

CSE145: Course Project Report

Daniel Chen
Student ID: 1602167
Email: dchen61@ucsc.edu

June 7th, 2020

1. Abstract

When a disaster happens, timing is very important. Knowing about a disaster as soon as possible could easily save lives and allow people time to plan and evacuate. News agencies could report these disasters to inform residents to evacuate. Disaster relief organizations and emergency agencies could dispatch earlier to assist locals and perhaps combat the known disaster. Smartphones are ubiquitous and a majority of people have social media accounts. Twitter has become one of the main communication channels in times of emergency. There are tons of Tweets being sent out every minute and machines could easily sweep through all of them. The problem is determining whether or not a Tweet indicates a real disaster. Keywords for disasters could be used metaphorically, making the problem even harder. This problem is also featured as a competition on Kaggle, and I became one of the participants. My goal is to create a reliable machine learning model that could classify whether a Tweet indicates a real disaster or not. I will create three different kinds of models: a Naive Bayes model using the Python library Scikit-learn, a Keras deep learning model that uses GloVe, and a Keras deep learning model that uses BERT. Both GloVe and BERT were pre-trained on much larger datasets, and BERT has been shown to produce state-of-the-art results on some natural language processing problems. Through transfer learning, using GloVe and BERT will help the performance of my models. To evaluate the performance and reliability of my classification models, I used accuracy and F1 score. Since Tweets are text data, there is a lot of data preprocessing and cleaning that needed to be done. For the two deep learning models, several hyperparameters were tuned to improve the model performances as well. On validation data, Naive Bayes performed the worst, resulting in a validation F1 score of 0.4920. The other two deep learning models performed significantly better, with the model using GloVe having a validation F1 score of 0.7519 and the model using BERT having a validation F1 score of 0.7758. On test data, after submitting the predictions from the models to Kaggle, the GloVe-based deep learning model had an F1 score of 0.7924 and the BERT-based deep learning model had an F1 score of 0.8221. These results show that the deep learning model with BERT has the best performance out of the three. After submitting the first deep learning model, I ranked 894th place out of 1654 teams and after submitting the latter model, I ranked 325th place. This was a major jump in the rankings. It is difficult to determine what the highest F1 score could be achieved so far by a classification model, because the labels to the test set are public (not provided by Kaggle though) and many people submitted them to achieve a perfect F1 score. However, one person who attached his/her notebook along with the submission showed that an F1 score of 0.8466 could be achieved. My highest achieved F1 score is 0.0245 short, but besides that, my best performing model still produced really good results.

2. Introduction

Twitter is one of the largest social media platforms around right now with the likes of Facebook and Instagram. The company touts an estimated 330 million monthly active users and an estimated 145 million daily active users. Everyone has a smartphone around no matter where they go and tweet all sorts of things. However, in times of emergency, people tweet about disasters in real-time. Using these Tweets, agencies such as disaster relief organizations and news agencies could determine the kind of disaster taking place and the location of it.

There is a vast amount of Tweets being sent out every minute, and it is manually impossible for people to look each of them. Machines, on the other hand, could sweep through them very quickly, which leads us to a binary classification task where we need to determine whether a Tweet is talking about a real disaster or not. Things that are obvious to people might not be the same for machines, such as sarcasm and metaphors.

Here is an example of two very different Tweets:

- 1) "On plus side LOOK AT THE SKY LAST NIGHT IT WAS ABLAZE"
- 2) "Thousands of wildfires ablaze in California alone."

Both of these Tweets contain the word 'ablaze'. However, in the first Tweet, the word 'ablaze' is used metaphorically to describe how red the sky looked and in the latter Tweet, 'ablaze' is being used literally, indicating a real disaster. This problem is one of the competitions on Kaggle¹ and the three different kinds of models I will be using are Naive Bayes with Scikit-learn, deep learning with Keras using GloVe word embeddings, and deep learning with Keras using BERT contextualized word embeddings. Then, I will submit the predictions from these models onto Kaggle.

3. Experimental Methodology

Data Understanding and Preprocessing

Kaggle has provided two datasets for participants, namely a training set and a test set. The training set contains 7613 Tweets and the test set contains 3263 Tweets. Each Tweet in the training data contains five attributes: id, keyword, location, text, and a label. For the test data, each Tweet has the same attributes except the label, which is held-out by Kaggle. The keyword and location are the two attributes that have missing values. However for this project, I only decided to use the text and label attributes, so I deleted the rest.

In the training set, 4342 instances ($\approx 57\%$) are 0's indicating not a real disaster and 3271 instances ($\approx 43\%$) are 1's indicating a real disaster. The data isn't perfectly balanced, but it is pretty close. Within the data, 110 Tweets are duplicates and the most frequent duplicated Tweet occurs 10 times. After taking a look at the duplicated Tweets, 18 unique Tweets are labeled differently than their duplicates. The Tweets are a little confusing and it isn't easy to determine whether it is a real disaster or not. Different people probably labeled these Tweets. To make things consistent, I relabeled the Tweets based on whether I thought they indicated a real disaster or not.

The Tweets provided in the datasets are very messy and aren't preprocessed, so I had to do some data cleaning. For each Tweet, I expanded some very common abbreviations using a Python script that I found online.² Then, I made all the characters lowercase, removed all links, and deleted all the link breaks, extra spaces, punctuations, symbols, and emoticons. After cleaning the data, there is a total of 17,076 unique vocabulary words in the training set. 75% of the Tweets contain 19 tokens or below and 99% of the Tweets contain 28 tokens or below. The longest Tweet in the training data contains 34 tokens.

After cleaning, I tokenized the text using the Tokenizer from Keras. The internal vocabulary gets updated from the list of Tweets and each Tweet gets transformed into a sequence of integers. Each unique word is mapped to an index (integer). Once the text is tokenized, I would pad the end of each sequence with 0's depending on if the length of the sequence is less than the max length that I will later choose in hyperparameter tuning. If the sequence is longer than the chosen max length, it will cut the end of the Tweet short. This is to make all sequences the same length.

I had to tokenize the text a bit differently for my deep learning model with BERT. In addition to what was previously done, I also had to include a special [CLS] token to the beginning and a [SEP] token to the end of every sequence before padding.

¹Kaggle competition: <https://www.kaggle.com/c/nlp-getting-started>

²Link to the script is here: <https://medium.com/nerd-stuff/python-script-to-turn-text-message-abbreviations-into-actual-phrases-d5db6f489222>

Evaluation Metrics

The three evaluation metrics I will be using to evaluate my predictive models are accuracy, F1 score, and binary cross-entropy. Binary cross-entropy will be used while training the deep learning models and only F1 score will be used when running my models against the test dataset. This is because Kaggle has not provided the labels for the test set, and they will provide the F1 score after submitting the predictions from a classification model.

Experiment Setup

Naive Bayes and deep learning work quite differently, so I have two separate experimental methods.

For my Naive Bayes model, I first randomly split the training data using a 90/10 split to create a validation set. I chose 90/10 because the training data is very small. Other very common splits are 80/20 and 50/50. The reason for creating a validation set when using Naive Bayes is because I did not want to submit the predictions from this model just yet and I needed a dataset to evaluate its performance. So, after creating my Naive Bayes model, I ran it against the training and validation set and recorded the accuracies and F1 scores.

For my deep learning models, I, again, first randomly split the training data using a 90/10 split to create a validation set. Then, I built my deep learning model and tuned my hyperparameters. The hyperparameters I chose to tune were: the max length of each Tweet, the max number of words in my vocabulary, embedding dimension, number of recurrent units for my LSTM layer, dropout, recurrent dropout, which optimizer to use, learning rate, and batch size. For the tuning process, I first chose a starting value for each hyperparameter and trained my model on 10 epochs. Then, I would tune each hyperparameter one at a time while keeping the values for the others constant and train my model on 10 epochs each time again. I chose the value for each hyperparameter that lead to the highest achieved F1 score on the validation set at any epoch during training. To prevent overfitting, I used the ModelCheckpoint callback method from Keras that monitors the F1 score and saves the model at the epoch that achieved the highest score. After tuning each hyperparameter, I then retrained the model with the final values and recorded the accuracies and F1 scores on both the training and validation set for that epoch.

Data Mining Algorithms

Naive Bayes

I created my Naive Bayes model using the Python package Scikit-learn. It makes classifications following this formula:

$$\hat{y} = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y),$$

and using Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i|y)$. $P(y)$ is just the relative frequency of class y in the training set. Creating a Naive Bayes classification model is much faster than deep learning.

Deep Learning with GloVe

For this model, I decided to use pre-trained word embeddings that were generated by Stanford's GloVe: Global Vectors for Word Representation. There are several different pre-trained word vectors that they offer, which were all trained on different corpuses. I chose the word vectors that were trained on the Twitter corpus.³ It includes: 2B uncased Tweets, 27B tokens, and 1.2M vocabulary words. This is a much larger training dataset compared to the training set I was provided by Kaggle, which has 7613 Tweets.

³Stanford's pre-trained word vectors can be found here: <https://nlp.stanford.edu/projects/glove/>

Here is the model architecture for my first deep learning model:

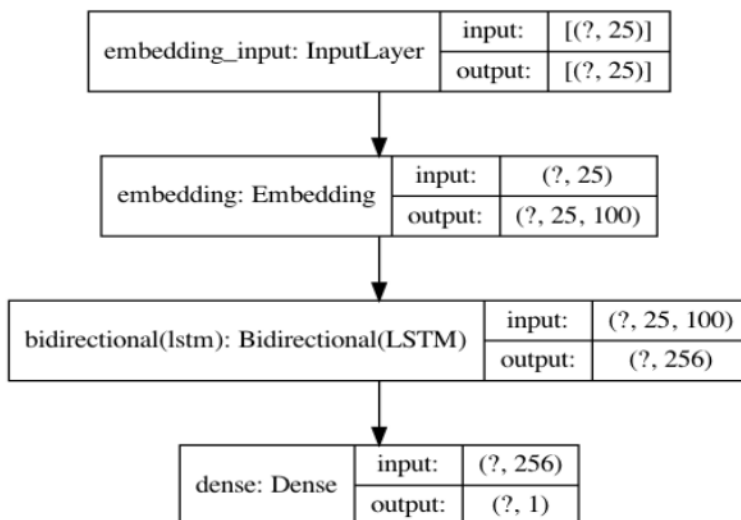


Figure 1: Layers for deep learning model using GloVe

These are the final values for the hyperparameters I tuned:

	Value Resulting in Highest F1
Max Length	25
Max Words	30,000
Embedding Dim	100
# Recurrent Units	128
Dropout	0.0
Recurrent Dropout	0.1
Optimizer	Adamax
Batch Size	32

All other hyperparameter values are set to their default values.

The model architecture is comprised of four different layers. The first layer just simply takes in the input and feeds it into the Embedding layer. In the Embedding layer, I imported the embedding matrix from GloVe with the trainable parameter set to False and this layer is used to create word vectors from the input. The output then gets passed into the Bidirectional LSTM layer with 128 recurrent units. This layer models a sequence and captures the context from the left and right. Since this is a bidirectional layer, the output dimension doubles to become 256 units and gets passed into the final Dense layer. The Dense layer uses the sigmoid activation function and produces the probability of a Tweet being a real disaster. To make classifications, I rounded the probability. If the probability was ≥ 0.5 , it would round to 1, which means that the Tweet indicates a real disaster. If the probability was < 0.5 , it would round to 0, meaning it isn't a real disaster.

Deep Learning with BERT

For this model, I used Google's Bidirectional Encoder Representations from Transformers (BERT). BERT was pre-trained on Wikipedia and BooksCorpus. What makes BERT different from GloVe is that BERT generates contextualized word embeddings, whereas GloVe generates context-free single word embeddings.

The difference can be shown using the word 'apple', which could be seen as a fruit or company. BERT will generate two different word embeddings, one for 'apple' as a fruit and one for 'apple' as a company. GloVe, on the other hand, will only generate one word embedding for 'apple' to use in all contexts.

Here is the model architecture for my second deep learning model:

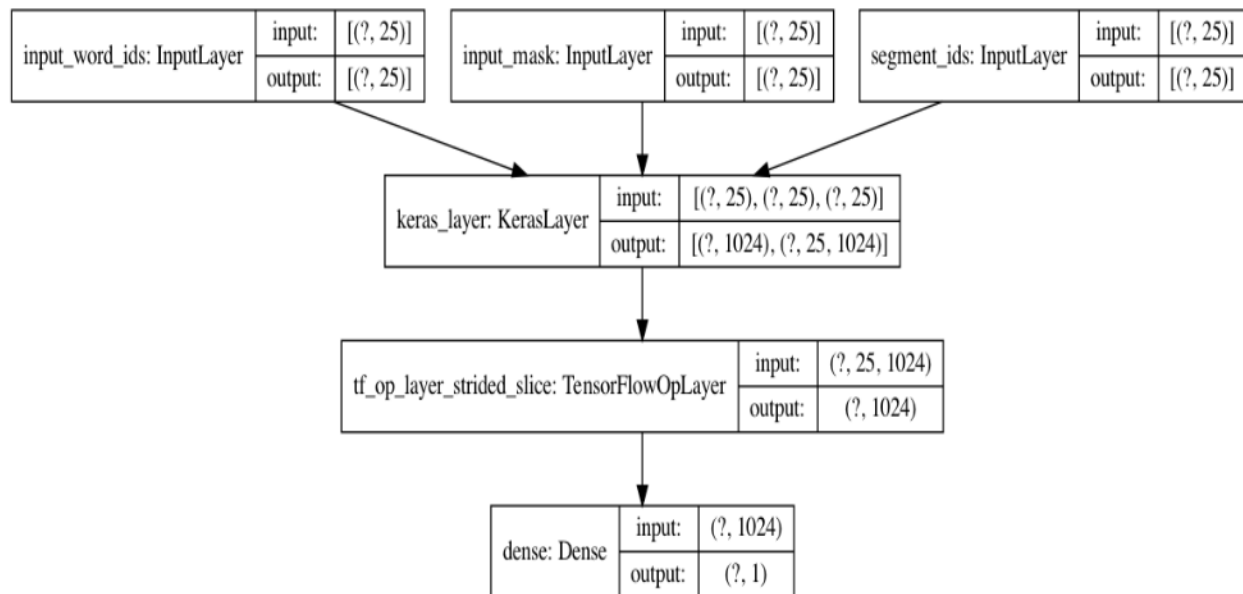


Figure 2: Layers for deep learning model using BERT

These are the final values for the hyperparameters I tuned:

	Value Resulting in Highest F1
Max Length	25
Optimizer	Adam
Learning Rate	2e-6
Batch Size	32

All other hyperparameter values are set to their default values.

This model contains 4 layers (3 main layers). The first is the input layer and the output gets fed into the second layer. The second layer is a special Keras layer that retrieves the contextualized word embeddings from BERT with the trainable parameter set to True. The BERT model that I chose to use contains 24 encoder layers with 1024 hidden units and 16 attention heads.⁴ This BERT model is the BERT LARGE one, which is huge. I took a slice of this output and it goes into the final Dense layer. Like the Dense layer in the GloVe-based deep learning model, it uses the sigmoid activation function to produce an output that is the probability of a given Tweet indicating a real disaster or not. This probability then gets rounded up or down to make a classification.

4. Results

Here are the results from my three different kinds of models on the different datasets:

⁴The BERT model that I used can be found here: https://tfhub.dev/tensorflow/bert_en_wwm_uncased_L-24_H-1024_A-16/2

	Training	Validation	Test
Naive Bayes	Acc: 0.5197 F1: 0.4800	Acc: 0.5178 F1: 0.4920	N/A
Deep Learning w/ GloVe	Acc: 0.8343 F1: 0.7849	Acc: 0.8082 F1: 0.7519	F1: 0.7924
Deep Learning w/ BERT	Acc: 0.8273 F1: 0.7838	Acc: 0.8399 F1: 0.7758	F1: <u>0.8221</u>

Kaggle held-out the labels for the test set and only computes the final F1 score, so that is why only the F1 score is the only evaluation metric shown for the test set. I do not have a F1 score for the test set using my Naive Bayes model because since the performance was so poor on the validation set, I decided not to submit the predictions to the Kaggle competition.

As you can see from the table, my deep learning model with BERT resulted in the best performance on both the validation and test set in both accuracy and F1 score. There is quite a jump between the validation F1 score and test F1 score, and this could be due to the fact that the validation set is pretty small (10% of training data) and is less diverse.

5. Results from the Kaggle Competition

At around the time of my last submission to this Kaggle competition, there was a total of 1654 teams.





//	iv41be		1.00000	1	4d
78	predictor!		1.00000	1	3d
79	Skanalytic		1.00000	2	1d
80	xinyu zhou 96		1.00000	1	6h

Figure 3: Top 80 people







321	Constantin Mihalache		0.82310	5	19d
322	Monika Stopyra		0.82310	2	1d
323	[Deleted]		0.82208	4	2mo
324	Andreas Tsouloupas		0.82208	17	1mo
325	Daniel Chen		0.82208	3	3d
Your Best Entry 					

Figure 4: Ranking from my BERT-based deep learning model





891	AnkitSingh21		0.79243	2	8d
892	Thota Seshu kumar		0.79243	1	5d
893	Rishabh Shah #2		0.79243	9	2d
894	Daniel Chen		0.79243	2	4d
Your Best Entry ↑					

Figure 5: Ranking from my GloVe-based deep learning model

My deep learning model using GloVe with an F1 score of 0.7924 brought me to 894th place and my deep learning model using BERT jumped my ranking up by 529 spots to 325th place. However, it should be noted that the ranking should be taken with a grain of salt because the ground truth to the test set is public, but isn't provided by Kaggle. The top 80 people received a perfect F1 score of 1.00, which should be nearly impossible. So, they must have just submitted the ground truth to this competition instead of submitting predictions from a classification model.

6. Related Work

Although it is difficult to find out what the highest achieved F1 score from using a classification model is, a person with a published Python notebook achieved an F1 score of around 0.8466, 0.0245 higher than the highest F1 score I achieved. After looking at this person's notebook, the author used BERT as well, but used a different kind of BERT model and preprocessed the data much more extensively.

Outside the scope of the problem for this Kaggle competition, Google also provides other BERT models that could be used for text generation, such as question answering. Stanford's GloVe provides other word embeddings as well that were trained on different corpuses, such as Wikipedia, Gigaword, and Common Crawl. These word embeddings might be better for problems like sentiment analysis on IMDB movie reviews. Other projects in the class, such as the one determining whether a Tweet was written by a robot or human and the one detecting fake news, are also very similar to this project.

7. Conclusion

The quicker agencies and organizations find out about disasters, the more time people have to evacuate, plan, and combat these disasters. Making sure Tweets actually indicate a real disaster is very important and saves time. From my results, the deep learning model using BERT achieved the highest F1 score, beating the deep learning model that uses GloVe by about 0.0297%. This is mainly due to the use of contextualized word embeddings.

Some of the limitations I faced while training my models were: a lack of memory space, no GPU, and very little training data. I trained my Naive Bayes model and deep learning model with GloVe on my local machine, however BERT was too big to handle on it. So, I used Google Colab for that model instead. Understanding Tweets is quite challenging for models because they are relatively short, the language used is different (more slang and abbreviations), and negations, such as 'not so bad' and 'not good'.

To improve my models in the future, increasing the size of the training data is very important. This will bring even more diverse Tweets and allow the model to see more variety. Fixing all the typos in the Tweets, expanding all the acronyms, and expanding all the contractions could also lead to some improvements to the F1 score as well.