Arun Agarwal
Professor Belorkar
CIS 4496 - Projects in Data Science
Honors Contract
April 30th, 2023

Data Report - NLP Disaster Tweets

Data Introduction:

The competition data consists of 10,000 hand-classified text tweets, split amongst training and test data files. The data was gathered by Figure-Eight Website and uploaded to Appen as a pre-labeled dataset (hand classified as disaster or non-disaster) (3). The competition includes three files, the mentioned training and test csv files and a sample_submission csv file to be used for submission format. The train and test datasets include 5 columns:

- id: the identifier for each tweet
- text: the tweet
- keyword: the key word from tweet (many are blank)
- location: the location the tweet was sent from (many are blank)
- target: the classification label of disaster (1) or not (0)

The data appears to change over time, as previous comeptitors were seen to have different data than what is currently being used by the competition.

Which data cleaning and processing techniques, if any, were used to make the data usable?

The main task for this competition involved the cleaning and processing techniques performed to make the data usable. First, the libraries used for preprocessing included pandas (to view the data frame), numpy, string (for various text cleaning items), re (for regex), and nltk (to obtain a list of stopwords). After reading in the raw data and performing some exploratory data analysis, I realized that the text was not cleaned at all; that is, it was taken straight from user's tweets and inserted into the data file. I began by examining the preprocessing steps taken by other competitors in the competition as well as the previous team who worked on this competition (1, 2), as I have never performed text preprocessing in my undergraduate education and needed some assistance to determine good starting steps. The previous team for this course made a list of abbreviations and their expanded word forms, which I utilized to expand the abbreviations in my copy of the data as well (2). I then lower-cased all characters/words, as the sentence case version of the tweet did not seem to provide any additional value. It also made more sense for "Upper" and "upper" to represent the same word rather than different instances, for example. Next, based on one of my findings from the exploratory data analysis, it appeared that emojis did not get loaded into the data frames correctly and were thus just causing issues with the accuracy of the model. I decided to remove all emojis and special characters using their Unicode values; however, it is likely that they could have provided value if the Unicode characters were translated into word representations using an online dictionary. Similar to the

emojis, one of the charts from the exploratory data analysis revealed that punctuation did not provide any additional value and could be confusing at times; therefore, all punctuation was removed. From here, the main manual preprocessing step occurred, thanks to another user on Kaggle (2). This user manually went through the text data to expand contractions and acronyms, replace slang and informal abbreviations, fix typos, and write out usernames and hashtags in separate words (2). This user made his code publicly available on the Help page of the competition; therefore, almost all teams/competitors make use of this code block for the manual cleaning of their data, including me, as it would be futile to not make use of the manual effort of another. After this, the exploratory data analysis also helped reveal that URLs did not provide additional value and were thus removed from the text; it should be noted that if the URLs were manually expanded, there may be words within them that would help to indicate the credibility of the tweet. I also removed HTML beacons and non-printable characters, as suggested by others. After all these preprocessing steps, I remeasured my model performance and noticed I was still performing far below other competitors; thus, I decided to explore the data further. This led to my decision to do more manual cleaning, for which I wrote my own function for. After performing all this preprocessing, there was a lot of whitespace produced, which I removed so that there would only be one space in between words. Finally, all this preprocessing led to duplicate instances to be found in the data that did not show up before preprocessing. In context, this makes sense, as humans tend to copy tweets of other people often rather than make original content (just switching up a few words or adding a bit of text). With these duplicate instances, I manually relabeled them, just as other competitors in the competition did. Considering that all these samples were manually labeled by humans to begin with, it seems fine for me to make decisions myself of what the correct labels for these samples should be. It also seemed obvious when actually reading the tweets. Interestingly, my preprocessing steps led to approximately 50 more mislabeled instances than most other competitors, which is most likely due to the additional removals that I did that most other competitors did not. I ended my preprocessing phase by creating four different versions of the dataset, one with mislabeled and duplicates samples kept in, another with no mislabels but with duplicates, a third with mislabels and no duplicates, and a final with no mislabels nor duplicates. Only the last of these four processed datasets could be tested in our final model due to extensive runtimes and minimal work time. It should also be noted that, while I filled the null values in the keyword and location columns with “no_keyword” and “no_location”, respectively, these columns ended up being removed from the data as there were simply too many blanks.

Which feature extraction techniques, if any, were used to generate new features to the dataset?

Rather than focus on the feature extraction techniques, I will mention the 12 features generated from these techniques. This is because the feature extraction was simply done using the apply function and various functions available in the string and nltk libraries. The first of these features was word_count, which was simply the number of words in the text. A graph of

the distribution of this feature revealed there was a noticeable difference in words between disaster and non-disaster tweets. The next feature was unique_word_count, to get the number of unique words in the instance. This feature displayed a normal distribution with a noticeable difference between disaster and non-disaster tweets, similar to the word count. Next, stop_word_count counts the number of stop words in the text instance, measured using the stop words dictionary from nltk. The distribution was relatively the same for the two classifications except for smaller counts of stop words, in which more disaster tweets existed with less stop words. In context, this may make sense, as I would expect disaster tweets to be from sources that use more formal language. Moving on, url_count counted the number of URLs provided in the instance. As mentioned in the previous section, URLs were removed based on the extremely similar distribution between the two classifications. The next feature created was mean_word_length, which is the average character count for the words in the text instance. We would expect disaster tweets to have larger mean word lengths, which was exactly what was seen when I plotted the distribution. Therefore, this was another feature that could help to distinguish between disaster and non-disaster. We also had measured the mode word length as another feature and noticed the same pattern seen for mean_word_length. Then, char_count was simply a count of the number of characters in the text, providing similar value to combining word_count and mean_word_length. I ended up noticing that disaster tweets had a larger character count more often than non-disaster tweets, indicating the feature can provide value. Similar to url_count, punctuation_count and emoji_count get a count of the number of punctuation marks and emojis in the text, respectively. Viewing these distributions also led to the decision that their respective characters did not provide value and were removed. Moving on, hashtag_count and mention_count were counts of the number of hashtags and mentions in the text, which both did not provide additional information. The final feature created was slang_word_count to count the number of slang words in the text. In context, those tweets containing more slang are more likely to represent non-disaster tweets; however, I was struggling to find an appropriate list/dictionary that contained a lot of possible slang words, so I could not fine-tune this feature enough. It therefore could not provide any additional value. I also wanted to created features based off the unigrams, bigrams, and trigrams, as other competitors noticed interesting patterns with these, but I unfortunately ran out of time to do so. One large mistake I believe I made with these features is not including them in my final BERT model. At the time, I did not realize how much time would end up getting invested into making the BERT model work, as well as how long it would take to run one instance. Thus, by the time I finished one model run, I needed to finish up working on the code and instead spend the remainder of my time on the writing portion. For the future, I should not only try my other three processed versions of the data, but I also need to try these versions with the valuable features engineered included.

What are the trends in notable features in the data?

Various trends in the data were noticed thanks to the charts created in the exploratory data analysis. First, by examining the percentage of missing values in each column of both the training and test data, I learned that the datasets are most likely taken from the same sample. This is because both sets had 0.8% of keyword instances and 33% of location instances missing. Next, by graphing the most common locations from the data, I was able to notice a lot of locations that seemed dirty/insensible. Therefore, I realized that the locations were not automatically generated but rather user-inputted. I was provided the information that the keyword column was user-inputted, and it did not appear to be as dirty. I also noticed that every single keyword that existed in the training dataset appeared at least once in the test dataset, again indicating that the training and test data came from the same sample. Examining the most common keywords allowed me to also see that this column could potentially provide value, as many of those words could only be used in one context (in contrast to "ablaze" that can be used in multiple, for example). Still, the dirtiness of location and the large number of blanks in both keyword and location led to their removal from the data. Moving on, as described in the previous section, distributions of the meta features led us to make conclusions about which features could be helpful in distinguishing between disaster and non-diaster tweets. These distributions overall helped us realize disaster tweets are written in a more formal way with longer words compared to non-disaster tweets because most of them are coming from news agencies. Furthermore, non-disaster tweets have more typos than disaster tweets because they are coming from individual users. All the meta features had their distributions graphed with a comparison of classification (disaster/non-disaster) and dataset (training/test). Since all meta features demonstrated almost identical distributions for the two datasets, I was able to further validate my claim that the data comes from the sample. In general, url_count, hastag_count, and mention_count could not provide value while word_count, unique_word_count, stop_word_count, mean_word_length, char_count, and punctuation_count could provide value thanks to the very different distributions for disaster and non-distaster tweets. One final thing I looked at was the distribution of the target variable. Specifically, 57% of text instances represented fake disaster tweets and the remaining 43% were real disaster tweets. Understanding that a slight class imbalance exists within the data is important for hyperparameter tuning in the model (led to the decision to use stratified K fold instead of normal cross validation, for example).

How do the observed trends relate to your project problem statement?

I answer this question in the previous section. Overall, the observed trends, such as the cleanliness of the keyword and location columns and the distribution of the target variable helped in deciding the best course of action during the preprocessing phase. Therefore, they relate to my project problem statement by helping me to build a model that better distinguishes between disaster and non-disaster tweets. The engineered features helped in better understanding patterns

in the data, which also led to certain preprocessing decisions and, in turn, a better model. However, the omission of these features from the BERT model may have led to a slightly worse performance compared to the top performers in this competition.

Does the data foretell any issues that may arise in later stages of the project lifecycle?

The data does foretell some issues that would arise in later stages of the project lifecycle. While many of have already been mentioned in the prior sections, I will repeat the most important findings here. First, the large amount of missing data foretold that they would lead to worse model performance later on, so they were removed. Next, the almost identical distributions existing between training and test data foretold that the model should not have difficulty with performance on the test data. Moving on, the large amount of mislabeled samples foretold that the model may have worse performance, as having two exactly identical instances with different target labels is sure to confuse the model. Finally, the large amount of duplicate instances in the data foretold the model may overfit if the data is kept in; thus, the BERT model we tested had duplicates (and mislabels) removed.

References:

1. Previous Group GitHub
2. Gunes Evitan Kaggle Competitor Code
3. NLP Disaster Tweet Data Source
4. NLP Disaster Tweet Kaggle Competition
5. Transformers Modeling BERT
6. How To Train a BERT Model - Towards Data Science
7. GitHub NLP Tokenization Code
8. My GitHub Repository