialized version of Recurrent Neural Network (RNN) that is capable of learning long term dependencies. While LSTM can only see and learn from past input data, Bidirectional LSTM (BLSTM) runs input in both forward and backward direction. We use such a model to evaluate the classification.

## **BERT (Bidirectional Encoder Representations from Transformers)**

This model utilizes the implementation of BERT at TensorFlow/models/official/nlp/bert from the TensorFlow Models repository on GitHub. We use L=12 hidden layers, a hidden size of H=768, and A=12 focus heads (Transformer blocks). On Wikipedia and Books Corpus, this model was pre-trained for English. Inputs have been "uncased" indicating that before tokenization into word bits, the text has been lower-cased, and all accent marks have been deleted. To download this model, the kernel needs to be activated on the Internet.



## BERT + K- Fold Cross Validation (StratifiedKFold)

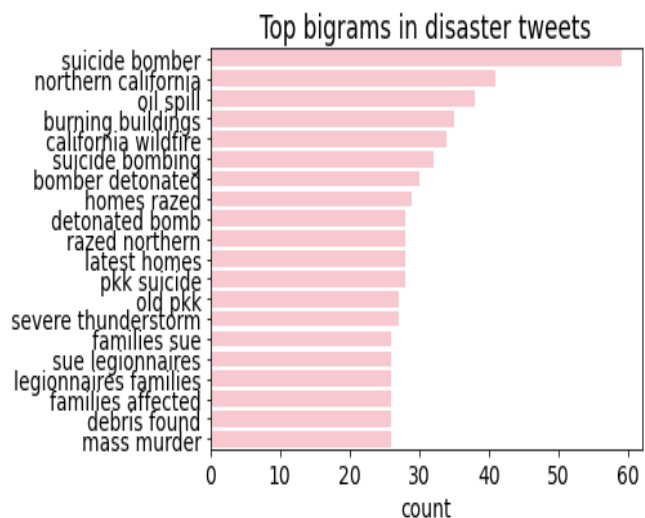
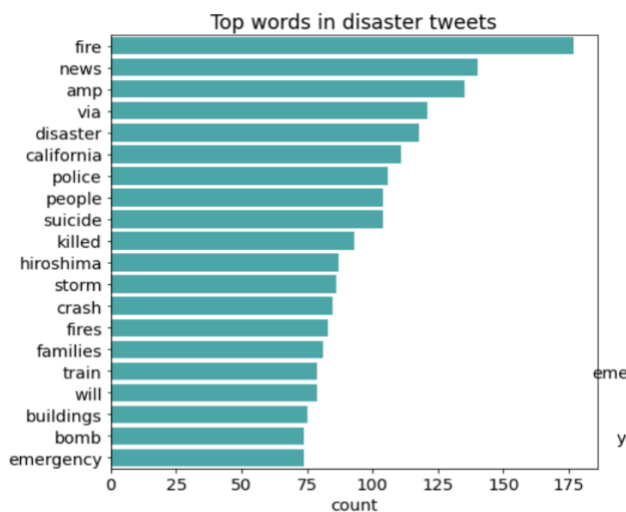
First of all, it can be shown that the training and test set are divided into keyword classes while the training/test sets are concatenated, and tweet counts are calculated by keyword. By looking at the ID function, we may also come to that conclusion. This implies that all keywords are stratified during training and test set formation. For cross-validation, we can duplicate the same break.

In both the training and test collection, tweets from any keyword group exist and are from the same sample. StratifiedKFold is used to reproduce the same split technique, and keyword is transferred as y, so stratification is performed depending on the keyword function. For additional training diversity, shuffle is set at Real. In the training and confirmation packages, all directories have tweets from any keyword group that can be seen from below.

### 4.3. Details of our experiments

#### Findings after Data Pre-processing and Visualization

We observed the most frequent words in unigrams and bigrams were different. Bigram seems more meaningful than unigram for most frequent words in disaster tweets.



We also observed that http links are also playing an important part in tweets for disaster analysis. Based on the set of links we can say that this tweet is related to disaster.

Also, hashtags are also important as they provide a valuable amount of information about the nature of the tweet. In some cases, the hashtags themselves are enough to classify the tweet, like

#disaster #covid #nature #disasterrelief #earthquake #tsunami #bencana #naturaldisaster #emergency #gempabumi #fire #disasterpreparedness #flood #coronavirus #disasterresponse #earth #disastermanagement #disasters #help #hurricane #disasterrecovery #chernobyl #rescue

Removal of NLP Stop Words mislead the prediction. It was not necessary for our use case as it changed the context of the tweets.

For example: This is not a cyclone; This is a cyclone in its starting stages.

If the stop word “not” in the first sentence is removed, the context of the tweet changes and could be classified as a disaster-related tweet like the second tweet.

## Findings from Embeddings

Both GloVe and FastText embeddings have more than 50% vocabulary and 80% text coverage without cleaning. GloVe and FastText coverage are very close but GloVe has slightly higher coverage.

## Observations from error analysis

After some of the initial models we implemented, we performed a detailed error analysis, which lead to the following conclusions:

- Ambiguity and subjectivity play an important role.

*Example: - 'like for the music video I want some real action shit like burning buildings and police chases not some weak ben winston shit' this tweet is labelled as disaster*

- There are only about 7000 data records for training which seems low and hence accuracy of the model suffers.
- Some need more contexts to fully understand, e.g. html

*Example: - 'RT NotExplained: The only known image of infamous hijacker D.B. Cooper. <http://t.co/JlzK2HdeTG>' this tweet is labelled as not a disaster. but the link mentioned in this tweet is*

- Non-ascii words and Emoji missing.

*Example: Now everyone is religious Ã83Ã83Ã82ÂiÃ83Ã82Ã82Ã83Ã82Ã82Ã4 #prayforblahblah #harvey #irma #maria #mexico #hurricane #earthquake <https://t.co/nzEbRWQG8W> - This text is labeled as disaster. The Emoji "face with rolling eyes" is not proper represented in the original text data*

- Semantic misconstrue (Sarcastic, metaphoric...)

*Example: Mmmmmm I'm burning.... I'm burning buildings I'm building.... Oooooohhhh oooh ooh... - This is classified as disaster*

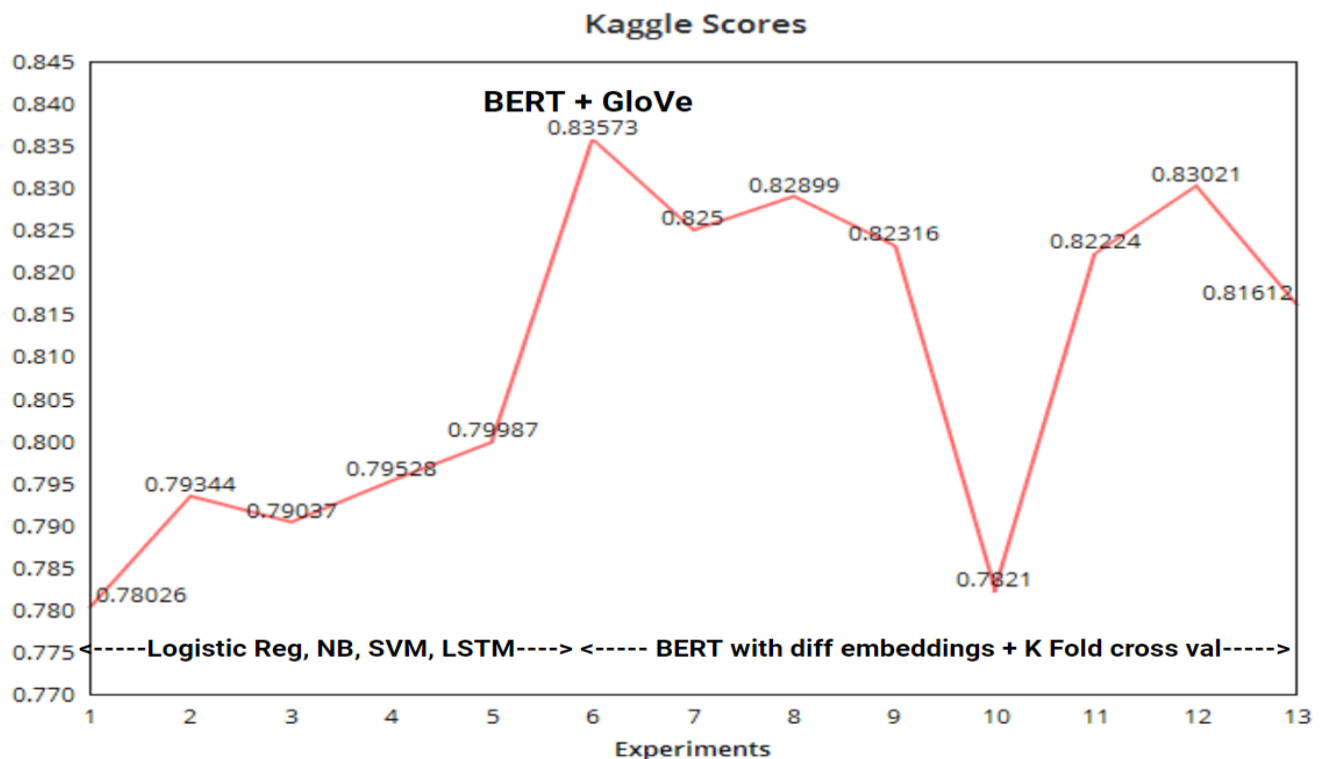
- Keyword influence or Misleading hashtag (#)

*Example: 'Who is bringing the tornadoes and floods. Who is bringing the climate change? God is after America He is plaguing her\n \n#FARRAKHAN #QUOTE' - This is a actual disaster tweet but misclassified due to the presence of misleading hashtag*

- A lot of misclassified messages are news, which is less useful than Tweets from disaster eyewitnesses and the usage of which is debatable
- Some events may not really happen
- Message is too short

## Results

Over the time of our experiments, our F1 score varied a lot based on the models we used.



Our highest score has been 0.83573 which we got when we implemented BERT with GloVe embedding. We then experimented with different BERT models by altering our hyper parameters, but we were not able to get a result higher than that. Following were the scores we got through our various experiments:

Model	F1 Score	Our Experiments
Linear classification model with cross validation	0.78026	This is our baseline model.
A multinomial naïve-bayes approach	0.79344	The naïve bayes approach worked but it did not improve our scores from before.
LSTM approach and SVM	0.79528	The lstm approach was a neural network approach which we tried using keras after detailed text cleaning. But even then, the model accuracy did not change much.
Bidirectional Encoder Representations from Transformers (BERT)	0.79987	After a detailed error analysis from the LSTM output, we made a few changes to the existing data processing steps and implemented a BERT as existing bidirectional LSTM even after fine tuning the model did not improve the accuracy.
BERT Along with GloVe vectorization and Logistic Regression	0.83573	We had a detailed study of the most frequent words in unigrams and bigrams are different. Bigram seems more meaningful than unigram for most frequent words in disaster tweets. From this we felt GloVe vectorization would be better than other text Embeddings.
Implemented k- fold cross validation on BERT and tried use of various activation functions. (also experimented on epochs, max_sequence length	0.83021	Implemented stratified k fold cross validation along with BERT. Also implemented some visualization for the BERT result.

## 5. DISCUSSION AND RELATED WORK

We found BERT to give a better score when combined with GloVe vectorization in our experiments. The reason we chose BERT was because it is state-of-the-art and used in many NLP tasks and it is open-sourced. An approach very similar to ours was used by Guoqin Ma in [6]. In this research paper, they had explored the problem of tweets classification with BERT in the field of disaster management. They had started with a baseline model of bidirectional LSTM with GloVe Twitter embeddings. Then, they proceeded to try different BERT models, out of which a BERT-based LSTM model achieved the best performance [6]. They got a maximum accuracy of 0.67 for their best model of BERT.

The BERT model from the research by Devlin et al. [7] gave an accuracy of about 92%, which seems to be the best score for BERT so far. Another important research done by Alam et al. [8] provided a lot of insights on the various machine learning algorithms that can be used to analyze the data generated by disaster events. This work provides an extensive analysis of Twitter data collected during the three disasters, namely Hurricanes Harvey, Irma, and Maria. The results and analysis of this work give a much-detailed view of how the research in this field would be useful in disaster management.

The research by Imran et al. [9] presents a human-annotated Twitter corpora of 19 different crises that took place between 2013 and 2015. This paper published the first largest word2vec word embeddings trained on 52 million crisis-related tweets. In the research [10], Spiliopoulou et al., train an adversarial neural model to remove latent event-specific biases and improve the performance on tweet importance classification. In this research, the adversarial model using BERT embeddings obtained an AUC of 0.62 for the critical class.

## 6. CONCLUSIONS AND OPEN PROBLEMS

From our various experiments with the Twitter dataset on disaster tweets, we found BERT gave the best accuracy. However, this was without taking into consideration the topic of sentiment analysis. Emotion analysis and sentiment analysis is a whole big topic that could be explored to great depths in a separate project of its own. Being new to the field of Natural Language Processing, this work was enlightening and helped us to experiment with a lot of different models which we came across. This problem of classifying tweets can be further improved by bringing into it the element of sentiment analysis and that can be left open for more research in the future. Altogether, the experience of working on this competition on Kaggle has been very knowledgeable.

*Our data is available at:* <https://www.kaggle.com/c/nlp-getting-started/data>

*Our code is available at:*

[https://github.com/ajithasriram/NLP\\_with\\_disaster\\_tweets.git](https://github.com/ajithasriram/NLP_with_disaster_tweets.git)

## 7. CONTRIBUTIONS AND CONFLICT OF INTEREST DECLARATIONS

Our team consists of Mahalavanya Sriram, Chandan Mannem and Aparajitha Sriram. We split our tasks and executed them accordingly. All the three of us took part in the data preprocessing phase equally. The detailed EDA and the LSTM and SVM models were worked on by Mahalavanya. The baseline cross validation model and the Naive Bayes were worked on by Aparajitha. The logistic regression and error validation parts were handled by Chandan. The BERT model, the pre analysis and the K-fold implementation on BERT were handled by all of us together. Apart from this, each of us took an active part in the models implemented by each other and helped in the error analysis phase and the visualization phase.

## 8. REFERENCES

- [1] Neubig, Graham, et al. "Safety Information Mining—What can NLP do in a disaster—." *Proceedings of 5th International Joint Conference on Natural Language Processing*. 2011.
- [2] S. Gharge and M. Chavan, "An integrated approach for malicious tweets detection using NLP," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2017, pp. 435-438, doi: 10.1109/ICICCT.2017.7975235.
- [3] Mukherjee, Shubhadeep, and Pradip Kumar Bala. "Detecting sarcasm in customer tweets: an NLP based approach." *Industrial Management & Data Systems* (2017).
- [4] Appen AI Resource Center. [Online]. Available: <https://appen.com/resources/datasets/>
- [5] Natural Language Processing with Disaster Tweets. [Online]. Available: <https://www.kaggle.com/c/nlp-getting-started>
- [6] Ma, Guoqin. "Tweets Classification with BERT in the Field of Disaster Management."
- [7] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
- [8] Alam, Firoj, et al. "A twitter tale of three hurricanes: Harvey, irma, and maria." *arXiv preprint arXiv:1805.05144* (2018).
- [9] Imran, Muhammad, Prasenjit Mitra, and Carlos Castillo. "Twitter as a lifeline: Human-annotated twitter corpora for NLP of crisis-related messages." *arXiv preprint arXiv:1605.05894* (2016).
- [10] Spiliopoulou, Evangelia, et al. "Event-Related Bias Removal for Real-time Disaster Events." *arXiv preprint arXiv:2011.00681* (2020).
- [11] Kaggle, Natural Language Processing with Disaster Tweets. [Online]. Available: <https://www.kaggle.com/c/nlp-getting-started/notebooks>.