Natural Language Processing: Classifying Real and Fake Disaster Tweets

Problem Statement

Humans can easily understand the meaning behind a line of text; whether that text is happy or sad, negative or positive, and the context behind that text. However, it is tricky for a machine to understand the same thing. In this day and age, Twitter and other online websites have become an integral part of quickly identifying a disaster of any kind and relaying the information to the public. This can be useful to an endless number of clients, such as news outlets, disaster relief agencies and other grassroots organizations, active politicians on Twitter, law enforcement, the fire department, hospitals, and many more. It is also important to have real-time information on any such events to be able to react to them in a timely manner. Therefore, the goal of this project is to build a model that can predict, as accurately as possible, whether some text is referring to a real problem or disaster, or whether it simply contains words that make it seem like it is. This model would speed up the process of identifying an actual disaster in real-time, based on the text a tweet contains.

Dataset

The dataset that models will be trained on is a series of tweets containing text. The dataset was obtained from Kaggle Competitions and contains 7,613 tweets. The columns of the dataset are as follows:

1. **ID** - the id number of the tweet.
2. **Keyword** - the keyword that causes the tweet to be flagged as a disaster tweet.
3. **Location** - location of the tweet.
4. **Text** - the words contained in the tweet.
5. **Target** - either 0 or 1, indicates whether the tweet is real (1) or fake (0) disaster tweet.

Data Wrangling and Cleaning

This part of the project involved several steps taken to clean the text data in order to make it readable by and consistent enough for the model to make accurate classifications. The tweets initially contained many issues, such as punctuations, capitalizations, misspellings, links, and emojis. I created a function that would solve each of these problems, first by making all text lowercase only, removing text in square brackets, links, all punctuation, numbers, and all emojis. For each of these steps, I reassigned the text to what I wanted it to look like by using the re.sub() function. The text was now consistent and clean enough for our model to be able to read each tweet and make as accurate of a prediction as possible. After that was done, I proceeded to remove all English stop words from the text, in order to eliminate all repetitive and unimportant words for our prediction. Stop words removed include words such as “the”, “is”, “of”, “be”, “so”, and “and”.

Exploratory Data Analysis

The distributions of real disaster tweets (target = 1) and fake disaster tweets (target = 0) was explored in order to get a better idea of what the data looks like, and to check for any outliers. The bar plot can be seen below.

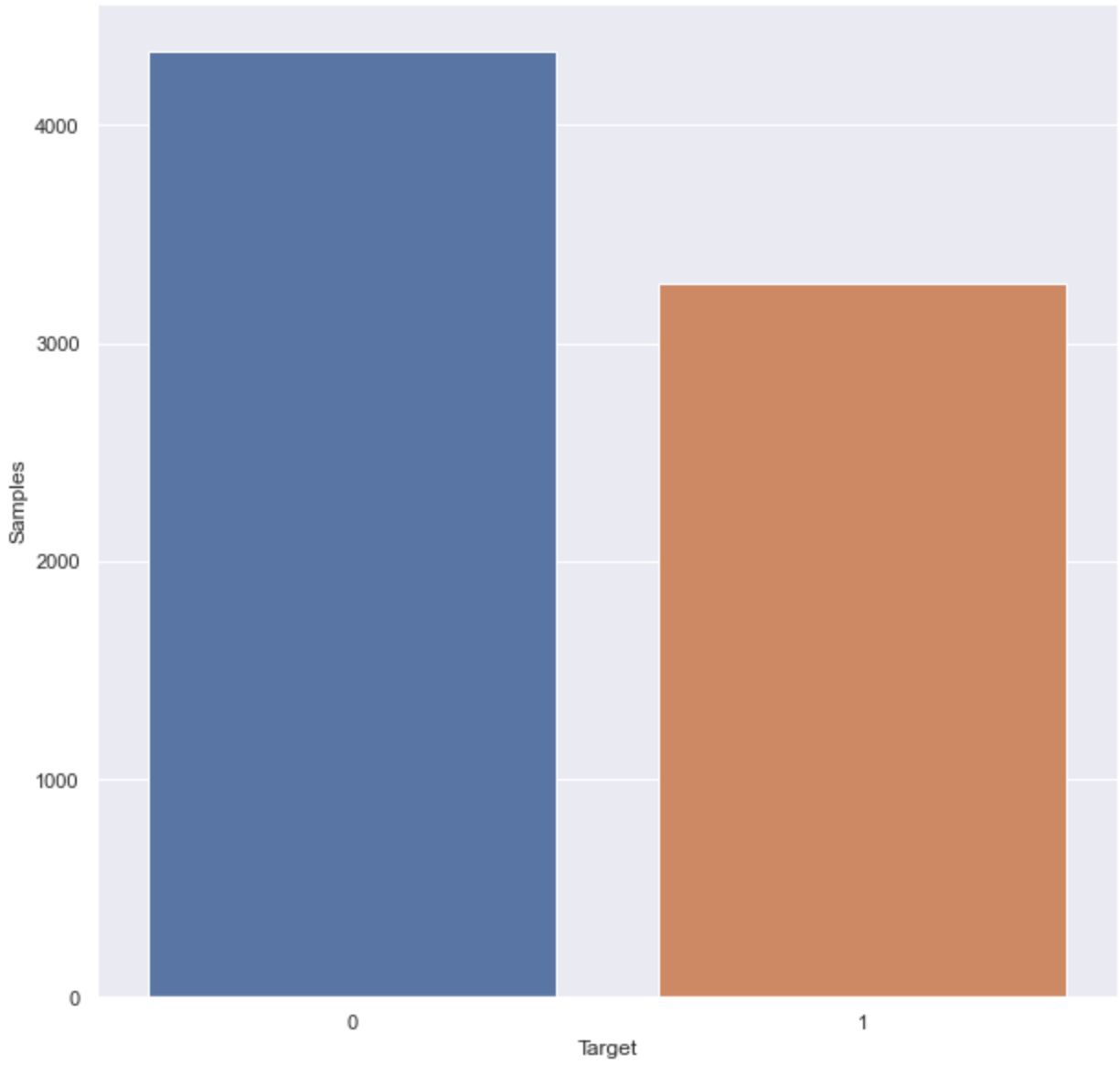


Figure 1

The figure above shows that there were more fake disaster tweets than real ones in the dataset by about 1,000 tweets. The fake tweets made up about 57% of the data, and the real tweets made up about 43% of the data, so the difference in distributions is not too large. Next, we looked at the length of characters and words in each disaster tweet type. The plots can be seen below.

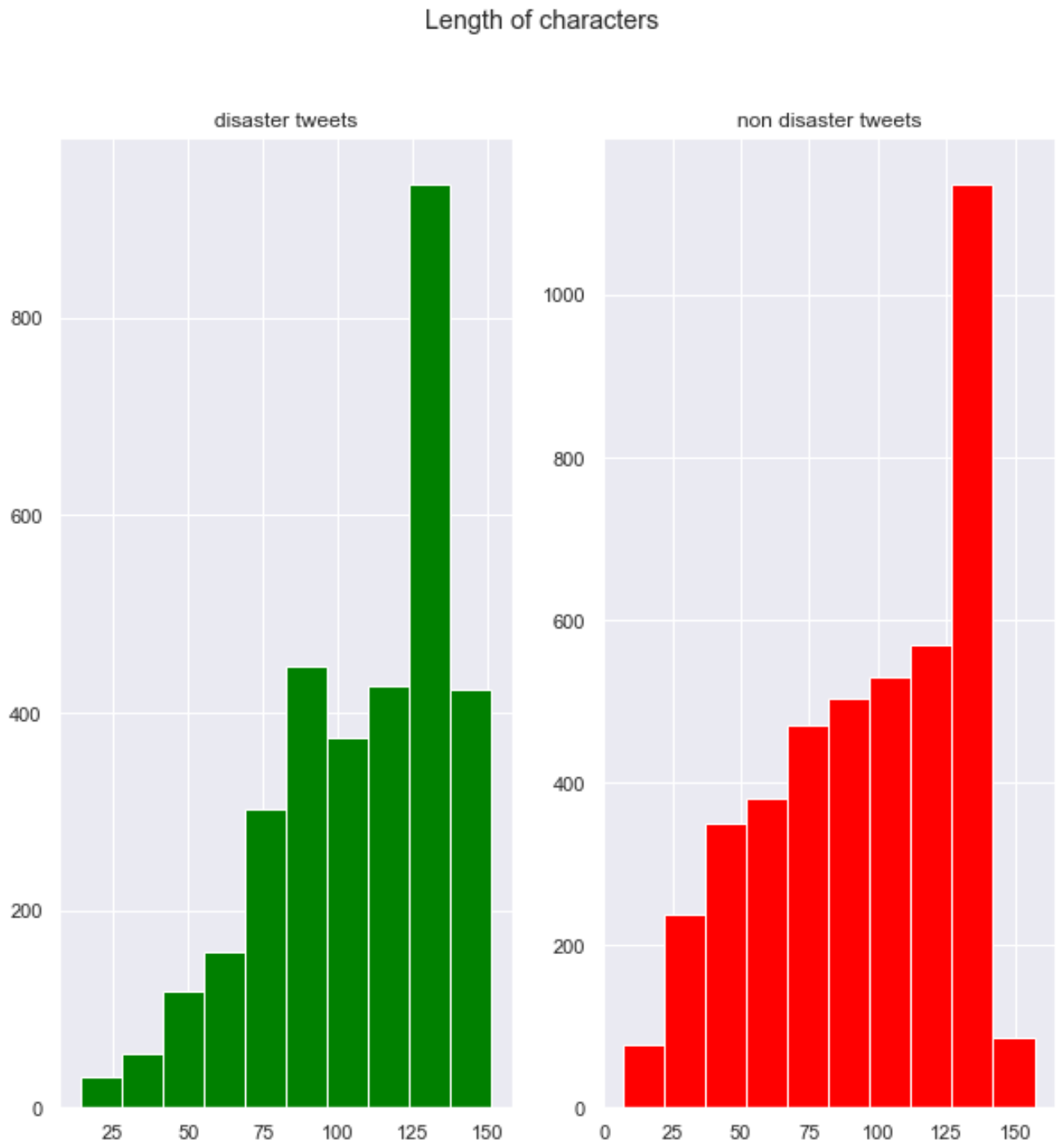

Length of characters
disaster tweets
non disaster tweets
0
200
400
600
800
1000
0
25
50
75
100
125
150

Figure 2

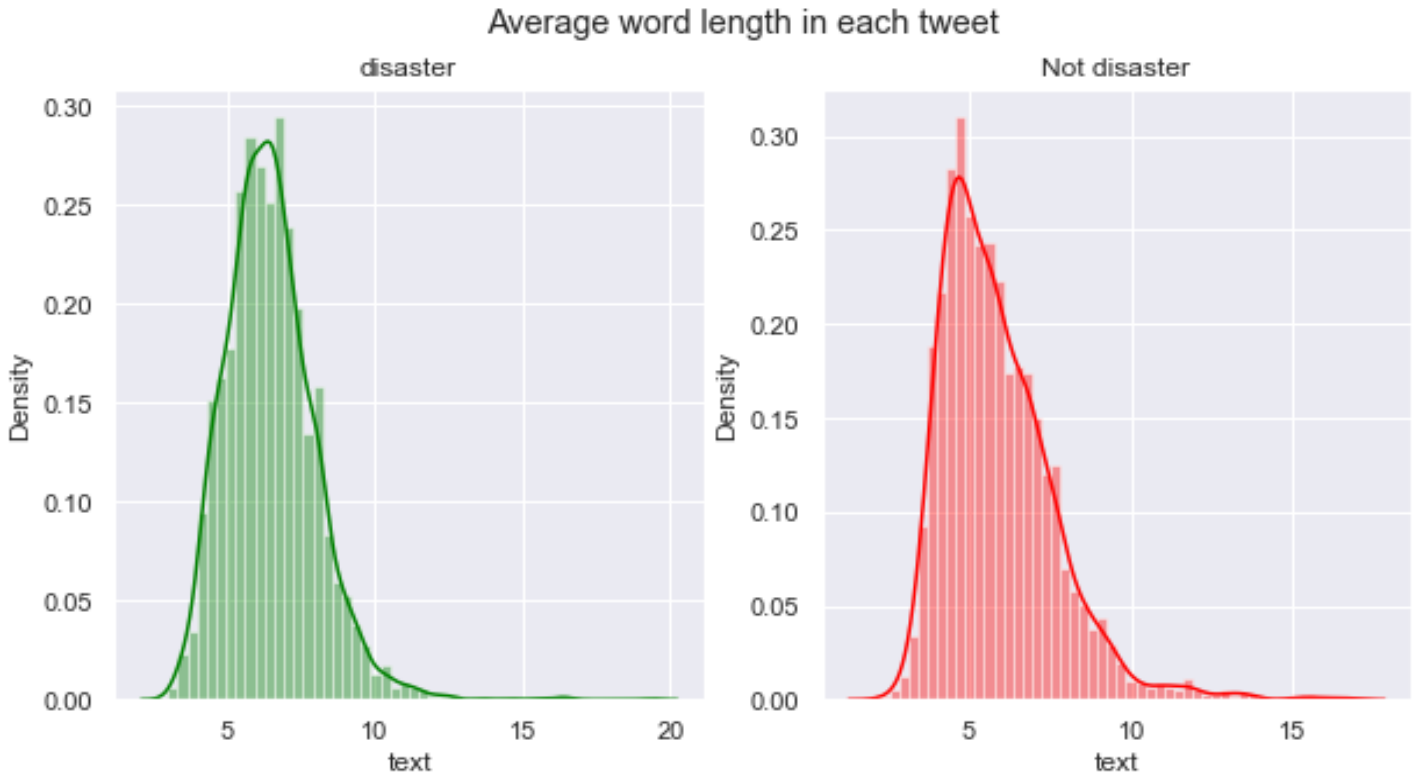


Figure 3

The distributions of both character and word lengths in fake and real disaster tweets were fairly even, so there were no significant outliers or other problems that needed to be addressed aside from cleaning the text data as discussed above.

Next, the keyword column was explored, to check for the most common disaster keywords present in tweets. Some of the data was missing in this column, but the top 15 keywords present and their frequencies in the dataset are shown below.

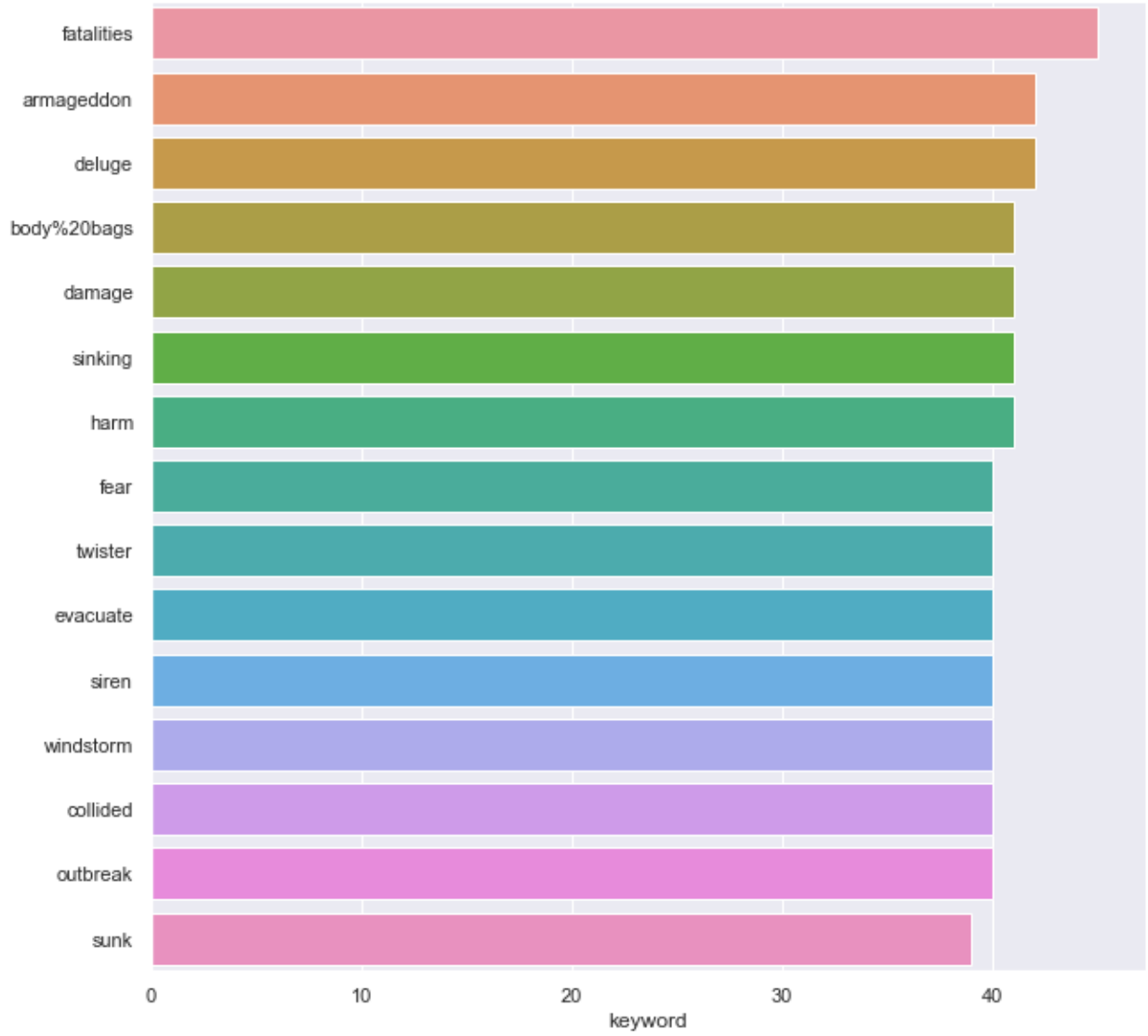


Figure 4

Next, we explored the correlations between targets 0 and 1 to ensure that there was no correlation and therefore no confounding variable that could hinder our model's accuracy. The targets had a correlation of 0.061 with one another, almost none. Seaborn's .heatmap() function was used to check for this, and the results can be seen in Figure 5 below.

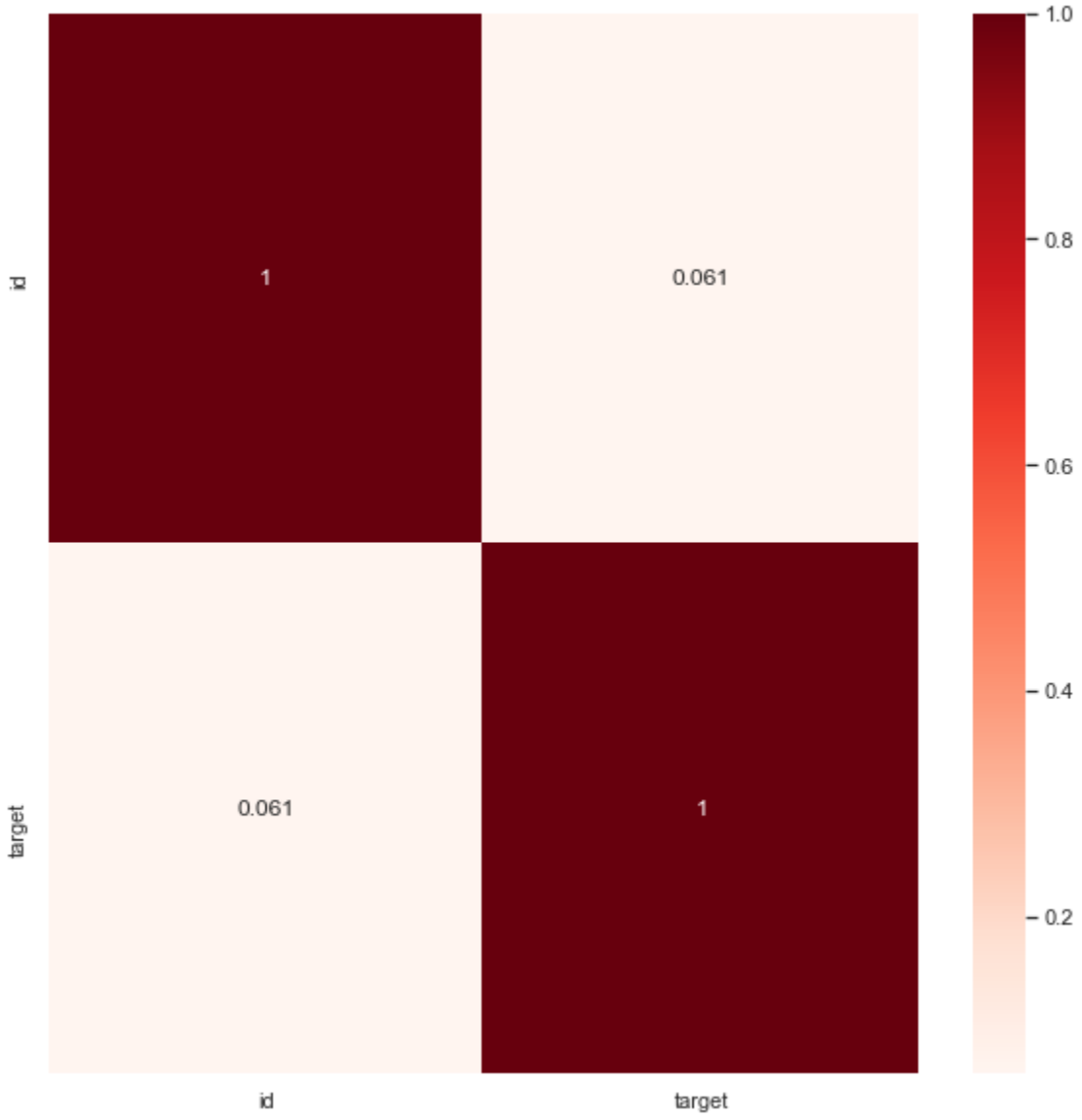


Figure 5

Model Selection

Three models were built to attempt to classify disaster tweets as real or fake. This is a classification problem involving text data, so the models were chosen accordingly. The predictor variable (X) in this case is the text in the tweets, which were vectorized using scikit-learn's CountVectorizer() function. The target variable (y) is the target itself, which is a binary set of numbers classifying a tweet as either a real disaster (1) or a fake one (0).

After defining X and y, we then split the data into training and testing data using the train_test_split module from scikit-learn. The data was split into 30% testing data and 70% training data, leaving us with 2,266 rows of testing data and 5,285 rows of training data. The text data in both training and testing sets was transformed using scikit-learn's TfidfTransformer, which normalizes the matrix and gives weights to the features. Now, we can proceed with actually building the classification models.

The first model is the Naive Bayes model; naive Bayes classifiers are probabilistic classifiers that assume all features and rows are independent of each other. It is a supervised classification technique, and very suitable for the binary classification problem we have. The model was imported from scikit-learn and fit to the training data. We then used the fitted model to predict X_test, which is the text used to test the model. The result is a series of binary predictions on whether that text is referring to a real or fake disaster. Next, we tuned the model's hyperparameters using scikit-learn's GridSearchCV module. The best parameters for this model was an alpha equal to 0.5. The model was fitted again using this alpha, to the transformed training data. It was then, once again, used to classify the testing text. The model performance was assessed in terms of cross validation scores, a confusion matrix, a classification report, and a balanced accuracy score was calculated. After hyperparameter tuning and transforming our data, the model had a training accuracy of 91%, and a testing accuracy of 80.22%, which means it was able to correctly classify the tweets around 80% of the time.

The second model is the support-vector machines (SVM) model. This is another supervised machine learning model that is used for two-group classification problems. It is great for categorizing text, which is the exact problem we have. The model's initial performance, after being fitted to the transformed training data and used to predict the testing data, was not great; about 78.5% accuracy. However, the same process was followed as the Naive Bayes model; the hyperparameters were tuned using GridSearchCV. The best alpha for this model was equal to 0.0001. The model was refitted using this alpha, and the performance was assessed again. We use the same method of cross validation scores, confusion matrix, classification report, and a balanced accuracy score to get the full idea of how the model is performing. The training accuracy was 95.2%. The model's testing accuracy improved to 78.9%; but still lower performing than Naive Bayes.

Lastly, we created a Logistic Regression model, imported from scikit-learn. This is our final supervised learning model, and very commonly used for binary classification problems. After hyperparameter tuning using GridSearchCV (the best C was equal to 0.5), the model was assessed, again in terms of cross validation scores, a confusion matrix, classification report, and a balanced accuracy score. The Logistic regression model's training accuracy was 87.9%, and the testing accuracy was 79.8%.

The testing accuracies between the three models were fairly close; Naive Bayes had the best balanced accuracy score, followed by Logistic Regression, and SVM came last. Looking at precision, recall, and f1 scores shown in the classification reports of each model, Naive Bayes was consistently the best performing in all of them out of all three models. We can then confidently conclude that the best model for this particular problem is the Naive Bayes model (although the Logistic Regression model is a close second).

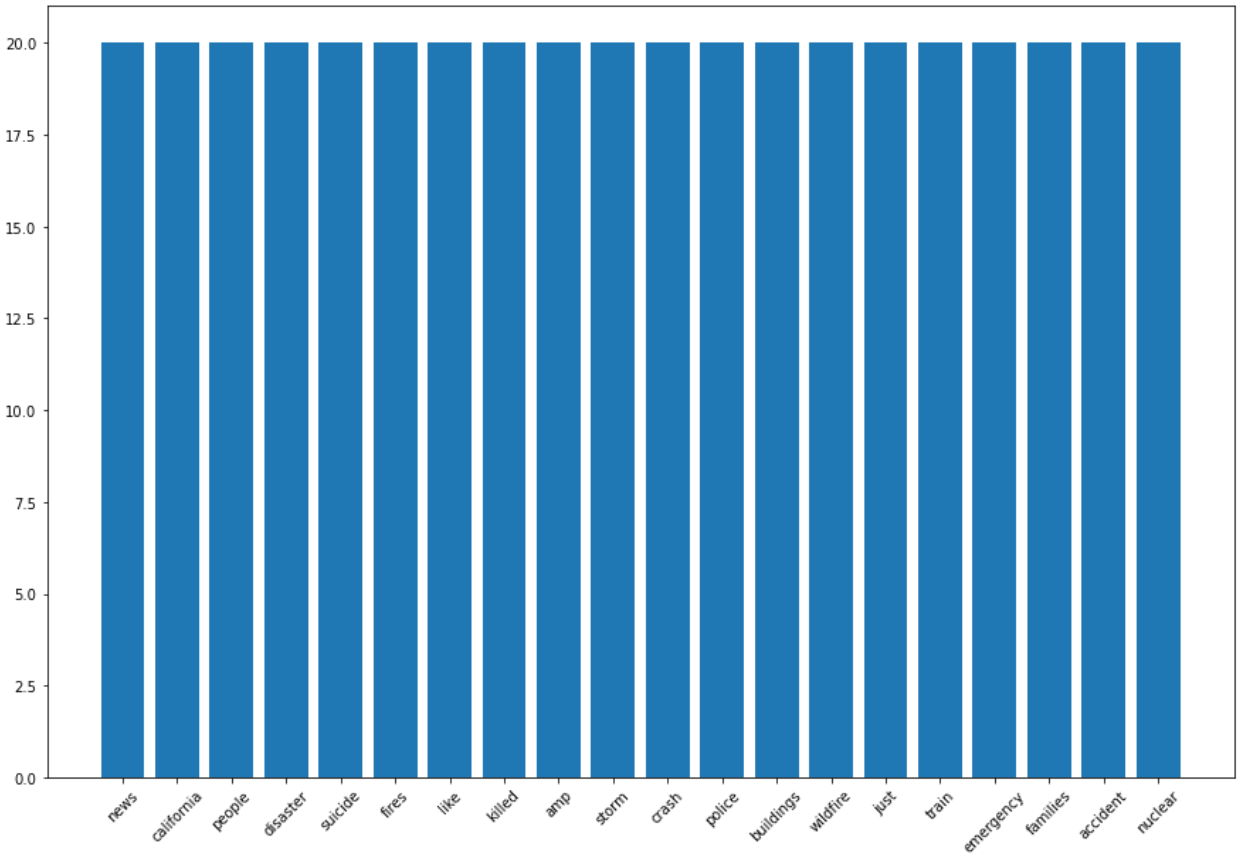
Below, there is a table summarizing each model's performance metrics.

Model Name	Testing Accuracy (%)
Naive Bayes Model	80.22
Logistic Regression Model	79.78
SVM Model	78.95

Figure 6

A Note on Feature Importance

We wanted to get an idea of which words are most important for the Naive Bayes model to make its classifications. We used the .feature_log_prob_ method of the Naive Bayes model, and sorted the feature names. Below is a chart of the words most likely to cause the model to predict a real disaster tweet.



This is another good indication that the Naive Bayes Model is, in fact, working as intended.

Further Research and Recommendations

This model can be used by a variety of clients, including:

1. The model can be used by news agencies, especially those that are active on Twitter, such as CBC, CBS, BBC, NBC, ABC, etc.
2. The model can be used by disaster relief agencies and other grassroots networks with the goal of providing disaster relief, that are active on Twitter.
3. Political agencies and politicians that are active on Twitter that may want to make a quick response to nearby disasters.
4. Law enforcement agencies that monitor Twitter, such as local police, that want to make quick decisions and take swift actions to time-sensitive disasters.
5. State-run organizations such as the Fire Department, Hospitals, etc. that need to respond quickly to disasters nearby.

Further research suggestions:

1. This project can be expanded to classify text in real-time on any type of social media, such as Facebook, Reddit, Instagram, etc.
2. The model can be expanded to also take note of the time and location of each tweet, in order to be more helpful in terms of identifying best responders in real-time.
3. The model can be expanded beyond disaster-related text and be used for other types of natural language processing problems, such as identifying good and bad restaurants, hotels, and so on, in your area (sentiment analysis).