Tweets, Real Disaster or not?

Work by:
Mohamed Khanafer
Aayush Kejriwal

Table of Content

1. WHAT THIS DOCUMENT IS

For better understanding our work, we here summarized our findings and thinking based on all the work done in the Jupyter notebook. We thus advise to read this guide first before looking at all the details in the Notebook, which is much more detailed.
In case you have any question in any point we argue or in running our code, please do not hesitate to contact us.

2. PROBLEM INTRODUCTION

We have a dataset of almost 11,000 tweets. The objective is to classify these tweets on whether they are about real disasters or not. The complication comes because many tweets have words that would imply a disaster but are used metaphorically. Therefore, merely identifying the presence of such words would not guarantee that the tweet is in fact referring to an actual disaster.

We have just three pieces of information to do this, the entire actual tweet, the key word in the text and the location from where the tweet has been made.

3. EXPLORATORY DATA ANALYSIS

Following are the key takeaways from our EDA. A more detailed version can be found in the jupyter notebook:

1. The text column does not have missing values, but there are repeated texts. This could be due to retweets.
2. Common stopwords in both disaster tweets and non-disaster tweets are quite identical. Therefore they might not be needed in the model.
3. The location data has 1/3rd missing values and also some vague locations such as ‘everywhere’, ‘worldwide’, etc. Also, the granularity of the data is not constant, with some tweets having a country as location and some having cities. Due to these reasons it is unlikely to be useful.
4. Looking at the keywords columns reinforces our earlier problem. While the keywords in disaster tweets are all something we associate with disasters, such words are also present as keywords in non-disaster tweets. Therefore relying solely on key-words will not allow us to accurately distinguish between the two classes.
5. Overall, our dataset is relatively evenly balanced between disaster and non-disaster tweets. Therefore, methods such as undersampling or oversampling will not be necessary.

4. MODELS

4.1 APPROACH I: Basic Processing Methods

1. We first started out with a baseline model using different classification algorithms. We used the countvactoriser function on the raw text to get a bag of words. This is the most simple technique and is used without data cleaning. We will use this to judge the merit of additional methods for solving this problem. At this stage, we found that the logistic regression gives us the highest score, with a F1 of 0.7924.

2. We then tried to clean the text, by converting everything to lower case, removing URLs, HTMLs and other special characters. The sources for the code we used are in the notebook. We used the same bag of words method and a logistic regression to judge the improvement due to text cleaning, and got a slightly higher F1 score of 0.7955.

3. At this stage, we tried a few more basic things, such as using tokenization, removing stopwords, and creating unigrams and bigrams. This once again improved our score to 0.8118.

4. As mentioned in the EDA, some of the tweets had duplicates, and we found that these duplicates did not always have the same target class. We tried to mislabel these manually but it led to a reduction in the score. Therefore, we will not use this step in our further models.

5. Up to this point, all our models relied on just the text column. We have not been using the keywords column in making predictions. We tried a few different methods to integrate this column in or model.

   At first, we just calculated the conditional probabilities of the tweet belonging to class 1, given the keyword. For those keywords that had such conditional probability to be higher than 0.9, we manually changed the prediction of our model to 1, even if it predicted a 0. This led to an improved score as well.

4.2 APPROACH II: Tokenization, Stemming, Lemmatization and TF - IDF

1. We then tried to use a little more advanced NLP concepts to try new models. We tried both stemming and lemmatization to get root words, and use these root words in the bag of words method to build a model. We find that lemmatization is a more appropriate method since it gets to the actual root words instead of just eliminating suffixes and prefixes as in stemming.

To this, we also added the key words, but this time we added it to the text as an additional word. This way the importance of the key word would be greater in the model.

We also switched from countvectoriser to TF-IDF, since it considers the importance of the word instead of just the count.

All of this led to a higher score of 0.8169 using a SVM.

2. In our next model, we used word embeddings instead of the bag of words approach. We anticipate this to lead to a higher score since it represents words as vectors, which allows similar meaning words to have a closer vector than completely unrelated words. This way we add complexity to the model as it is not considering each word as a standalone object, but also considers whether other words are similar or not. Again, we did this on the cleaned text. However, the result was surprisingly lower than the bag of words method. We got an F1 score of 0.795.

3. In the 8th model, we try adding POS tags. This does improve the score from the previous one, but still is not as high as our other models where it was not included.

4.3 APPROACH III: Better Preprocessing with correction of Spelling Mistakes

At this stage we looked at our processed text once more. Despite the earlier cleaning steps, we found that there were still a few errors. These were due to spelling mistakes and due to contradictions (eg: can't was treated as can t instead of can not). Therefore, we added functions to correct these errors. These were done in addition to the earlier cleaning steps.

We ran all the models discussed above again, with this cleaner data, but our scores were surprisingly equal to or less than the ones we got earlier. Despite this, we think that these additional cleaning steps are more logical and represent the text better, and therefore despite not having better scores, we will use these steps in our next approach.

4.4 APPROACH IV – BERT Models

After combining many traditional NLP processes, we decided to take it a step further by exploring some more advanced libraries.
One of the most used tools for numerous applications in NLP is Google's BERT (Bidirectional Encoder Representations from Transformers).

What makes BERT special is that it is a deeply bidirectional model. Here, bidirectional means that BERT is able to learn information coming from both the right and the left side of the specific context of the word during the training phase of the model.

As per our understanding, this takes sentences from a document as an input and generates contextualised sentence embeddings as an output. These embeddings would be vectors that capture the relative semantic meaning of the sentence.

When it comes to its architecture, BERT builds on top of Transformers. Here, instead of going into the details of the model's architecture, we use the [simpletransformers library](https://github.com/ThilinaRajapakse/simpletransformers/) which allow us to quickly train and evaluate Transformer models. They give access to pretrained BERT models that we will use below.

For the models under BERT, we planned to use 5 stages of our dataset to see how different levels of cleaning and normalisation impact the score. These were:

1. Raw Data
2. Text after cleaning as discussed in approach 1
3. Above cleaned text with normalisation
4. Above cleaned and normalised data with relabelling of retweets as in Approach 1.4
5. Data after further cleaning as in approach 3
6. Above data with normalisation

Out of the above models using BERT we found that the 5th one, which has the text cleaned but not normalized, gives us the best score. This was also our overall highest score, with a F1 of 0.82822. However, the remaining BERT models still outperformed earlier equivalent models in most cases.

Based on the results, we believe that relabeling the data has consistently been reducing the score, and therefore should not be considered.

Also, a possible reason for normalization not improving the score with BERT is that by only keeping the root words in the sentences, we are forcing the vectors of the sentences to be closer since the differences caused by word forms become negated. These differences are perhaps important distinguishing factors between disaster and non-disaster tweets.

5. FURTHER OPTIONS

In our models we have rarely considered information outside the text column. Therefore, our predictions are based on a single input. We could however generate certain additional features such as lexical diversity of the tweets. This might not necessarily improve the score, but it could capture certain subconscious typing styles of people when they tweet about something serious like disasters. We could also use the length of the tweet in characters and words, and also Boolean variables representing the presence of URLs. It is not necessary but there could be a difference in the percentage of disaster tweets that have a

URL to non-disaster tweets, for example disaster tweets may usually accompany a news article reporting on the disaster.

Such additional features could be used to build a separate model, which could be ensembled along with the model we built above, to get a new set of predictions.

6. SUBMISSION TO KAGGLE

Our submission was the model with 0.82822 accuracy, positioning us in the top 15%.