

---

# Real or Not? NLP with Disaster Tweets: Project Report for CSc8850 Advanced Machine Learning



**SHIBA-INU team**

---

Kiril Kuzmin, Xumin Cai, Chenyu Wang,  
Chan Aek Panichvatana, Pragna Reddy Kancharla,  
Krishna Mani Teja Katragadda

Department of Computer Science  
Georgia State University  
Atlanta, GA 30303

## **Abstract**

This project is a participant of the Kaggle competition, which challenge was to predict whether or not a tweet is talking about a natural disaster. Our approach includes tweets preprocessing and cleaning; mapping tweets into vectors and classifying them; making post-predictions based on the keywords information of the tweets. Four models have been implemented and analyzed: TF-IDF with Logistic Regression and SVM classifiers, LSTM GloVe, BERT, and Word2Vec based on the google-300-negative pretrained set with SVM classifier. BERT turned out to be the best performing model reaching public score of 84.25% of Kaggle leaderboard.

## **1 Introduction**

Natural language processing (NLP) is a part of text mining that performs a specific type textual analysis that allows a computer to interpret text. The aim of NLP is to interpret, formalize, or “understand” natural language in order to conduct different human activities such as language processing or responding to questions [1]. There are numerous NLP applications [2] including spell checking, keyword search, finding synonyms/antonyms, parsing information from websites/documents, machine translation (from one language to another), semantic analysis, automatic question answering, etc.

The best instructor: Dr. Sergey Plis

Modern social media provide great sources of data for many NLP applications. Twitter, one of microblogs where users interact with short messages known as “tweets”, is one of them. Among many other things, it allows conveniently collecting information about real-time processes in an inexpensive, safe, and quick manner. The applications here are diverse, starting with raising “situational awareness” (e.g., monitoring how a natural disaster or accidents are developing in real time), predicting trends in post-catastrophic processes (e.g., the spread of diseases that followed the disaster), and ending with predicting which candidate is more likely to win an election [3].

Such NLP problems are often raised on the kaggle platform in form of competitions. A recent one, started in January 2020, is called “Real or Not? NLP with Disaster Tweets on Kaggle” [4]. This competition aims to build a machine learning tool that discerns if a tweet really talks about a natural disaster or not. The competition opens with an example of a tweet “On plus side LOOK AT THE SKY LAST NIGHT IT WAS ABLAZE” published by a user named Anna [5]. This example transparently illustrates one of the challenges in the problem: some words could be used in metaphorical sense, like a word “ablaze” here. Obviously, there are many other challenges on the way including misspellings, slang, short average length of a single tweet, and large variety of topics discussed on twitter [6]. To approach this challenge, in the project, we use modern NLP developments as LSTM (Long Short Term Memory) recurrent neural networks, words embedding with Word2Vec, and BERT (Bidirectional Encoder Representations from Transformers).

The report is organized as follows. Section 2 Data Processing introduces the data cleaning technique used in our project. Section 3 Methods shows the exploration of the models applied to the task. Our experimental evaluation results are reported in Section 4 Experiments and Results. We conclude the report with the discussion in Section 5.

## 2 Data Processing

The data we use is taken from [4], which contains 7613 training data sets and 3263 test data sets. Each of the tweets provides four parameters, namely, *id*, *keyword*, *location*, *text*. In the training data sets, there are 3271 out of 7613 ( $\sim 48\%$ ) of tweets labeled as 1, which indicate the real disaster and 4342 out of 7613 ( $\sim 52\%$ ) of tweets labeled as 0 which indicate the not real disaster. In this project, we use *text* as our raw data input. In Fig. 1, we explore the top 20 frequent occurrence words in the raw training data. Fig. 1a shows the top 20 words in tweets which indicate real disaster and Fig. 1b presents the top 20 words in tweets which indicate not real disaster. We can see that some frequent words in a real disaster such as ‘suicide’, ‘bomb’, ‘Hiroshima’ distinguish from the words in not real disaster. Moreover, some words such as ‘like’, ‘via’, ‘amp’ show the high frequency in both groups. We explore the top 20 frequent keywords as well, which shows in Fig. 2. Fig. 2a and Fig. 2b shows the top 20 frequent words in real and not real disaster tweets respectively. Keywords in real disaster tweets such as ‘wreckage’, ‘oil spill’, ‘typhoon’ shows the high frequency which not occur in Fig. 1a. Similarly, keywords in not real disaster tweets such as ‘body bag’, ‘armageddon’, ‘aftershock’ show the high frequency but not in Fig. 1b. Hence, we realize that we need to consider the impact of these keywords, which can bring us additional information about the tweets.

### 2.1 Cleaning Method

In this project, we provide several cleaning methods for the raw text data, i.e., *lowering text*, *removing digits*, *removing punctuation*, *removing stop-words*, *word lemmatization*. Text lowering transforms the text into uniform case and enable to calculate the unique words more precisely. Since we are more interested in the semantic meaning of the text, cleaning digits in the text would reduce noise which existed in the sentence. We also provide cleaning method to remove the stop words such as ‘a’, ‘to’, ‘the’ and punctuation such as ‘@’, ‘!’, ‘)’ which carry few information compared to other vocabulary and character. Word lemmatization is

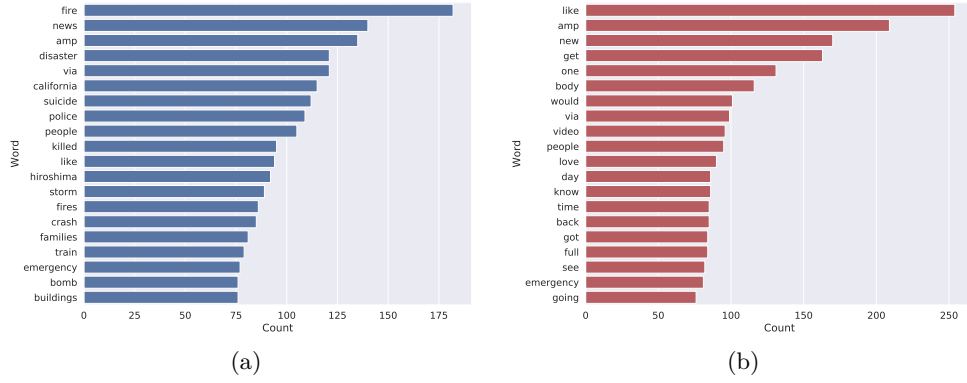


Figure 1: Top 20 frequent words in (a) Real disaster tweets and (b) not Real disaster tweets.

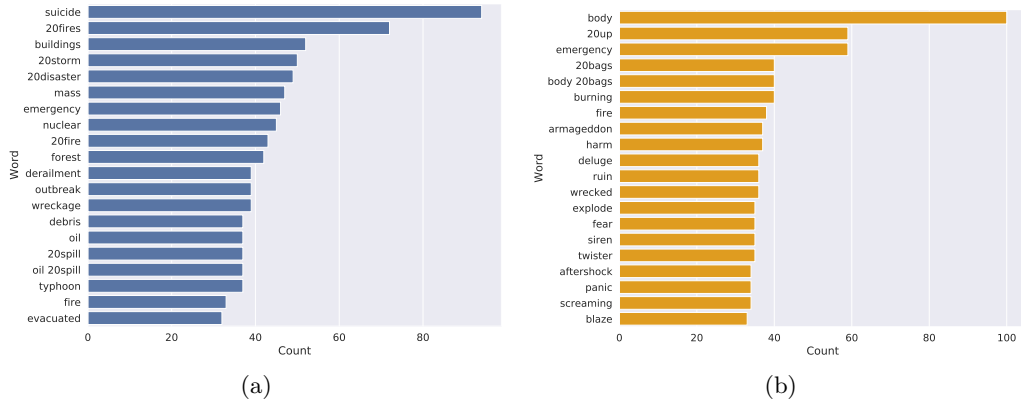


Figure 2: (a) Top 20 frequent key words in Real disaster tweets (a) and not Real disaster (b) tweets.

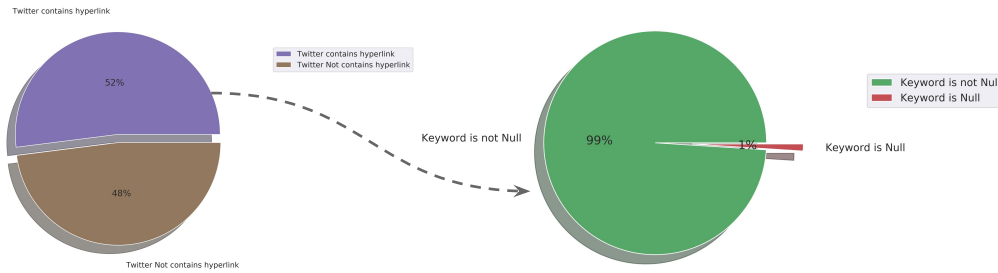


Figure 3: There are 52% of training data contains hyperlink and 48% do not contain hyperlinks (left) and among those 52% hyperlink-tweets, 99% of them contains keyword and 1% of those not.

the process that groups the inflected words to uniform term [7]. For example, 'am', 'is' will become 'be' after applying word lemmatization method. Thus, we can group the words with similar meanings. It is useful for the model based on the frequency of the words. However, it did not always increase the performance of the model with semantic embedding. We optionally use the combination of these cleaning method to process the raw text based on the model we used in this project.

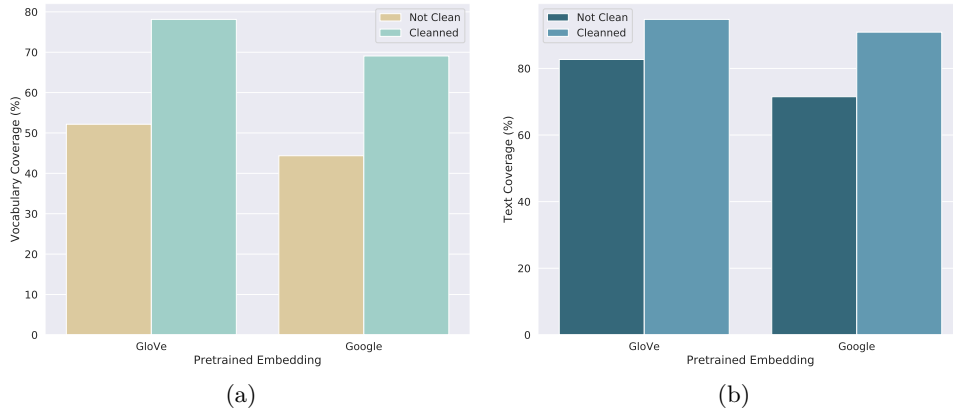


Figure 4: (a) Vocabulary embedding coverage with GloVe and Google Negative (b) Text embedding coverage with GloVe and Google Negative.

## 2.2 Hyperlink Analysis

The hyperlinks in tweets are special strings which can include meaningful information. Thus, we conduct a hyperlink analysis for the raw data. We find that there are 3971 out of 7613 ( $\sim 52\%$ ) tweets contain one or more hyperlinks, and 3642 out of 7613 ( $\sim 48\%$ ) do not contain hyperlinks. Among the 3971 tweets which contain the hyperlinks, 2172 out of 3971 of those are labeled as 1 and 1799 out of 3971 are labeled as 0. We attempt to access these hyperlinks and obtain the related HTML tags. However, we have found that part of the hyperlinks either expired or contains non-meaningful tag information. To this end, we explore the keywords of these tweets. We find that 3951 out of 3971 ( $\sim 99\%$ ) hyperlink-tweets whose keyword are not null and only 20 out of 3971 ( $\sim 1\%$ ) tweets whose keyword do not exist. We provide the optional method which can replace the hyperlink with the keyword of tweet if it exists. By doing that, we can keep the semantic information from hyperlinks.

## 2.3 Equivalent Classes

We have noticed that the training set contains 18 groups of incorrectly labeled tweets. In each group, there are at least two tweets that are absolutely identical (in texts, as well as in location and keywords if present), but labeled differently (some as relevant, some as irrelevant). For example, tweets number 4232 and 4235 have labels relevant (1) and irrelevant (0) respectively despite the fact that they contain the same text “*Caution: breathing may hazardous to your health*”, the same keyword “hazardous” and no location.

We understand an *equivalent class* as a set of tweets that have the same text (maybe after preprocessing the text of the tweets). In some cases, tweets from the same equivalent class have different label, in which case we call this class *interesting*. Thus, originally we had at least 18 interesting classes. This number increases to 79 after preprocessing. The main reason for such increase is the deletion of hyperlinks from the tweets after preprocessing. Unfortunately, more than a half of all hyperlinks in the tweets are inaccessible or contain non-meaningful information. For example, all tweets from the interesting set {6816, 6834, 6837, 6840, 6841} labeled as (0, 0, 1, 1, 0) respectively, contain the same text “*Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring hyperlink*” with different hyperlinks, all of which do not work (any more). That may be one of the reasons why simple replacement of the hyperlinks with the related HTML tags does not improve performance of our classifiers. Slight improvement (about +0.01 to all 4 parameters: accuracy,  $F_1$ , sensitivity and specificity) was demonstrated, when a whole interesting class was replaced by a single tweet with a target that was decided by majority voting. In case of equal number of votes, the target for set to 1.

## 2.4 Embedding Coverage Analysis

We used word embedding to retrieve the semantic meaning of text in LSTM, BERT, Word2Vec, thus it is essential to analyze the coverage with the corresponding embedding matrix. In this embedding coverage analysis, we evaluate the coverage of our raw data with GloVe [8] and Google Negative 300 [9]. Fig. 4a shows the vocabulary coverage with GloVe and Google Negative. We can see that the vocabulary coverage with GloVe and Google is  $\sim 52\%$  and  $\sim 44\%$  before the text cleaning. After applying cleaning methods, the vocabulary coverage increase to  $\sim 78\%$  and  $\sim 69\%$ . Fig. 4b shows the coverage of text embedding in our training data. In the raw text data,  $\sim 82\%$  and  $\sim 71\%$  of tweets contains the embedding vocabulary with GloVe and Google and increase to  $\sim 95\%$  and  $\sim 91\%$  after cleaning. In this analysis, we find some interesting words which does not include in the embedding matrix. For instance, “MH370” related to the aircraft accident of Malaysia Airline which happened on March 8, 2014, is not included in both GloVe and Google Negative. Hence, we replace “MH370” with “*aircraft accident*”. Another example is the word “*Hiroshima*” and “*Fukushima*” which corresponded to the nuclear exploration and nuclear accident, they are not included in Google Negative as well. Thus, we replace these words to the related words which included in embedding matrix.

## 2.5 Post-predictions

The following set of keywords with probability  $\geq 0.95$  means that the tweet is relevant:

`{bushfire,debris,derailment,evacuated,forestfire,hostage,oilspill,  
outbreak,rescuer,sinkhole,thunderstorm,typhoon,wreckage}`.

The experiment shows that if we change to ones (labeling them as relevant) the labels of those tweets that have a keyword belonging to this set, it improves performances of our classifiers. Such changes in labels made after classification was performed are called post-predictions. Thus, having run a classifier and we make post-predictions relabeling some of the tweets.

# 3 Methods

## 3.1 Baseline Model

Since the beginning, machine learning algorithms while dealing with natural language are faced with a major hurdle – their algorithms operate on a numeric feature space, they cannot work directly on textual data, where natural language is text. We need to transform the textual data into low-dimensional vector representation which is the fundamental step in the process of applying machine learning for analyzing text.

One of the well-known methods to perform text vectorization is Bag of Words (BOW). In BOW, a vocabulary is created by extracting the unique words from the documents. The meaning and similarity of the words in the document are encoded in the vocabulary and the term frequency of the word in the document is calculated [10].

However, there are some potential problems with BOW when applied to large corpora. The feature vectors produced by BOW are based on solely on the word frequencies, there might be some words that occur frequently overshadowing other words in the document. In such situations, Term Frequency–Inverse Document Frequency (TF-IDF) is a simple method which can be used.

The main advantage of TF-IDF is that it uses a normalizing factor in its computation. It considers the relative frequency of words in the document against their frequencies in the remaining corpus. A TF-IDF score is a measure used to evaluate the importance of a word across the corpus. TF-IDF score for a word is calculated by multiplying two statistical measures, Term Frequency (TF) and Inverse document frequency (IDF) for a word [11].

For implementing TF-IDF, we are using tokenized tweets that are obtained after the cleaning process. Term Frequency (TF) measures the frequency of a word in the document with respect to the total number of words present in the document. While computing the term frequency, every word is considered equally important. However, there can be certain words which are unavailing in conveying information. To reduce the importance of such words, they are multiplied by the Inverse Document Frequency (IDF). IDF measures the importance of a word, i.e., how common or rare a word is across the whole corpus. IDF is calculated by taking the total number of documents, dividing by the number of documents that contain a word, and applying logarithm to it. Multiplying these two measures will give in a TF-IDF score for each word [11].

$$\text{TF}(t, d) = \frac{\text{Number of times word } t \text{ appears in a document } d}{\text{Total number of words in a document } d}, \quad (1)$$

$$\text{IDF}(t, D) = \log \frac{\text{Total number of documents}}{\text{Number of documents with word } t \text{ in it}}, \quad (2)$$

$$\text{TFIDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D), \quad (3)$$

where  $t$  is a term,  $d$  is a document, and  $D$  is corpus of words.

After the TF-IDF scores are calculated for all the words in our corpus (13,841 words) and a vector representation of words is obtained as 13,841-dimensional sparse vectors, we can use them to train our base-line classifiers: Logistic Regression (LR) and Support Vector Machine (SVM). Despite the fact that these model do not account for semantics in the tweets, they have demonstrated a quite decent performance. As shown Tab. 1, SVM classifier model has obtained an accuracy of 76.1% and LR has obtained an accuracy of 78.1%.

Table 1: Accuracy,  $F_1$ , Precision, and Recall for the baseline TF-IDF + SVM and LR classifiers.

Baseline models	Accuracy	$F_1$	Precision	Recall
Logistic Regression	0.7615	0.7561	0.7745	0.7611
SVM	0.7815	0.7775	0.7845	0.7817

### 3.2 LSTM Glove

Recurrent Neural Network (RNN) with Long Short-term memory (LSTM) model has been successful in providing impeccable accuracy. One of reason to select LSTM in this project is also the statistic of successful Kaggle completions that shows 23% of top three scores are using LSTM model.

The original LSTM [12] was introduced in 1997, and it has been modified to advance its functionality in various variant architectures.

In this project, LSTM model is considered as shallow model consisting of 5 layers; namely, embedding, LSTM, dropout, dense, and activation. The embedding layer is the input layer to map variable to a vector of continuous numbers (categorical). This layer helps to present variable in the transform space. The concept of one-hot (or one-of- $K$  scheme) is applied in this layer to convert or transform input variable into encoding data that is not based on the high or low of categorical value. This layer is defined with an use of a vector space model as weighting mechanism which is important in semantic approach. This approach helps the algorithm to “understand” words in similar contexts having similar meanings. The LSTM layer defines the number of hidden nodes. If nodes or neurons is too sparse, it would result the underfitting and if nodes is too dense, it would result in the overfitting as well as

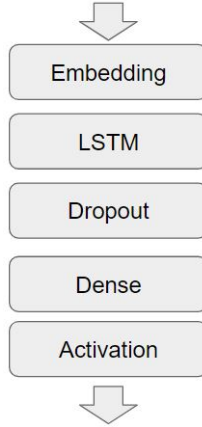


Figure 5: LSTM model design.

in large computational expenses. The two thirds of the size of input layer plus the size of output layer are used in this model. Regularizer is applied to avoid overfitting. Regularizer does add penalty on the weight size (norms) into the loss function [13]. The most popular norms include  $L_1$  norm (the sum of the absolute values of the vector) [14],  $L_2$  norm (the square root of the sum of the squared vector values) [15], and  $L_{2,1}$  norm of a matrix (with  $L_2$  norm in rows, and  $L_1$  in columns) [16]. The norm  $L_2$  is chosen in this model. Dropout layer is added to drop some hidden nodes randomly during training, and as the result, it reduces the sensitivity of some individual hidden nodes and helps to prevent overfitting. Dense layer and activation layer are combined in this model to define the output. The sigmoid function is chosen in the activation layer for the binary classification.

Global vectors for word representation (GloVe) [8] is added to the embedding for the comparison of the model accuracy. With the pre-trained vector GloVe, the model creates a matrix of embedding for each word from training dataset by loading each unique word (as tokens) and locating the embedding weight vector from GloVe embedding. GloVe with 300-dimensions is chosen in this model. For this method, the words not found in GloVe mapped to random vectors from  $\mathcal{N}(\mu, \sigma^2)$ , where  $\mu$  and  $\sigma$  are a mathematical expectation and a standard deviation of the embedded words that are in GloVe.

A parameter `trainable` is set to `False` to disable the learned word weight to be updated as it is not required [17]. Note that in Keras (which is used in this LSTM model), each layer has a “trainable” parameter which is `enable` by default. To freeze the weight learning of this embedding layer, this parameter is set to `False`, indicating that this layer should not be trained because GloVe is pre-trained.

### 3.3 Word2Vec

Numerous NLP methods [2] do not take into account semantics of the words treating them as separate units. Such methods do not incorporate notion of similarity between words, the words are represented merely as indices of a large vocabulary. Such models have several obvious advantages as simplicity and robustness. One of the well-know examples of such model is a so-called  $N$ -gram model [18] often used for statistical modeling of languages nowadays. However, if we want to incorporate semantic meaning in our model or merely train it on billions of words these simplistic methods cannot work well and quickly enough. In such situation, we need more advanced approaches that not only can be used on very large sets of data with billions of words, and millions of words in the vocabulary, but also would take into consideration words meaning itself and with respect to their position in a sentence. The Word2vec method is one of such methods developed in 2013 by a team of Google researchers led by Czech computer scientist Tomas Mikolov [19, 20, 21].



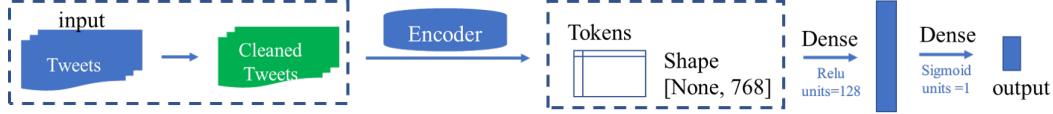


Figure 6: A view of the framework of BERT-based model.

The main advantage of Word2vec is that it can compute in a reasonable time (less than a day in 2013 for 1.6 billion words data set, see the abstract in [19]) continuous vector representations of words from very large data sets. It was a breakthrough in comparison with many models that were state-of-the-art in 2013, like Latent Semantic Analysis, or neural networks that provided distributed representations of words. The main observation made in [19] was that while neural networks performed quite well, the computational complexity caused by the non-linear (hidden) layers was making them impractical, especially for large datasets. Two simpler but much more computationally efficient methods were proposed in [19, 20] and proved to be quite efficient in deriving distributed representations of words.

These two methods are Continuous Bag-of-Words (CBOW) and Continuous Skip-Gram. CBOW is somewhat similar to the feedforward neural networks language model from [22], with the exception that the non-linear (hidden) layer is not present, while the projection layer is shared for all words. CBOW predicts a current word based on its context, i.e., surrounding words in a sentence (or a document). Continuous Skip-Gram method does the opposite: it predicts a context given a current word.

In our model, we are using pre-trained vectors from Google News dataset (so-called GoogleNews-vectors-negative300, available on [9]). The vectors are obtained using Continuous Skip-Gram method with the context window equal 10 words. The vector were trained on part of Google News dataset (that contains 100 billion words). This model contains mapping of about 3 million words into 300-dimensional vectors, whose similarities are defined by the angle between vectors (cosine distance).

We treat a tweet as a 300-dimensional vector equal to an average of vector embedding of the (cleaned and preprocessed as described in the section Data Processing) words in it. If a word happens to be not included in the GoogleNews-vectors-negative300 corpus we assign zero embedding to this word, i.e., simply ignore it. It worth mentioning that we have tried several ideas here, including assigning random values to such word, but zero-assignment provides the best results.

### 3.4 BERT

To continuously improve the performance of the disaster classification task, we repeat, in retrospect, what is still not covered with the previously mentioned methods. In the baseline models, we count the frequency of each word (e.g., TF-IDF) and take it as the feature to feed the training model. We also have implemented the LSTM, a type of recurrent neural network, which can recapture the information of the order of the words in a sentence, and utilize entire sequences of words rather than a single word to predict the new state.

Recently, a newly emerging framework of *transformer* has achieved outstanding performance in NLP tasks. The transformer improves the shortcomings of RNN’s most criticized efficiency during the training, which uses the self-attention mechanism to achieve fast parallel training. Based on the powerful learning ability of the transformer, the model can increase to a deep depth, which fully explores the characteristics of the deep neural network and improves the accuracy of the task.

BERT [23] is a Bidirectional Encoder Representation from Transformers, which can encode the corpus to the embedding in the level of sentence and jointly consider both the left and right context in all the layers. Research shows that BERT has been widely cooperated with twitter textual data to perform the specific task such as rumor detection or hate speech



detection [24, 25]. To implement the BERT-based model, there are two main steps, which are pre-training and fine-tuning, respectively.

- The pre-training is executed on two types of tasks, named the “Masked Language Model” and “Next Sentence Prediction”. The essence of the “Mask Language Model” is to mask some of the words or substitute them with random words and then ask the model to predict the word in a specific position. For the “Next Sentence Prediction” task, it requires the model to understand the relationship between sentences and predict whether a sentence is behind another one.
- The fine-tuning process on the different tasks can be initialized with the same pre-trained parameters. The difference between the pre-trained parameters and fine-tuning parameters is subtle, which can also be verified according to the controllable parameters in our experiment.

Due to the unique bidirectional encoding mechanism of BERT, some of the cleaning techniques mentioned in other models are not suitable for BERT. Generally, the essential principle is that we cannot add some irrelevant tokens, which may undermine the original semantics. At the same time, it is better to keep the sentence structure but delete those non-informative words, punctuation. Hence, in the model of BERT, we need to selectively and cautiously choose the cleaning methods mentioned (e.g., removing digits, removing hashtags, removing punctuation) while abandoning some techniques such as removing stop-words, lemmatization and stemming.

After the cleaning process, we use WordPiece embeddings [26] with a total of 30,000 token vocabulary. The tokenization method employed a BPE (Byte-Pair Encoding) technique to cut the word and simplify the token list. Apart from the special rule of the tokenizing, BERT also uses the special tokens to label the sentence (i.e., initialize any sequence with a token [CLS] and separate different sentences with a token [SEP]).

Once we obtain the input of the token list for the encoder, we utilize the output from the encoder as the features vector and add two dense layers with the activation functions RELU or SIGMOID. Fig. 6 plots a view of the basic framework of our BERT-based method.

## 4 Experiments and Results

### 4.1 LSTM Glove

The data processing has significant impact to an accuracy improvement from 54% to 78%. However, some level of data processing, such as, HTTP link replacement would decrease an accuracy. Vanilla LSTM is referred to shallow LSTM model without semantic and to compare with the GloVe which is underline vectored embedding. The comparison of performances on the training and the test data sets as well as some other statistical characteristics are reported in Fig. 7 and Tab. 2. As shown in Tab. 2, LSTM with a GloVe accuracy is better +3%, which is a little improvement. However, LSTM with GloVe has eliminated the over-fitting problem when the same LSTM layer is used. Fig. 7b reveals that this trained model classifies true negative error value is too high. This provides the improvement opportunity in the area of data processing where fake twitter is classify as real twitter more than it should be.

Table 2: Accuracy,  $F_1$ , Precision, and Recall for LSTM.

<b>LSTM</b>	Accuracy	$F_1$	Precision	Recall
Vanilla LSTM	0.7677	0.7683	0.7695	0.7677
LSTM with GloVe	0.7937	0.7884	0.8008	0.7937

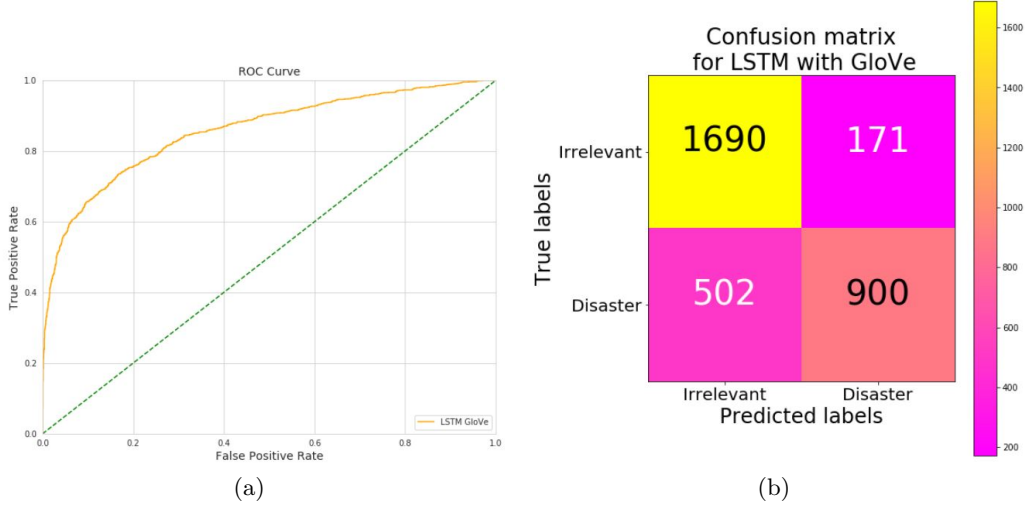


Figure 7: Performance of LSTM with GloVe on test data represented as (a) an ROC curve, and (b) a Confusion Matrix.

## 4.2 Word2Vec + SVM

For this model, we first preprocessed data (as it is described in the section Data Processing), then we map tweets into 300-dimensional vectors (as it is described in the section Methods). Next, we use SVM classifier to separate tweets into two classes: relevant or irrelevant. To tune metaparameter  $C$  in SVM, we performed a 5-fold cross-validation using `GridSearchCV` from `sklearn` and searching for optimal  $C$  in the range  $[0.5, 5]$  with step 0.1. The best  $C$  for our problem is equal to 2. Finally, post-predictions were made using the set of 95%-important keywords as described in the section Post-predictions. The comparison of performances on training and test data sets as well as some other statistical characteristics are reported in Fig. 8 and Tab. 3.

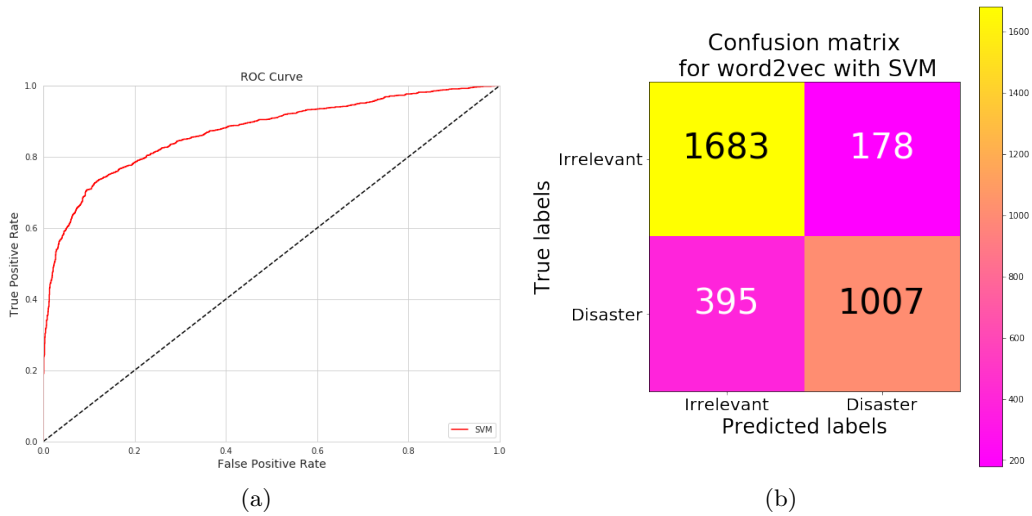


Figure 8: Performance of Word2Vec embedding with SVM classifier on test data represented as (a) an ROC curve, and (b) a Confusion Matrix.

Table 3: Accuracy,  $F_1$ , Precision, and Recall for Word2Vec embedding with SVM classifier.

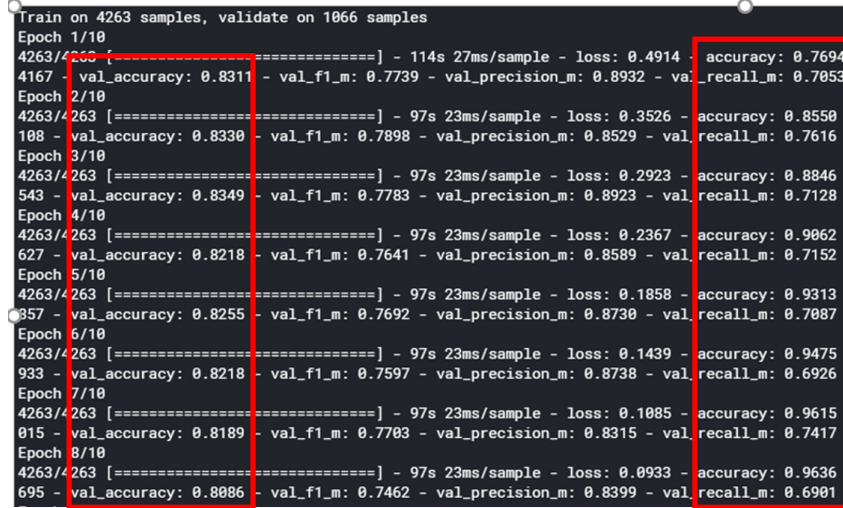
Word2Vec + SVM	Accuracy	$F_1$	Precision	Recall
Training set	0.8443	0.8426	0.8461	0.8443
Test set	0.8244	0.8219	0.8270	0.8244

### 4.3 BERT

For our task, we use a **BERT**<sub>BASE</sub> pre-trained model from the TensorFlow Hub [27] to reduce the processing time. The pre-trained BERT was implemented with 12 hidden layers, in each of which the size is 768. The model also cooperated with 12 attention heads, which can help the encoder to find other related words when performing the encoding.

Besides, two dense layers, with the activation functions of relu and sigmoid, respectively, are added after the output vector from the pre-trained model. We use the batch with the size of 16 and fine-tune the model for 2 epochs and select the fine-tuning learning rate of 6e-6, which shows the best result in our experiment.

During the experiment, we found it not suitable to set a high learning rate or a large number of epochs to train in the fine-tuning task. After continuously testing, we found that even the same parameter setting cannot always provide the same performance again. The reasons behind may be that the fine-tuning process is random within little times of epochs, and even slight changes of any parameter may lead to a dynamic result, which is a local minimum. However, if the model is trained with more epochs, the model will easily step into an over-fitting stage. As shown in Fig. 9, the accuracy is constantly increasing while the validation accuracy is decreasing after the third epoch.



```

Train on 4263 samples, validate on 1066 samples
Epoch 1/10
4263/4263 [=====] - 114s 27ms/sample - loss: 0.4914 - accuracy: 0.7694
4167 - val_accuracy: 0.8311 - val_f1_m: 0.7739 - val_precision_m: 0.8932 - val_recall_m: 0.7053
Epoch 2/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.3526 - accuracy: 0.8550
188 - val_accuracy: 0.8330 - val_f1_m: 0.7898 - val_precision_m: 0.8529 - val_recall_m: 0.7616
Epoch 3/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.2923 - accuracy: 0.8846
543 - val_accuracy: 0.8349 - val_f1_m: 0.7783 - val_precision_m: 0.8923 - val_recall_m: 0.7128
Epoch 4/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.2367 - accuracy: 0.9062
627 - val_accuracy: 0.8218 - val_f1_m: 0.7641 - val_precision_m: 0.8589 - val_recall_m: 0.7152
Epoch 5/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.1858 - accuracy: 0.9313
857 - val_accuracy: 0.8255 - val_f1_m: 0.7692 - val_precision_m: 0.8730 - val_recall_m: 0.7087
Epoch 6/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.1439 - accuracy: 0.9475
933 - val_accuracy: 0.8218 - val_f1_m: 0.7597 - val_precision_m: 0.8738 - val_recall_m: 0.6926
Epoch 7/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.1085 - accuracy: 0.9615
015 - val_accuracy: 0.8189 - val_f1_m: 0.7703 - val_precision_m: 0.8315 - val_recall_m: 0.7417
Epoch 8/10
4263/4263 [=====] - 97s 23ms/sample - loss: 0.0933 - accuracy: 0.9636
695 - val_accuracy: 0.8086 - val_f1_m: 0.7462 - val_precision_m: 0.8399 - val_recall_m: 0.6901

```

Figure 9: The accuracy of training increases while the validation accuracy decreases in BERT model.

Our best result with the BERT model on the leader board is 84.25%. It takes 21 seconds to run through all the 3263 testing tweets, each of which consumes 16ms to complete. After multiple times of tuning, we found the result from the model with pre-cleaning is slightly better than the model without cleaning. But we cannot say one is outperforming than the others. The comparison of performances on training and test data sets as well as some other statistical characteristics are reported in Fig. 10 and Tab. 4.

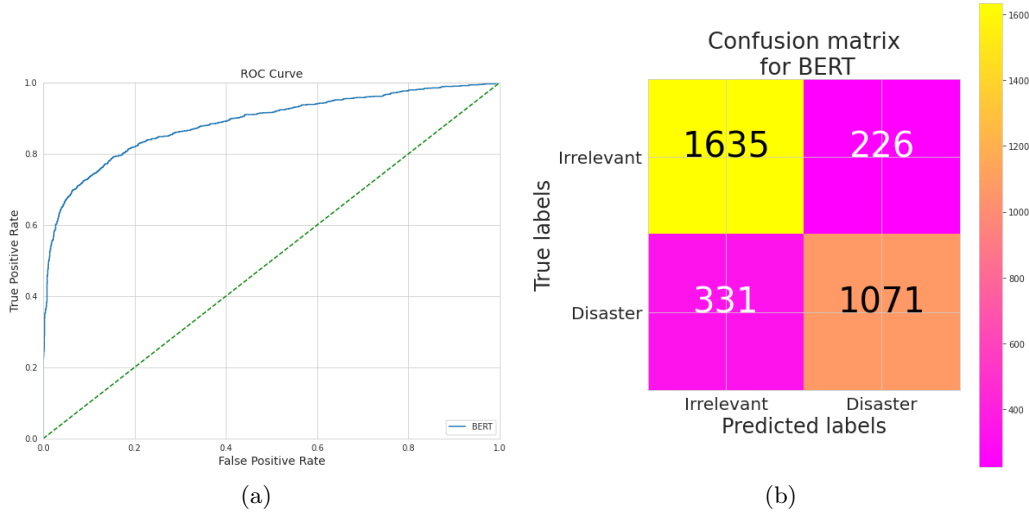


Figure 10: Performance of BERT with two dense layer network on test data represented as (a) an ROC curve, and (b) a Confusion Matrix.

Table 4: Accuracy,  $F_1$ , Precision, and Recall for BERT.

BERT	Loss	Accuracy	$F_1$	Precision	Recall
Bert with Cleaning	0.3158	0.8774	0.8474	0.8759	0.8276
Bert without Cleaning	0.4020	0.8314	0.7920	0.8689	0.7387

#### 4.4 Comparison of the models

We compare our three main models – LSTM, Word2Vec with SVM, and BERT – not only by their accuracy,  $F_1$ -score, sensitivity and specificity, but also by contrasting their ROCs with each other. See them all plotted in the same Fig. 11a. In this graph, we can see that ROC for BERT is almost everywhere above ROC for Word2Vec with SVM, denoted as SVM in the legend. Likewise, ROC for Word2Vec is almost everywhere above ROC for LSTM. This observation corroborates the fact that accuracy,  $F_1$ -score, sensitivity and specificity for BERT are greater than those for Word2Vec with SVM, which in their turn are large than for LSTM. Therefore, we conclude that BERT is out best performing model. Notably, the public scores for the methods on kaggle [4] (calculated as  $F_1$ -score for 30% of the test set) also support such conclusion, see Fig. 11b.

## 5 Discussion

The goal of this project was to create a machine learning classifier which could label tweets as either relevant to the natural disaster (1) or irrelevant (0). The pipeline in our approach boils down to the following 3 steps: tweets preprocessing; mapping tweets into vectors and classifying them; making post-predictions based on keywords information.

Tweets preprocessing included lowering text, removing digits, punctuation, and stop-words, as well as a word lemmatization. Additionally, classes of identical tweets with contradictory labeling were treated as single tweets which label was decided by a majority voting among the class.

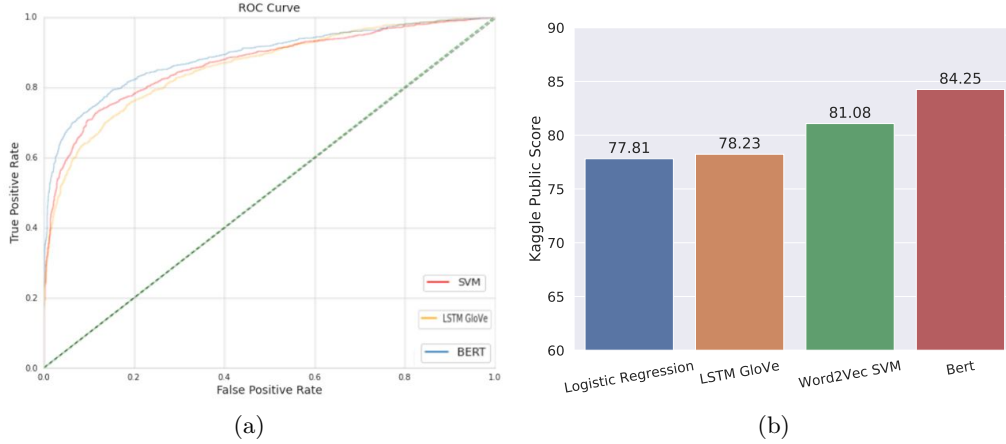


Figure 11: (a) ROC Curves comparison. BERT has the largest area under the curve, while LSTM has the smallest. (b) The public scores we have achieved on kaggle for baseline model with LR classifier (blue), LSTM GloVe (orange), Word2Vec with SVM classifier (green) and BERT (red).

For tweets mapping and classifying, TF-IDF with SVM and LR classifiers were deployed as a baseline model, and other 3 models (the main models) were explored as the candidates for a final and best-performing model: LSTM, Word2Vec with SVM classifier, and BERT.

The predictions of these classifiers were slightly improved by making post-predictions, i.e., correcting labels of the tweets marked by the classifiers as 0's but have the keyword that with high probability determines that the tweet is relevant.

The main models were compared with a set of statistical characteristics as accuracy,  $F_1$ -score, sensitivity, specificity, area under the curve, and public scores on Kaggle. All the findings here suggest that BERT is our best performing model (with 0.8774 accuracy,  $F_1 = 0.8474$ , 0.8759 sensitivity, and 0.8276 specificity), reaching 0.84253 public score and earning 306 place out of 2,420 teams on Kaggle leader board (on May 3, 2020).

We believe that the performance of the Word2Vec model can be further improved if a smarter way of averaging words in a tweet in the embedding step is performed. This averaging could take into account the relative importance of the words (including keywords) for prediction of the label. In this case, the post-predictions step may be redundant.

Another improvement may be achieved if the mislabeled tweets in the training dataset were labeled correctly. We perform majority voting for the equivalent classes which may be crude if the ratio of votes is close to 0.5.

It is clear that a very important part of a classifier for this problem is a proper embedding (mapping into dense vectors) of the words in a tweet. We believe that training the transformer that maps words into vector on a dataset related to a specific problem might substantially improve the performance of the classifier. Thus, it may be a good idea to use disaster related embedding matrix to more precisely extract the correlations of the semantic information from the tweets.

The code and data are publicly available on Kaggle platform, see, e.g., [LSTM](#), [Word2Vec](#), and [BERT](#). Our public score could be found by team name “Shiba\_Inu” on [kaggle leaderboard](#).

## References

- [1] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

- [2] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.
- [3] Bollen J Rojas F DiGrazia J, McKelvey K. More tweets, more votes: Social media as a quantitative indicator of political behavior. *PLoS ONE*, 8(11):e79449, 2013.
- [4] Real or Not? NLP with Disaster Tweets on Kaggle. <https://www.kaggle.com/c/nlp-getting-started>. (Accessed on 05/01/2020).
- [5] Anna K on Twitter: “On plus side LOOK AT THE SKY LAST NIGHT IT WAS ABLAZE” / Twitter. <https://twitter.com/AnyOtherAnnaK/status/629195955506708480>. (Accessed on 05/01/2020).
- [6] Nadia Felix F. Da Silva, Luiz F. S. Coletta, and Eduardo R. Hruschka. A survey and comparative study of tweet sentiment analysis via semi-supervised learning. *ACM Comput. Surv.*, 49(1), June 2016.
- [7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.
- [8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [9] Pretrained vectors from Google News dataset GoogleNews-vectors negative300. <https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit>. (Accessed on 05/01/2020).
- [10] Kilian Q. Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alexander J. Smola. Feature hashing for large scale multitask learning. *CoRR*, abs/0902.2206, 2009.
- [11] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3), June 2008.
- [12] Jürgen Schmidhuber Sepp Hochreiter. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.
- [13] Mark Crowley Benyamin Ghogh. The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial. <https://arxiv.org/abs/1905.12787>, May 2019. (Accessed on 05/03/2020).
- [14] Romer Rosaless Mark Schmidt, Glenn Fung. Optimization methods for  $l_1$ -regularization. <http://people.duke.edu/~hpgavin/SystemID/References/Schmidt-UBC-TR-2009-19.pdf>, March 2008. (Accessed on 05/03/2020).
- [15] Hastie Trevor Friedman, Jerome and Robert. Tibshirani. *The elements of statistical learning, volume 2*. Springer, New York, USA, 2009.
- [16] Yale Chang.  $l_{2,1}$  norm and its applications. [http://www1.ece.neu.edu/~ychang/notes/L21\\_norm.pdf](http://www1.ece.neu.edu/~ychang/notes/L21_norm.pdf). (Accessed on 05/03/2020).
- [17] Jason Brownlee. How to use word embedding layers for deep learning with keras, October 2017.
- [18] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992.
- [19] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

- [20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.
- [21] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.
- [22] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003.
- [23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [24] Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Manuel Rangel Pardo, Paolo Rosso, and Manuela Sanguinetti. Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 54–63, 2019.
- [25] Lin Tian, Xiuzhen Zhang, Yan Wang, and Huan Liu. Early detection of rumours on twitter via stance transfer learning. In *European Conference on Information Retrieval*, pages 575–588. Springer, 2020.
- [26] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [27] Pretrained BERT from Tensor Flow Hub. [https://tfhub.dev/tensorflow/bert\\_en\\_wwm\\_cased\\_L-24\\_H-1024\\_A-16/2](https://tfhub.dev/tensorflow/bert_en_wwm_cased_L-24_H-1024_A-16/2), (Accessed on 05/01/2020).