Brendan Magdamo, Brendan Manning, Stephanie Scheerer
Projects in Data Science
5 April, 2021

3.2 - Data Report

Data Cleaning and Preprocessing

During exploratory data analysis, several preprocessing combinations were used and tested for effectiveness. Among the things considered were emojis, links to other pages, stop words, and punctuation. Additionally, all text was converted to lowercase in order to create uniformity across all instances. In our original pass through the dataset, all of the things listed above were completely removed from the dataset. Our baseline model did not rely at all on Natural Language Processing, so we wanted to strip the text data as much as possible so as not to confuse the naive model. As our model became more complex, we ran some tests in order to test the effectiveness of keeping some of these things in the data.

Since NLP techniques work so well because they are able to parse word dependencies and context, we decided to keep stop words in so that the models had more text from which to derive context. We ran our model both with and without stop words included and the version with stop words performed markedly better (about 0.01 higher in F1 score, which is significant in relation to the Kaggle leaderboard).

Next, we turned our attention to emojis. Unlike links, and to a certain extent punctuation, emojis can contain useful information for prediction. Emojis are represented in text by their UTF encoding, which we took into account when removing them the first time. Our first strategy was to extract the most popular emojis and then create a multi-hot or one-hot encoding for each instance that had one of the emojis embedded in the text. This encoding would then be appended to the output of our NLP model in order to be run through a traditional classification algorithm such as logistic regression. However, this proved to not be necessary, as the language models we used (BERT and RoBERTa) take into account the UTF encoding of emojis when doing translation tasks. This meant that we were able to keep emojis in our text data to preserve extra meaning, but we did not have to do any feature extraction in regards to emojis, as the language models took care of that under the hood.

The next determination we made was in regards to punctuation. The punctuation library we used (string.punctuation) included symbols such as “#” and “@”, which can be useful when dealing with Twitter data. The context of a word can change depending on if it is used in a hashtag or not. Because of this, we experimented with keeping the punctuation in. However, keeping punctuation in actually decreased the F1 score slightly, most likely because there was extraneous punctuation that hurt the model more than keeping the hashtag information helped. Creating a custom punctuation dictionary was considered, but was ultimately deemed unnecessary, and punctuation was kept out of our final model.

Links were the easiest determination for us to make, as links do not usually provide any helpful information in these kinds of sentiment analysis tasks. There are a wide variety of links in the dataset, as people often link to things like images, news articles, etc. on Twitter. The unique nature of web addresses also makes it difficult to find patterns in links that may aid in prediction accuracy. Links were kept out in our final model as well because of this.

When creating our final model, two new considerations were made to improve data cleaning: word abbreviations and non-ASCII characters. Using an abbreviations dictionary found online, we were able to replace all abbreviations with their meanings in the Tweets. This includes things like replacing "$" with "dollar" and "asap" to "as soon as possible". Here, the fact that everything was converted to lowercase helped, as we did not need to make the find-and-replace algorithm case sensitive. Replacing abbreviations not only makes the Tweets more readable to a person, it may also help an NLP algorithm parse the context of a Tweet because its pre-trained libraries are more likely to contain the phrase "as soon as possible" rather than "asap". Of course, it's entirely possible that the pre-trained library contains both, but we did achieve better results when taking this step (replacing abbreviations in conjunction with removing non-ASCII characters improved the F1 score on the test set by about 0.02).

The removal of non-ASCII characters was also something that we used to improve our score; these characters are likely to have little to no meaning to a BERT model, so it made sense to remove them. This was not something that we originally considered because we did not realize the scope of the problem. A manual glance through the text feature showed a fair amount of non-ASCII characters that needed to be removed. We believe that some of these may be emojis that did not render correctly when the Tweets were scraped. This meant that the model wouldn't be able to correctly identify these strings as emojis, so removing them helped our model in achieving a higher F1 score.

Feature Extraction and Augmentation

Our main focus was the cleaning and preprocessing part of the text data, but we did attempt a few feature extraction techniques, namely the multi-hot encoding of emojis and back translation.

The multi-hot encoding was described earlier when discussing emojis, so back translation will be discussed here. Back translation, which is the process of translating the text to a different language and then translating it back to English, is a common text augmentation technique used to slightly alter the text and possibly reveal new information for the model. Our original attempt at back translation was through Google Translate's API. However, there turned out to be a problem, as a new change in their API changed the way they tokenized words and did not allow for batch translation. There was an alteration somebody had made to the code available in a Github repository, but cloning the repo did not work when done in Google Colab. Next, we tried a new package called "google_trans_new" which did work, but did not allow for the user to pass a list as the argument, so we could not batch translate our dataframe. The last attempt we made

was using a model called MarianMT, which can be found in Hugging Face's transformers package. MarianMT allows for translation to-and-from languages and we downloaded the pre-trained models that allowed us to translate from English to Romance languages and then from Romance languages back to English. We were attempting to choose two languages to try back translation with: one with similar syntax to English and one that is completely dissimilar to English in order to see which model performs the best. This model worked when applied to a small list, but when applied to our dataset, the GPU ran out of memory. Models like this take far too long to run on CPUs, so we are currently working on either implementing this model in batches or increasing GPU memory in order to run all of our text at once.

Trends In the Data

During exploratory data analysis, we considered several aspects of the data that will be discussed here. First we considered the makeup of the Tweets themselves rather than their associated meaning. This is considered in Figures 1 and 2, where the Tweet length and number of words per Tweet are considered for both classes. All figures can be found at the end of the document. It is clear in both figures that the distributions are approximately equal across both classes. Had they not been, we may have had to consider some additional preprocessing in order to maintain some similarity across classes so that our model did not end up overfitting.

Next, we considered popular stop words for each of the classes. This visualization is seen in Figure 3, and was done more out of an abundance of precaution than anything else. Stop words are usually equally distributed between documents, as they are just the most common words in the English language. Figure 3 was created just to make sure there were no large discrepancies in the texts, and also to see if there was a particular stop word that may be indicative of a Tweet being labeled as a disaster or not.

Perhaps the most useful visualization is seen in Figure 4. Figure 4 shows the 20 most common keywords for each of the classes. Keywords in this dataset are essentially just tags for each Tweet that describe the type of disaster that may or may not be being described in the Tweet. This is useful because it helped track keywords for class 0 that may be confusing for the model. For instance, "screaming" was among the most frequently used keywords in class 0, which is extremely ambiguous in its use case. Screaming could very well be used in a disaster scenario, but it is much more likely to be used in a more relaxed, non-literal setting. Screaming was the eleventh most frequent keyword for class 0; class 1's eleventh most frequent keyword was "nuclear disaster" which is much less ambiguous, and details the difficulty a model may have in classifying some of the non-disaster Tweets.

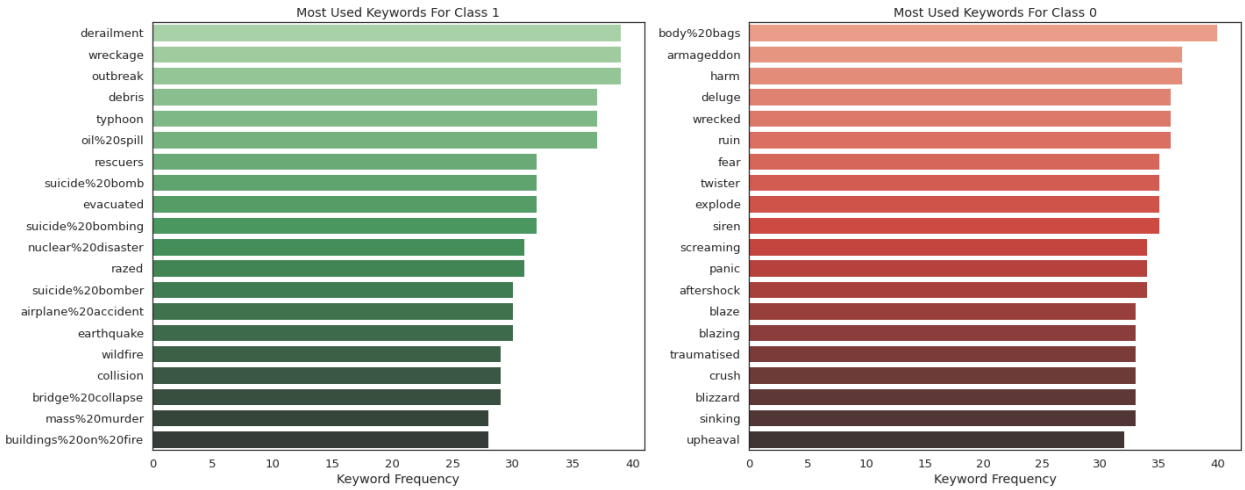
Another useful trend we observed was the frequency of different types of emojis across the training set, which can be seen in Figure 5. This graph showcases the difference in emoji use between the classes. For instance, the crying laughing emoji is more prevalent in the non-disaster class, while the actual crying or fire emojis are used more frequently in the disaster class. This confirms our suspicion that emojis may contain useful information for prediction. An

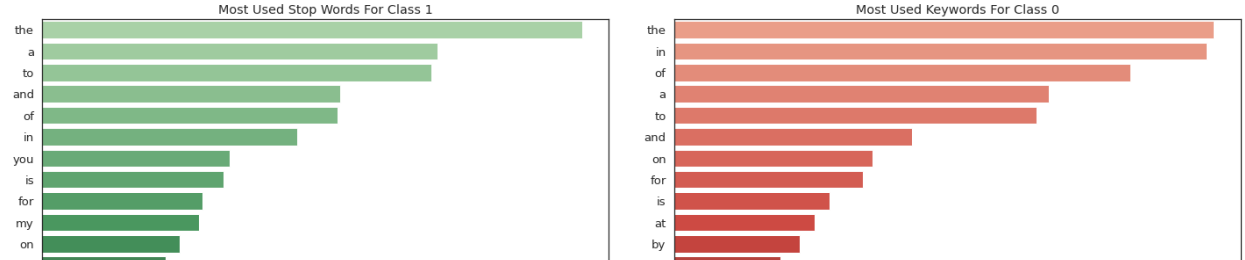
even distribution of emojis between classes would indicate that emojis were relatively useless for prediction purposes, but Figure 5 shows otherwise. It is important to note, however, that the vast majority of emojis in the dataset are ones which are used very infrequently. The NLP algorithms will presumably learn to only pay attention to those specific emojis.

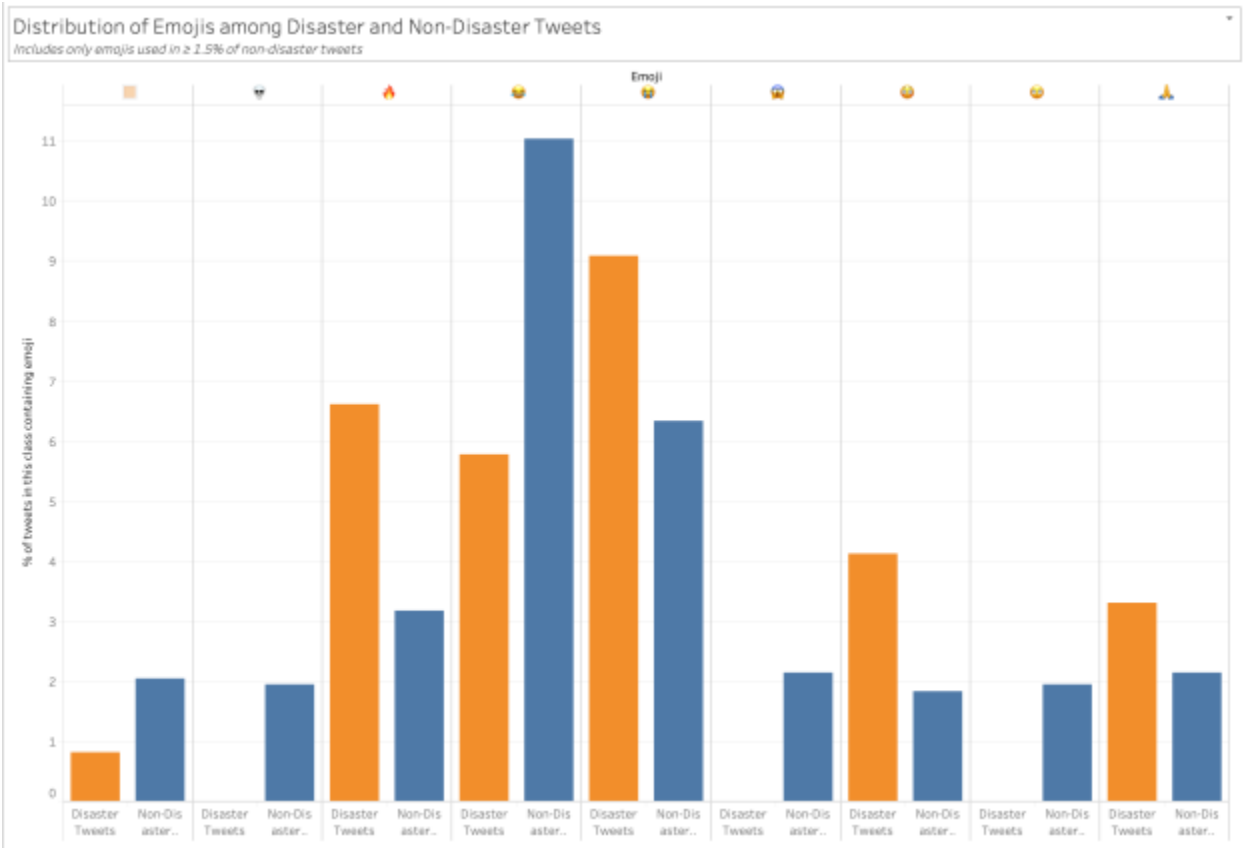
Overall, these observed trends helped guide our preprocessing techniques. Had there been conspicuous imbalances in the classes in terms of Tweet length and words per Tweet, we may have had to remove some extraneous information from some of the longer Tweets. We may have also decided to keep stop words out of Tweets if that had helped to bring the distributions closer together. The other figures we made helped to guide our understanding of how the model may end up misclassifying different training examples. The keyword and emoji visualizations show us what information is in the training data that may confuse an NLP model, and we were able to observe our F1 score for those specific keywords to make sure the scores did not lag too far behind our overall F1 score.

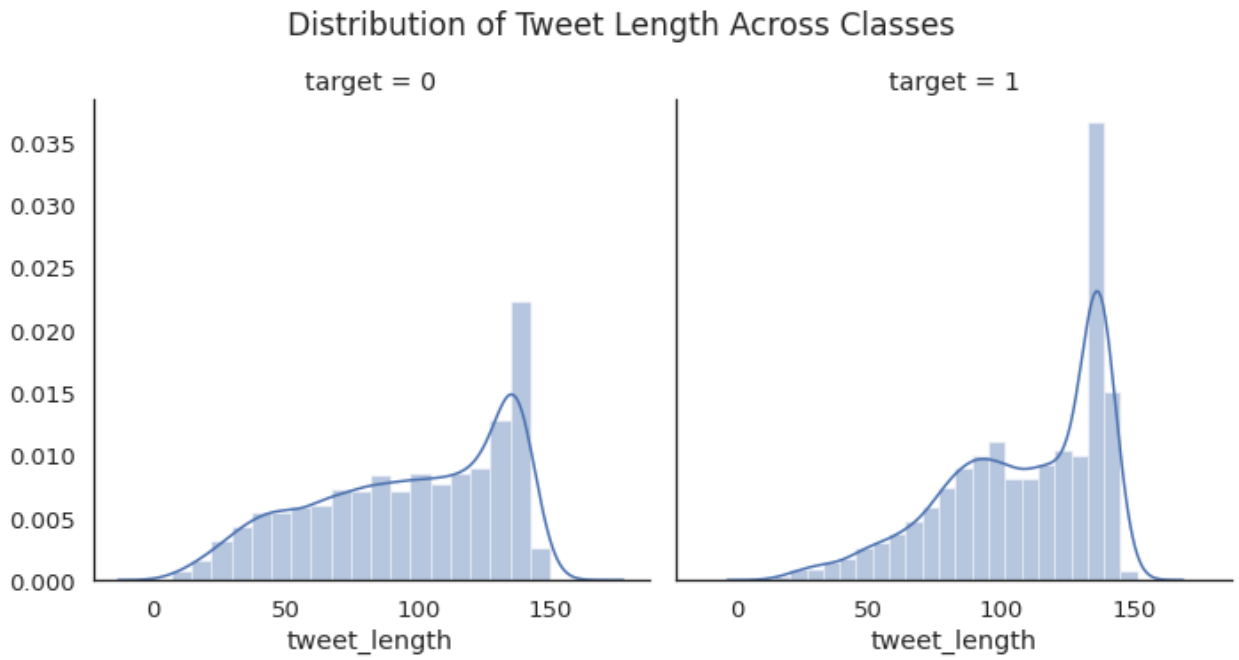
Possible Issues

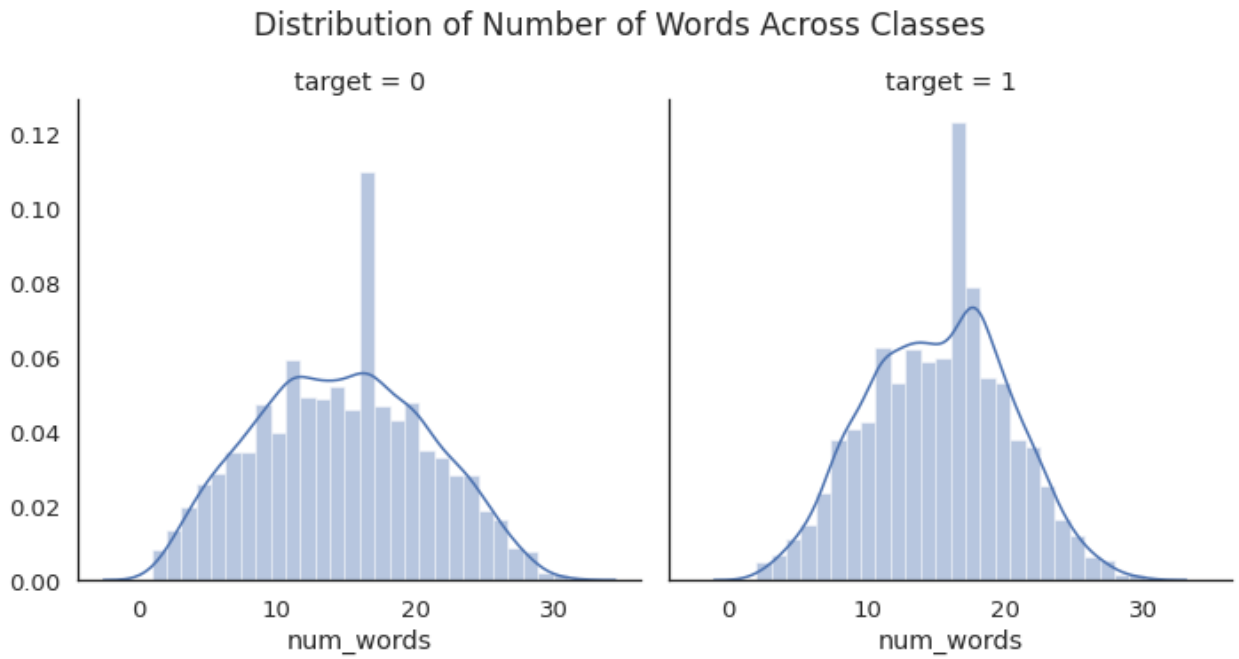
In our exploratory data analysis, we came across two potential problems: some emojis looked like they didn't render correctly and some examples looked like they may be misclassified. The emoji problem was rectified by removing all non-ASCII characters so that these misrepresentations did not confuse the model. This is also less of a worry than it may seem since some systems and applications don't properly render some emojis, despite them working perfectly well "behind the scenes." For the misclassified examples, there was not a lot we could do, as we would have to manually comb through all of the data to remove examples. We decided to leave the examples, as we also did not have access to the original Tweets in order to see the context or images that may have been provided at the time.






Distribution of Emojis among Disaster and Non-Disaster Tweets
Includes only emojis used in ≥ 1.5% of non-disaster tweets
Emoji
% of tweets in this class containing emoji
0
1
2
3
4
5
6
7
8
9
10
11
Disaster Tweets
Non-Disaster..


Distribution of Tweet Length Across Classes
target = 0
target = 1
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
50
100
150
tweet_length
tweet_length


Distribution of Number of Words Across Classes
target = 0
target = 1
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
num_words
num_words