Capstone III Final Report

Introduction

Context:

With the rise of social media comes an abundance of data on how people interact with one another through use of the internet. While it is easy for a human to understand the intent of a post on social media, it is much harder for a machine. This is mainly due to use of sarcasm and buzzwords being used under many different contexts. It is for these reasons that I am interested in looking into tweets containing disaster keywords and building an NLP model that can effectively classify untrained tweets as either about a real disaster or not. This is extremely valuable to organizations that provide disaster relief as they can more efficiently identify real disasters from fake ones. This same project structure could be used with other data to detect fraud, or perform sentiment analysis, and has many more applications.

Problem Statement:

How can we build an NLP model that can classify any given tweet as about a disaster or not while maintaining 90% accuracy or better?

Success Criteria:

Correctly classifying tweets as about a disaster or not. Choosing the best model based on the highest recall while also considering runtime. The aim at the start of the project was to achieve 90% accuracy or better with our best model.

Stakeholders:

The primary stakeholders are disaster relief organizations.

Data Sources:

Data source link: https://www.kaggle.com/datasets/vstepanenko/disaster-tweets

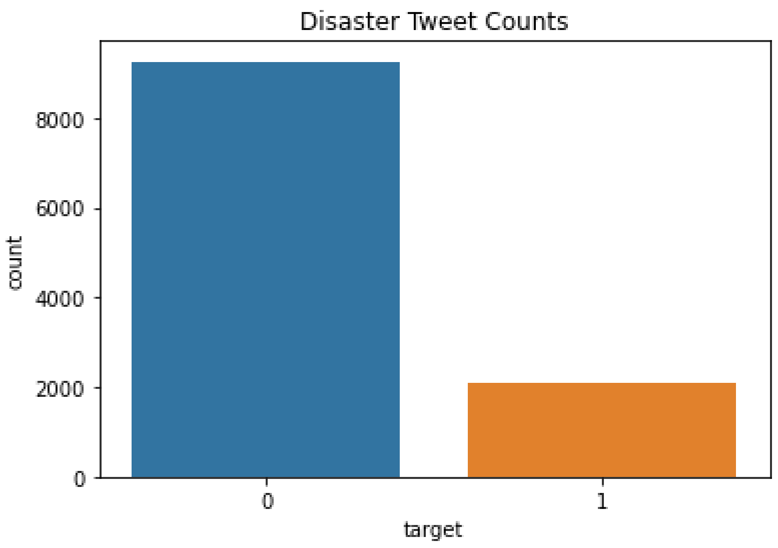
All of the data was scraped from Twitter. The tweets were collected on January 14th, 2020. Topics people were tweeting about include the eruption of Taal Volcano in Batangas, Philippines, COVID-19, bushfires in Australia, and teh Iran downing of airplane flight PS752. The data consists of 1 table, *tweets.csv.*

Data Wrangling

Data Importing and Setup:

The initial data on Kaggle was made up of 1 table: *tweets.csv*. The table contained the following columns: *id, keyword, location, text, and target*. The only columns that we needed for our project were *text* and *target* which hold the tweets and indicator variable values (1 for disaster, 0 for not disaster), respectively.

The first thing that we wanted to do was check the value counts for our target variable to see if it was balanced or not. This is important because dealing with imbalanced data can affect how some models perform. Our data was quite imbalanced, with 9,256 non-disaster tweets and merely 2,114 disaster tweets. This is something that we will monitor throughout the project as we proceed.



Data Cleaning:

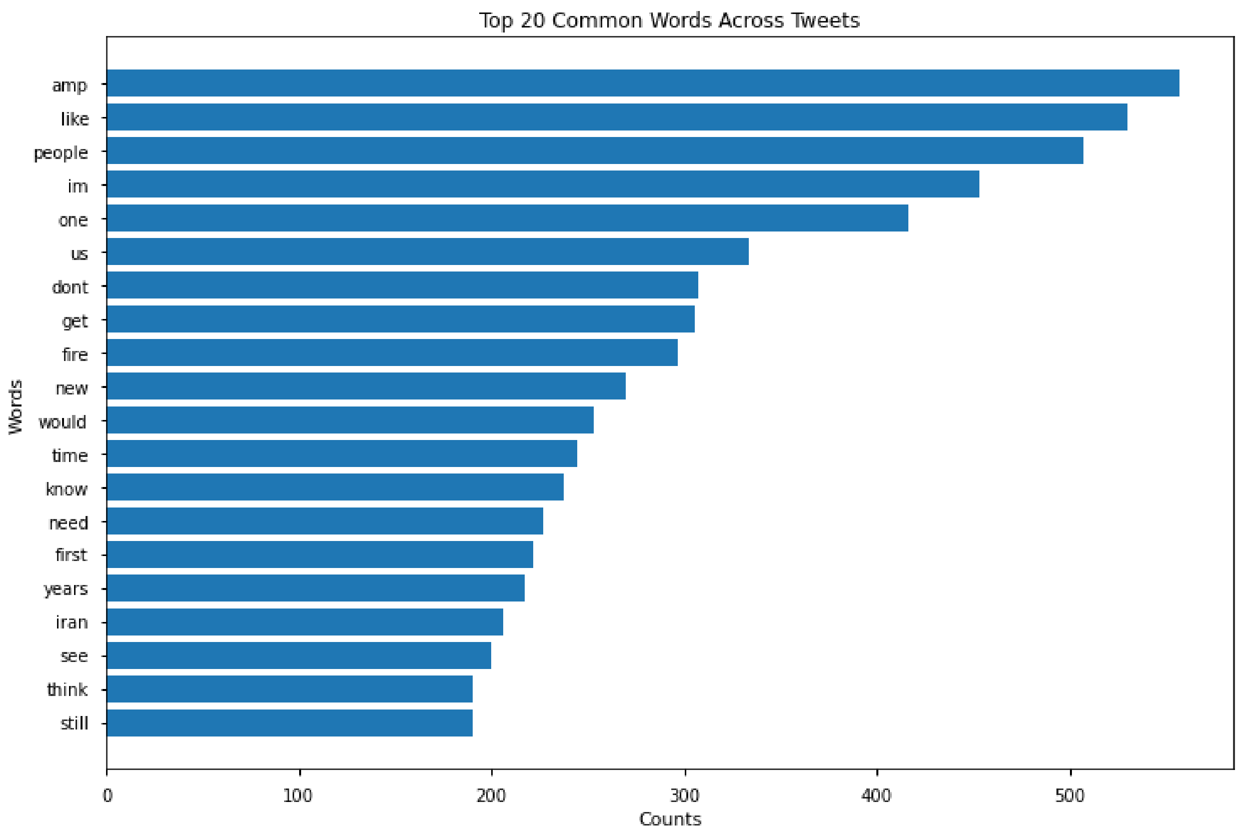
The majority of data cleaning needed to be done had to do with removing unwanted characters from our tweets, removing stop words, and tokenizing. To deal with unwanted characters, we defined our own text cleaning function to convert words to lower case and used regular expressions to eliminate numbers, punctuation marks, double spaces, links, emojis, and other unique strings requiring removal. We then used this function in a for loop to iterate through all the tweets in our table. Thankfully, foreign words were not of too much concern as they were not very common throughout the data, but it is important to note that we kept them in. Once our text was in the form we liked, the next step was to tokenize the text, which means breaking down the tweets by word. This gave us a list of lists, with each inner list containing the tokenized words from each tweet.

Once we had our list of lists we then used for loops to remove stop words and append the remaining words to new lists, and then append these lists to a new list of lists. The result is the same list of lists as previously mentioned only with stop words removed this time.

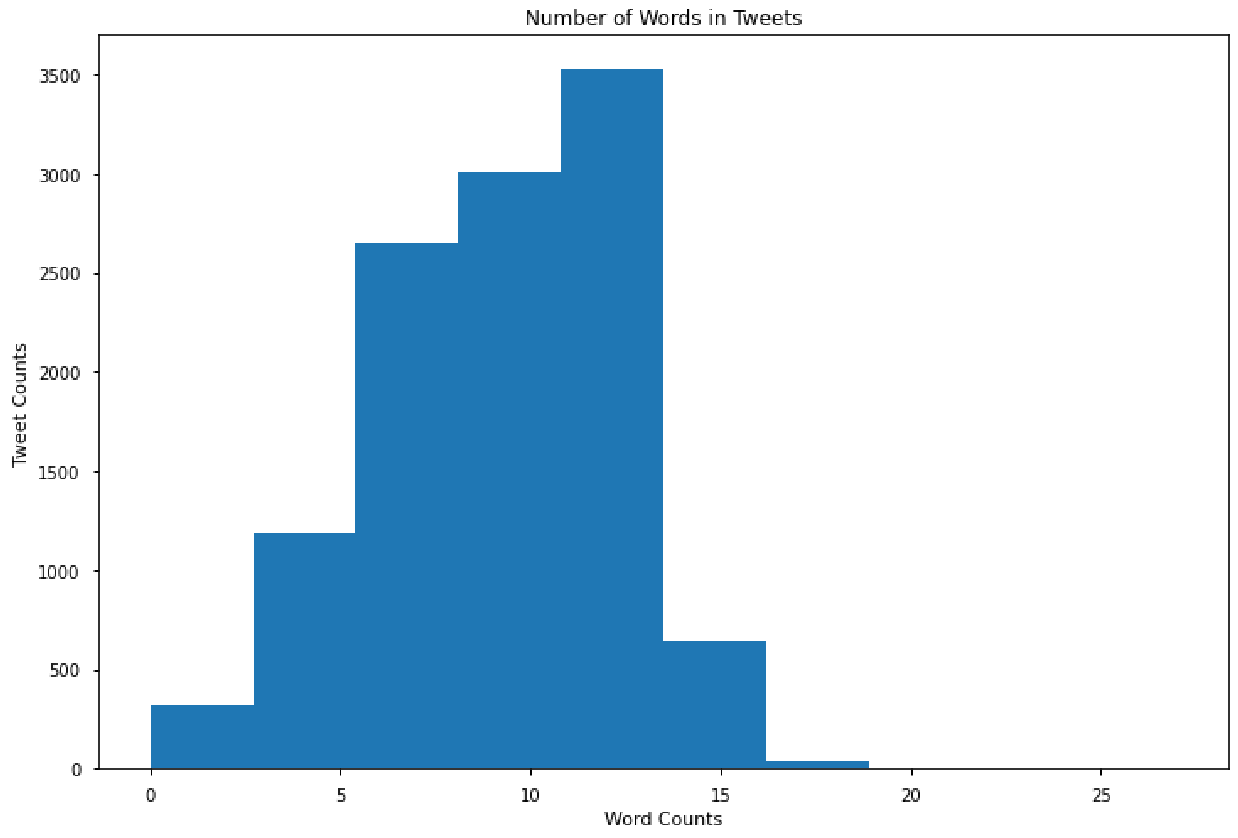
Exploratory Data Analysis

Exploring Our Data:

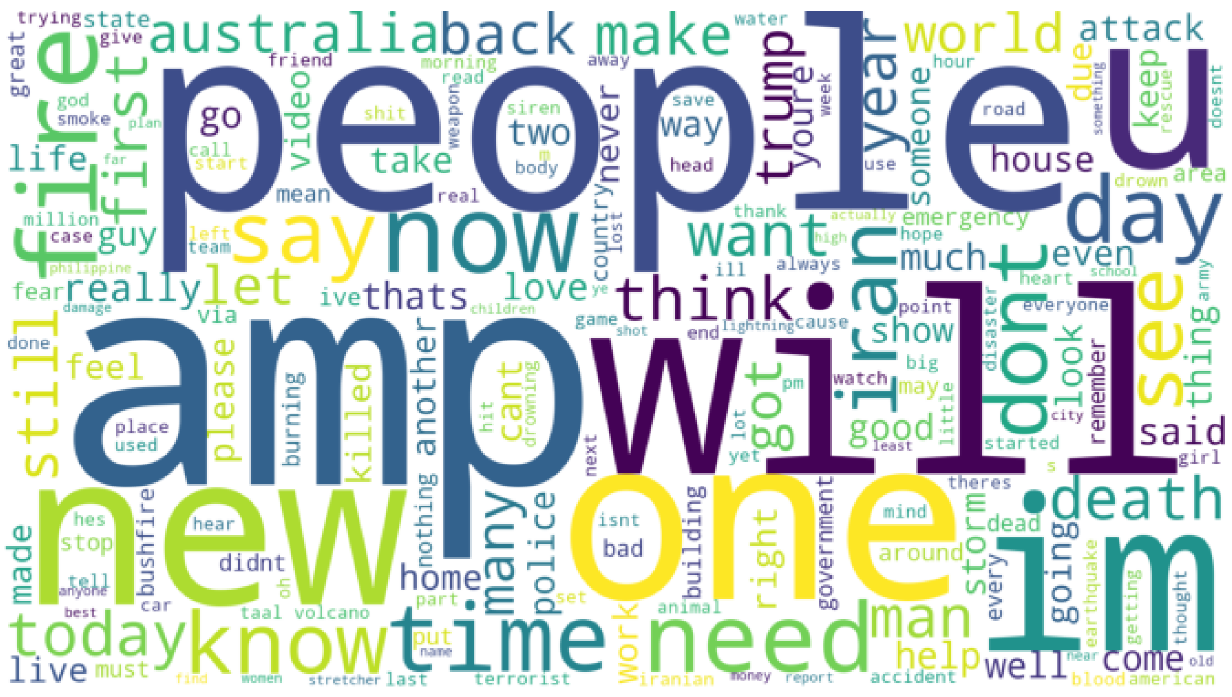
There aren't too many useful visualizations that can be created when dealing with NLP data, but what we did was took a look at the most common words among all of our tweets. The resulting plot is shown below:



Some of the words in the plot seem more general and some more related to disasters, intuitively. We could consider removing some more of these common words to make our model even stronger down the line. In addition to looking at word counts we also plotted a histogram of the amount of words per tweet:



Lastly for our EDA, we plotted a word plot to view the most common words in our tweets in a different way then earlier:



Before moving on we converted our tokens back into sentences by using a for loop and appending spaces between words.

Preprocessing

Final Cleaning Steps:

The first thing that needed to be done in this stage was removing the unwanted columns that have thus far been untouched. After this was done it left us with a table of just the cleaned tweets and target indicator. We noticed that there was one tweet with a null value for the cleaned tweet (most likely due to an error in the cleaning stage earlier) and just dropped this one row. We then were ready to split our data into train and test sets using a 80/20 split. Our last step for preprocessing was to create count and tfidf vectorizers and fit both separately with our training data (features, not target). We then transformed using both vectorizers on our training and test features. This left us in a solid spot to begin modeling using our two different vectorized features.

Modeling

Important Information:

It's worth noting that we did not perform cross validation when training our models and selected our hyperparameters without performing a gridsearch. This was mainly for computational time purposes but taking these additional steps would likely help models improve. Since we aren't performing cross validation we only looked at the scores on the test set to see how our models adjusted to new data. Since we care more about labeling disaster tweets correctly (we want our model to identify disaster tweets correctly more than having high precision within the positive class) we used recall for the positive (disaster) class as our evaluation metric.

Imbalanced Data:

The models we tried:

- *Logistic Regression*:
  - Chosen Hyperparemeters:
    - *penalty*: l2
    - *max_iter*: 500
    - *C*: 0.1
    - *solver*: lbfgs
  - Count Vectorizer
    - Score on Test Set (Disaster Class):
      - *Recall*: 0.54
  - TFIDF Vectorizer
    - Score on Test Set (Disaster Class):
      - *Recall*: 0.36

- *Random Forest*:

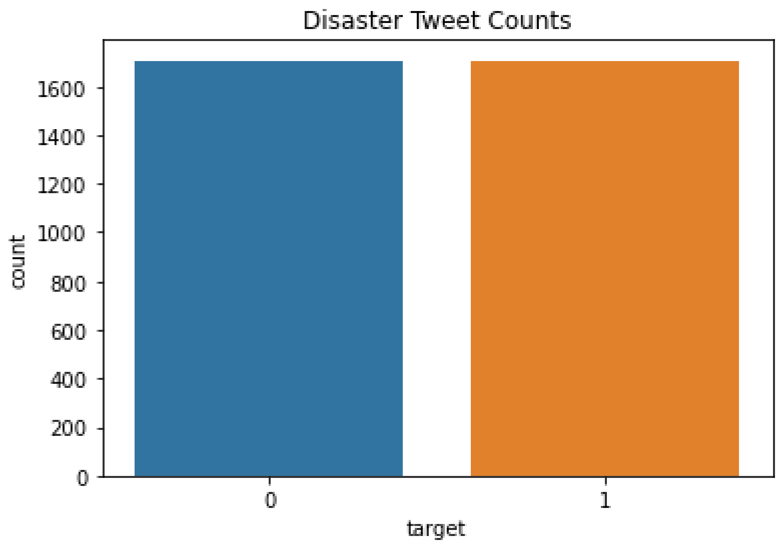
    - Chosen Hyperparemeters:
        - *n_estimators*: 500
        - *n_jobs*: -1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.49
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.49

- *KNN*:
    - Chosen Hyperparemeters:
        - *n_neighbors*: 5
        - *n_jobs*: -1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.20
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.18

- *SVM*:
    - Chosen Hyperparemeters:
        - *C*: 1
        - *gamma*: 1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.19
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.44

- *Gradient Boosting*:
    - Chosen Hyperparemeters:
        - *n_estimators*: 100
        - *max_depth*: 10
        - *max_features*: auto
        - *learning_rate*: 0.1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.36
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.36

While our best model is the logistic regression one we trained and tested first, a recall of 0.54 is not very strong and is barely better than guessing randomly. To try to improve these scores, we decided to balance our data by undersampling the non-disaster (0) class for both our count and tfidf vectorized data. This resulted in the following countplot (same for both vectorizers):



It’s worth noting that while the amount of tweets in each class are identical for both vectorizers, the random undersampler picked different tweets for each of the two vectorizers. We fit all the same models with our balanced data.

Balanced Data:

The models we tried:

- *Logistic Regression*:
    - Chosen Hyperparemeters:
        - *penalty*: l2
        - *max_iter*: 500
        - *C*: 0.1
        - *solver*: lbfgs
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.75
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.77

- *Random Forest*:
    - Chosen Hyperparemeters:
        - *n_estimators*: 500
        - *n_jobs*: -1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.78
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.75

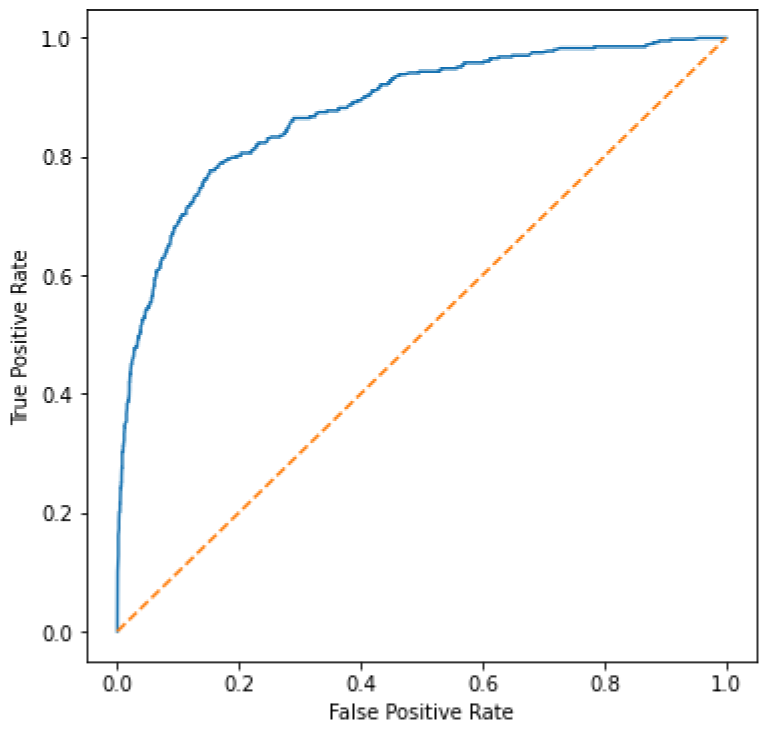
- *KNN*:
    - Chosen Hyperparemeters:
        - *n_neighbors*: 5
        - *n_jobs*: -1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.26
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.20

- *SVM*:
    - Chosen Hyperparemeters:
        - *C*: 1
        - *gamma*: 1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.29
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.76

- *Gradient Boosting*:
    - Chosen Hyperparemeters:
        - *n_estimators*: 100
        - *max_depth*: 10
        - *max_features*: auto
        - *learning_rate*: 0.1
    - Count Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.72
    - TFIDF Vectorizer
        - Score on Test Set (Disaster Class):
            - *Recall*: 0.70

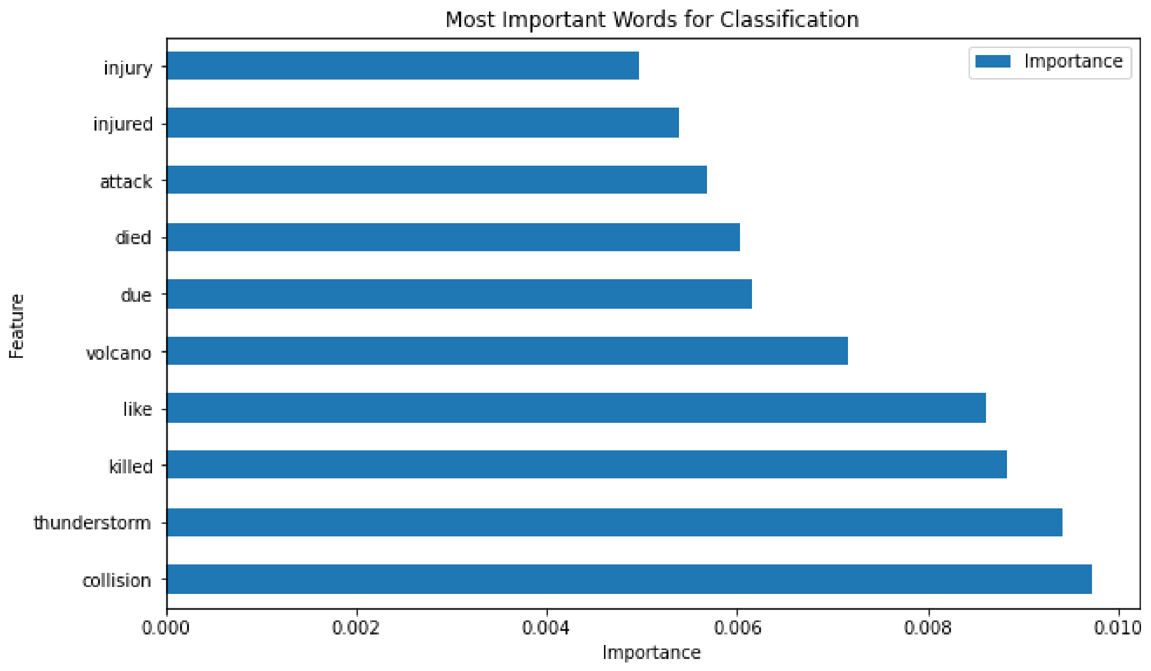
Final Results:

All of our models using the balanced data outperformed those using the imbalanced data. We did lose a lot of data through the undersampling, but these scores are undoubtedly much better. Our best model is the random forest using our count vectorized data with a recall of 0.78. The ROC-curve for our best model can be seen below:



The ROC AUC score was 0.880. This is a pretty good score and tells us that our model definitely has some predictive power.

Before finishing up we plotted the most important words in helping classify disaster tweets correctly by extracting the feature importances from our best model:



We can now see that some of the most helpful words were collision, thunderstorm, and killed. Words like these being high in importance is unsurprising as we could see these words relating to disastrous events. While we can’t be sure which words contribute to predicting which class correctly, we can assume that at the very least these three top words likely contribute to predicting the positive class of disaster tweets.

Future Scope

- While the model is not perfect, it can be used to help better identigy which tweets peratin to real disasters and which do not. This allows disaster relief organization to more efficiently do their jobs.
- There are many models that I did not get the chance to try, and it is possible that there exists a better model that will produce more useful results. To avoid losing a lot of our data when undersampling, we could try to incorporate some deep learning neural networks that may produce better recall scores. Computational time would likely suffer, but we could try using a neural network on both the imbalanced and balanced data to see if results improve at all.
- We could look further into the feature importances by plotting word clouds for tweets, subsetted by TP, TN, FP, and FN. This would allow us to better understand which words contribute to which predictions (right and wrong ones).
- I would love to dive into a similar project with sentiment analysis or fraud detection to see how similar or different the processes are. Much of it would be similar but I would be interested to see what other issues arise and how to fix them.