# INTRO TO MACHINE LEARNING FINAL PROJECT

# DISASTER TWEET DETECTION

MATTHEW BUCHANAN

## ABSTRACT

In recent years Twitter, or X, has become one of the most popular social media websites for people all around the world. Topics range from gossip and petty drama to politics and current affairs. This project is focused on natural and man-made disasters shared on twitter. When an emergency such as an earthquake or plane crash happens the first thing many people do is pull out their phone to document it and share the news with friends and strangers. Law enforcement and rescue agencies are among the many organizations interested in detecting disasters in real time via social media. Such detection would allow authorities to be aware as soon as an event occurs and stay updated throughout its duration.[1] Seven machine learning models are used to analyze twitter data with the goal of predicting when a real disaster is occurring.

## PROBLEM DEFINITION AND GOALS

The goal of this project is to use machine learning to detect whether a given twitter post is about a real disaster. The implications of successful sentiment analysis like this are wide ranging. From detecting disasters and identifying disinformation campaigns to filtering automated "bot" posts.

The project idea and data come from the machine learning and data science website Kaggle.com.[2] It is part of their getting started collection of challenges and is still open and ongoing.

The data came from a company called "Data for Everyone" whose website link no longer works. It is a collection of 10,000 tweets that have been hand labeled as to whether they are about a real disaster or not. Only 7,615 of the items have labels available for use in training machine learning models. The rest are held back by Kaggle.com for evaluating user submitted results.

The data is formatted as follows:

- id: a unique identifier for each tweet.
- text: the text of the tweet itself.
- location: an optional user provided location the tweet was sent.
- keyword: keywords selected from the text by the labelers which may also be blank.
- target: the label which states whether the tweet is about a real disaster or not.

The data set is rather small to work with at 7,613 rows available for testing and training. The data is also unfiltered, and several rows have missing values. Before we can train any models, we will have to clean and impute it.

We will train 7 different models on the data:

- A knn classifier
- A naïve bayes classifier
- An elastic net regression classifier
- A random forest classifier
- A gradient boosted machine classifier
- A svm classifier
- A feed forward neural network classifier

Finally, we will take a majority vote of all 7 models to see how they work together as an ensemble.

## RELATED WORK

Natural language processing is one of the most popular fields in artificial intelligence and machine learning at the moment. Recent advancements in techniques such as attention and transformers have enabled amazing products like ChatGPT from OpenAI and Gemini from Google.

This project is based on an open beginner's competition on Kaggle and many people have submitted results. Many of the techniques used are advanced natural language processing techniques such as long short-term memory and pre-trained models. Many utilized powerful cloud processing like Google's collab to for training. The competition is graded by comparing uploaded predictions from a separate test set with hidden labels. The metric is F1 score. Looking at the leaderboard reveals several people have managed an F1 score of 1.

An example of the many articles that address this Kaggle competition is "Emergency Tweet Classification and Model Deployment" by Isaac Misri. [3]

## DATA EXPLORATIOON AND PREPROCESSING

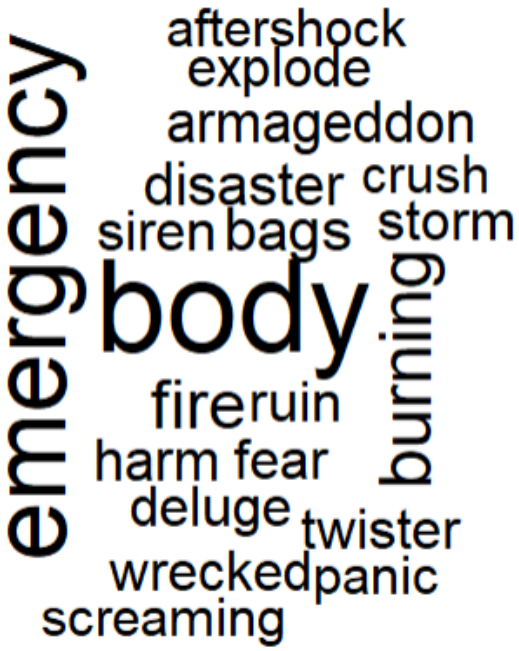
The data set from Kaggle contains 7,613 rows with just 5 variables:

**id** – a numeric value that assigns a unique identifier to each tweet. This variable has no predictive value and so was dropped from the data.

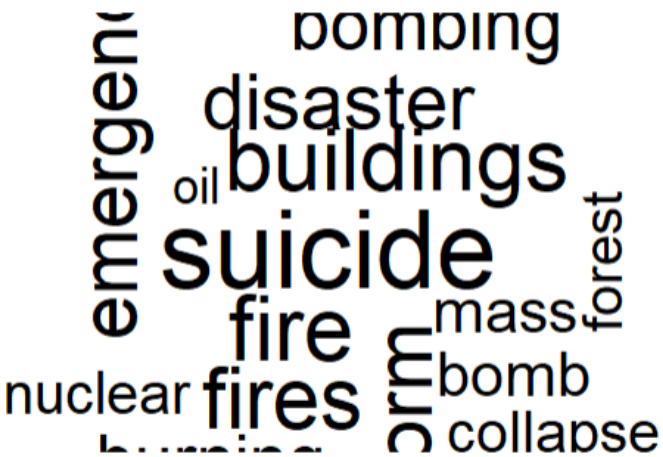
**keyword** – a character string containing keywords derived from the tweet by the company that labeled the data. No information is available on the methodology of assigning these keywords. The keywords are all lowercase. There are 222 unique keywords in the data.

Some rows had two words concatenated together, such as “airplane%20crash”, with the characters “%20” as separator. Since the meaning of “airplane crash” isn’t the same as just “airplane” or “crash”, the values “%20” were replaced by a space to preserve the original meaning for training.

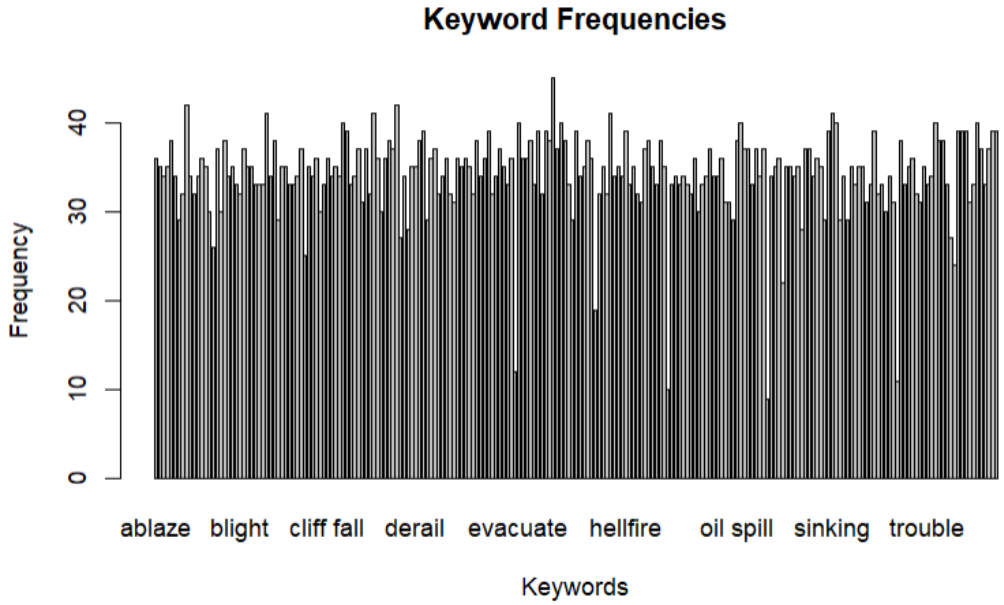
Word clouds were generated for the two different values of the target variable. The word could for false disaster tweets:



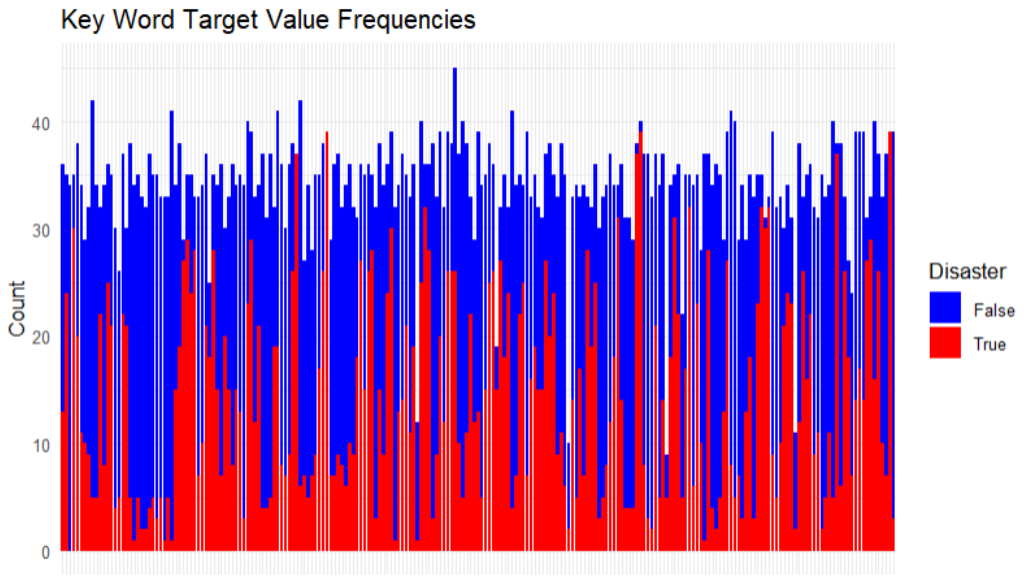
And here is the word cloud for real disaster tweets:



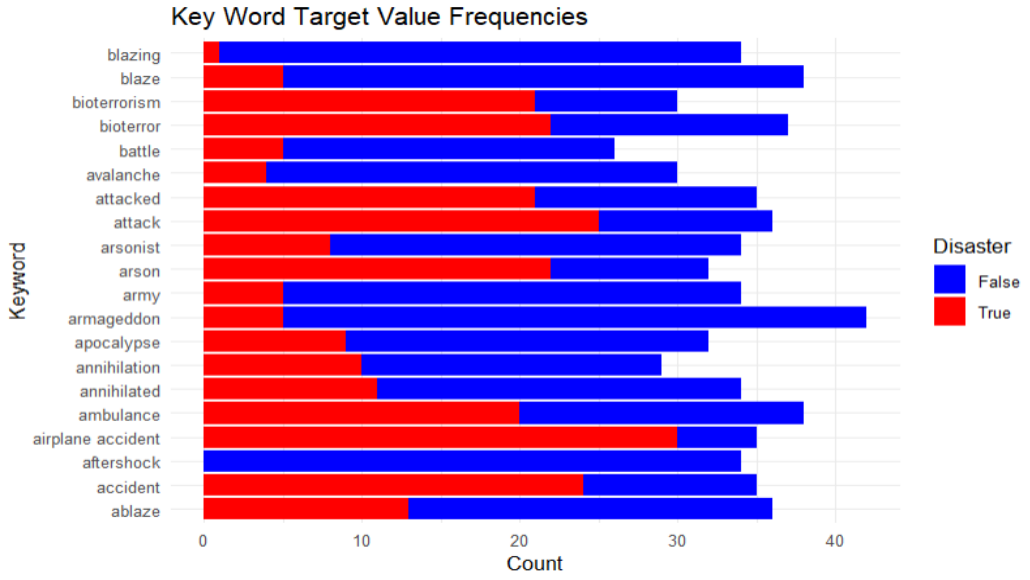

Words like disaster and emergency appear in both groups, but the two clouds are quite different. Next the frequencies of each keyword were graphed. While there are too many to list individually, It is worth noting how uniform the frequency is.



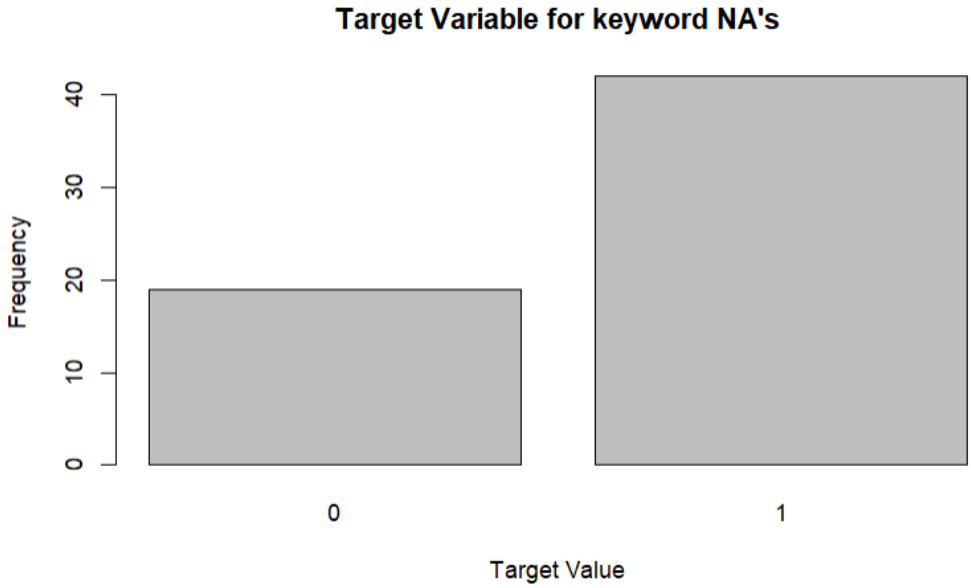
The distribution of key words between the false and real disaster groups was plotted next. Most words mostly appear in one group or the other, with a fairly even distribution between real and false disasters across all the keyword values.




To better show this, the first twenty keywords and their distribution for the target variable are shown below.

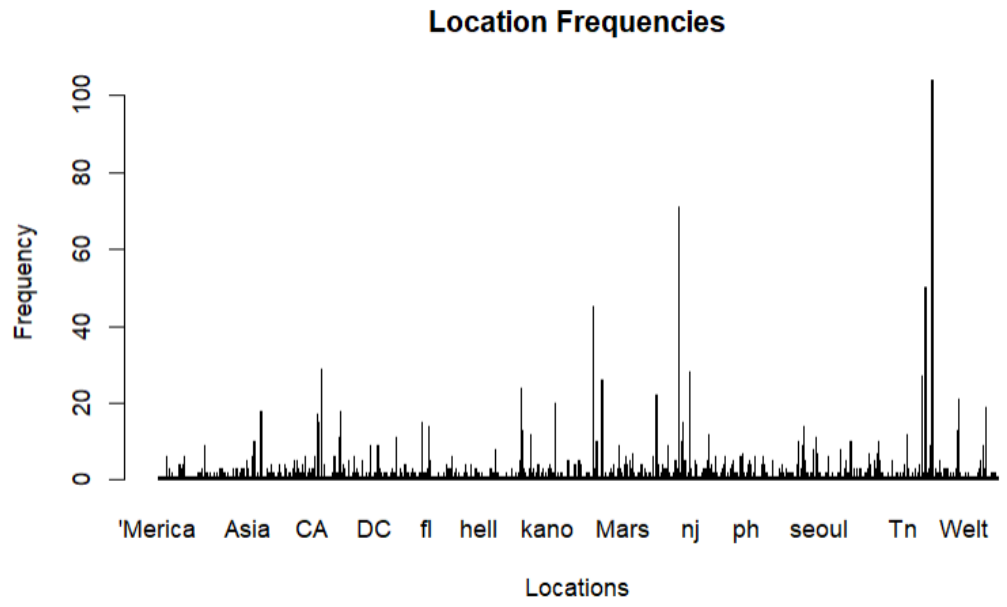


The final step was checking the distribution for missing values of keyword across the two target variable groups. As shown below, the distribution of keyword NA's is very imbalanced with most 43 NA's in the real disaster group and 20 in the false disaster group.

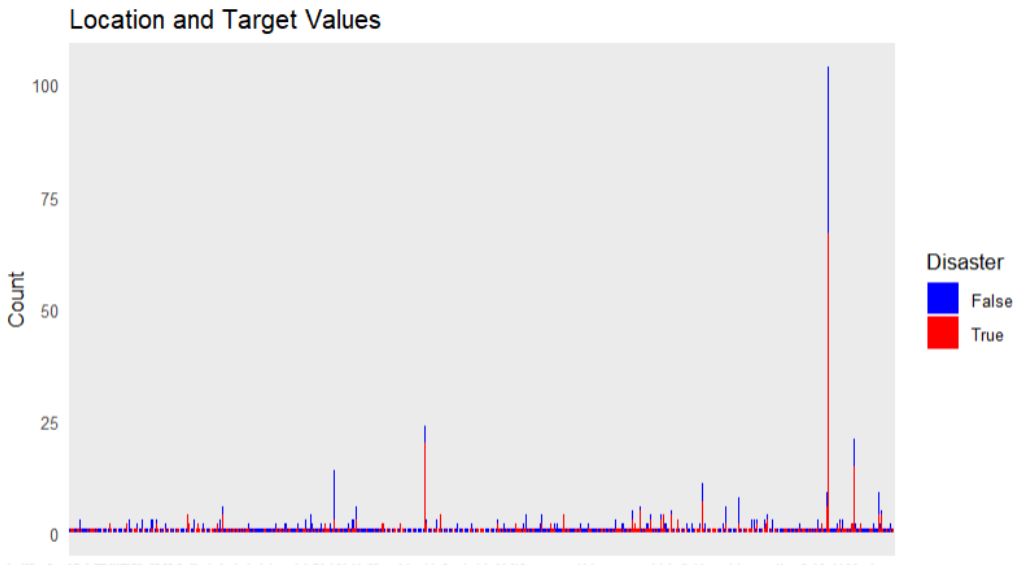



**location** – a character string containing the location the tweet was posted from. This item is optionally entered by the user at the time they create the tweet. Not only are many values missing but many of the values that are present don't represent real locations, for example "reading a book". Some are simply gibberish. This variable is challenging to work with because of its sparsity and number of missing values as well as the fact that users can enter whatever they want into the field. However, with such limited data available for training we can't afford to discard it. The locations are uncleaned and there are 3,342 unique values.

A plot of the frequency of locations is shown below. With so many unique values the variable is very sparse. Most values appear very few times or even once, while some appear over a hundred times.

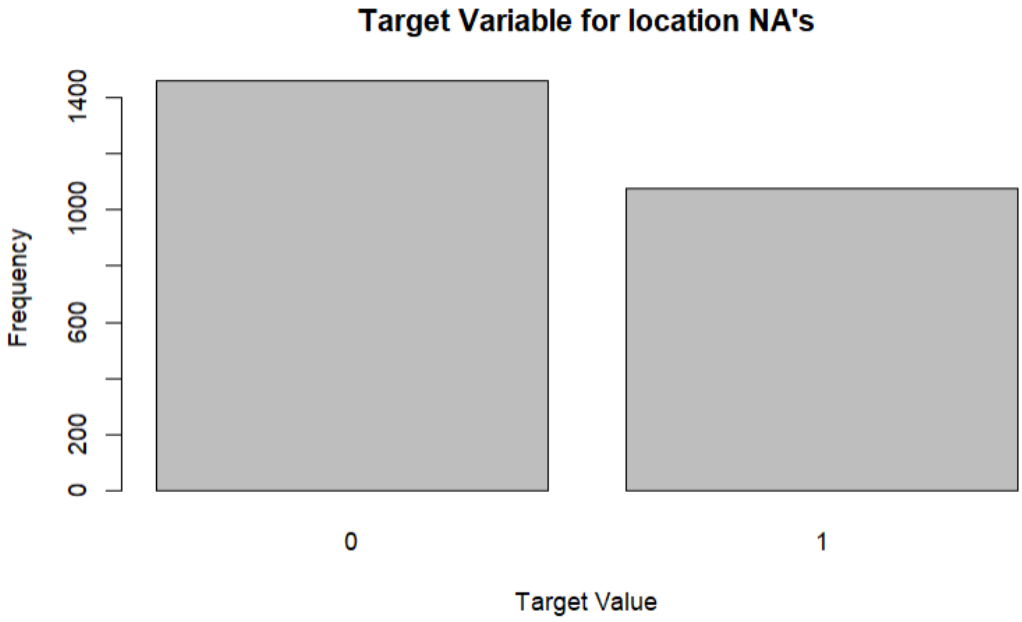



Similarly, the plot of the location variable's distribution between the target variable's values is also very sparse.




Below is a plot of the first 20 location's distribution between the values of the target variable. It shows that some locations are gibberish, and many appear only once.

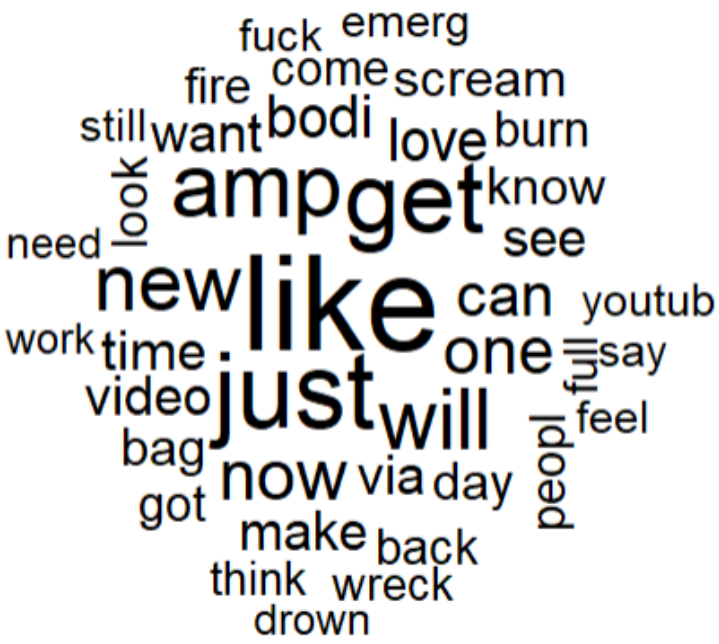

Finally, the distribution of NA's for location between the target variable's values is shown below. The distribution is fairly balanced this time.



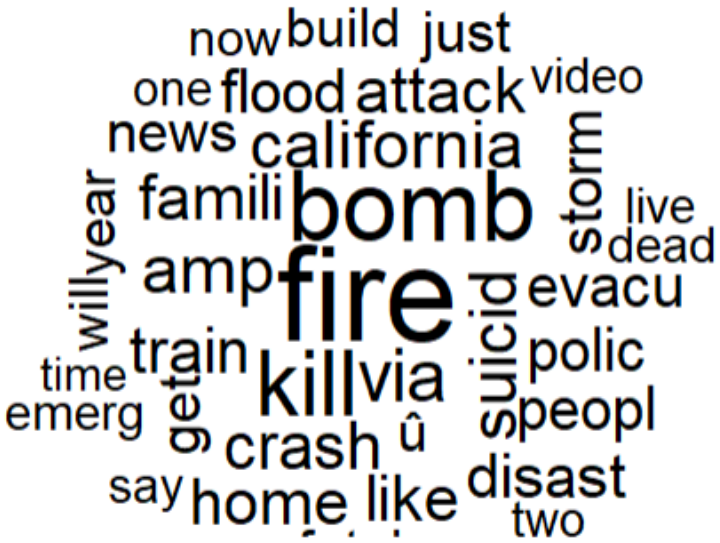
**text** – text is a character string. It is the body of a tweet. There are no missing values for this variable and none of the values are empty strings. This variable needed heavy processing before it could be used. The average number of words per tweet after processing for false disaster texts is 8.5, while the average from real disaster texts is 9.2 .  Processing included:

- Transforming all values to lowercase
- Removing stop words
- Removing http links
- Removing hash tags and @ symbols
- Removing forward and backward slashes
- Removing numbers and punctuation
- Stemming
- Stripping remaining white space

After processing there are 6,792 unique words in the text column. A word cloud for text from false disasters is shown below.

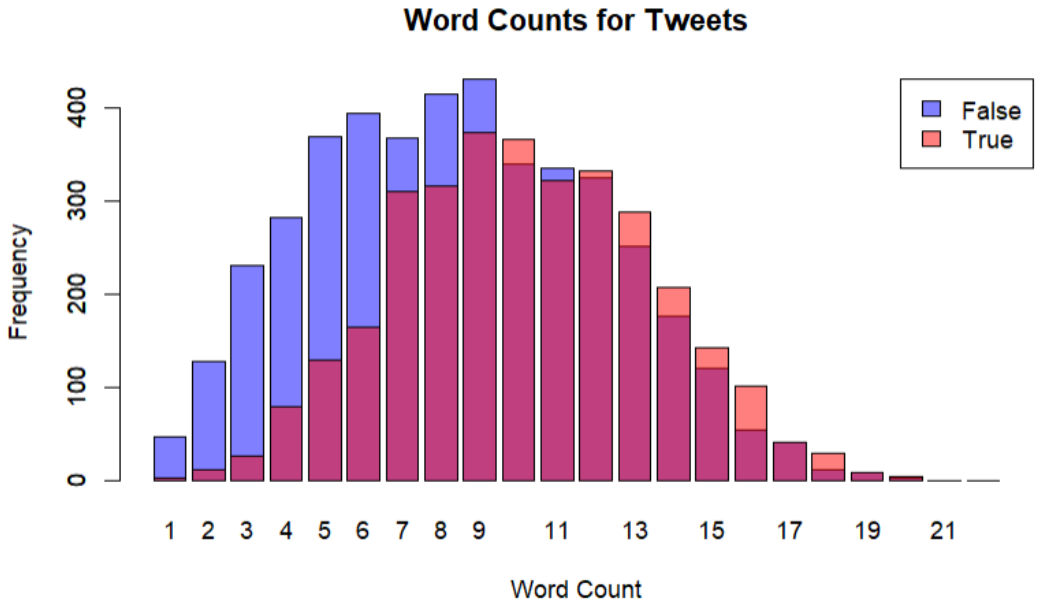


And one for text from real disasters.

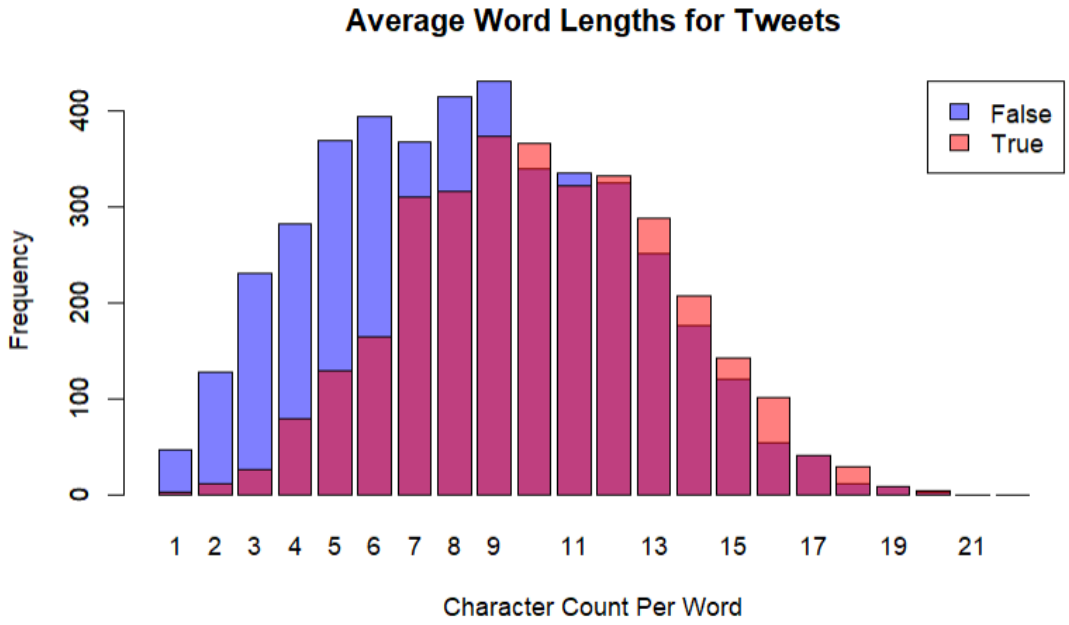



The two word clouds are quite different this time. Hopefully this difference in distribution will help the models differentiate the two groups.

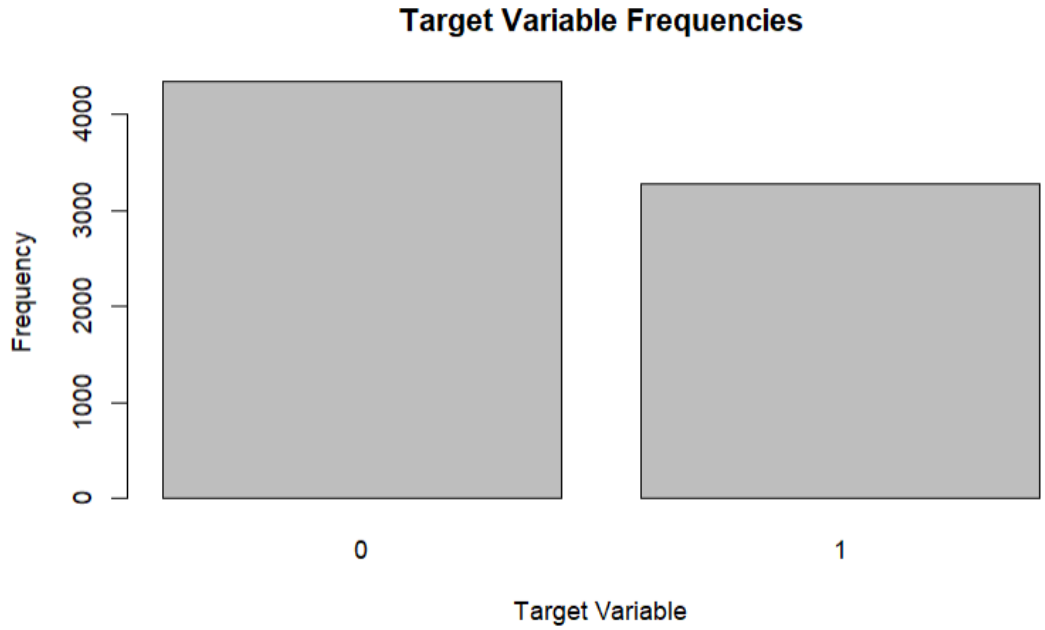
Below the word counts per text for the two values of the target variable is shown.



Finally, the average word length distribution is plotted.

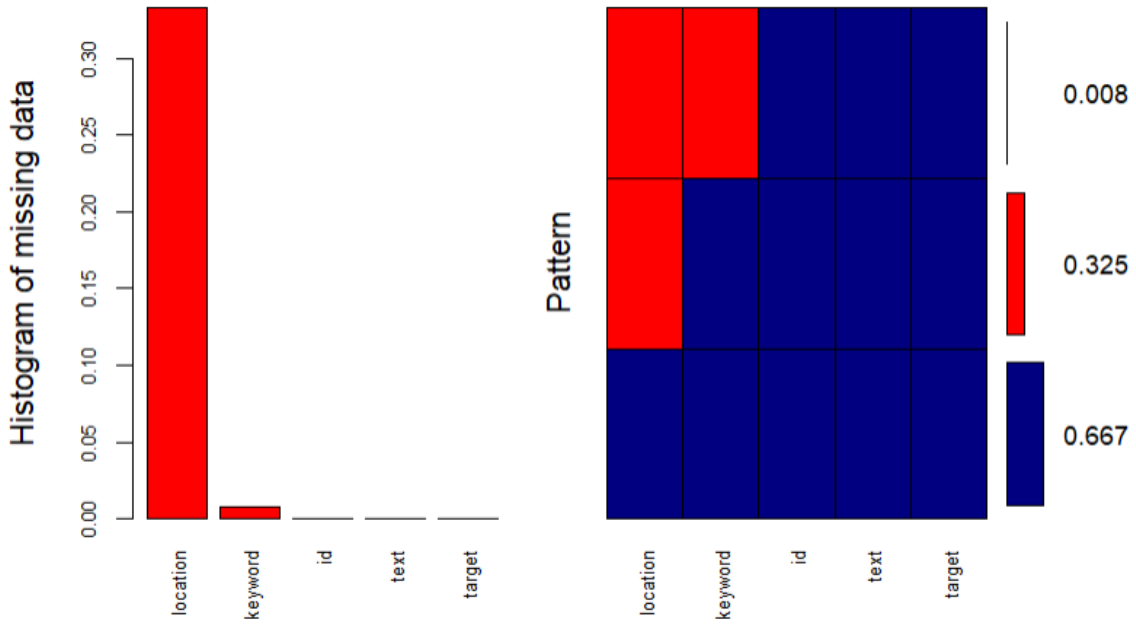



**Target** – The target variable is binary with no missing values. Its distribution is fairly balanced at ~ 57/43 split and requires no re-balancing.



**DATA IMPUTATION**

With so many unique values for both keyword and location variables, determining correlation is very challenging. There is no good way to impute the missing values for text or location. The missingness for both variables was correlated with the target variable, so the absence of value is useful for training. Because there is already so little data available to train the models, no rows were deleted. Instead, the new strings “unknown” for location and “none” for keyword were used for the missing values. A plot of missing values is given below.




**ENCODING**

Both the keyword and location variables were encoded using feature hashing implemented by the “textrecipes” library with the step_dummy() function. 25 terms were used for keyword and 5 terms were used for location.

For the text variable tf-idf encoding was used via library “keras” function layer_text_vectorizer with 500 tokens. Anymore tokens than this resulted in computer hangs in R Studio. The tf-idf columns were then z normalized.

## DATA ANAYSIS AND EXPERIMENTAL RESULTS

The data from Kaggle was split into two sets: a training set with 90% of the data, and a test set with 10%. Again, with so little data to train on I wanted to make the training set as large as possible. After splitting the data seven models were trained:

- K-Nearest Neighbors model using the caret library
- Naïve-Bayes model using the caret library
- Elastic Net regression model using the caret library
- Random Forest model using the caret library
- Gradient Boosted Machine model using the caret library
- Support Vector Machine using the caret library
- A Feed Forward Neural Network model using the keras library

For the first six models 10-fold cross validation was used, and the caret package was allowed to auto tune the hyper parameters. For the neural network model, the training set was split 90/10 into new training and validation sets and the tfruns library with a flags file was used to tune the hyper parameters.

**KNN Model** – Accuracy was used as the metric for auto tuning. K values of 5, 7, and 9 were tried and evaluated using accuracy with a final value of k = 7 chosen. The results from the test set:

```
            predictions
test_labels   0   1
          0 339  39
          1 128 150
[1] "Accuracy: 0.745427"
[1] "Precision: 0.793651"
[1] "Recall: 0.539568"
[1] "F1: 0.642398"
```

Accuracy: 74.5%

AUC: 0.7182

**Naïve Bayes model** – Accuracy was used as the metric for auto tuning. The final hyper parameters chosen were Laplace = 0, adjust = 1, and use kernel = false. The results on the test set:

```
            predictions
test_labels   0   1
          0 334  44
          1 141 137
[1] "Accuracy: 0.717988"
[1] "Precision: 0.756906"
[1] "Recall: 0.492806"
[1] "F1: 0.596950"
```

Accuracy: 71.8%

AUC: 0.6882

**Elastic Net model** – Accuracy was used as the metric for auto tuning. The final hyper parameter values were alpha = 0.1 and lambda = 0.01323339.

```
            predictions
test_labels   0   1
          0 327  51
          1  91 187
[1] "Accuracy: 0.783537"
[1] "Precision: 0.785714"
[1] "Recall: 0.672662"
[1] "F1: 0.724806"
```

Accuracy: 78.4%

AUC: 0.7689

**Random Forest model** – Accuracy was used as the metric for auto tuning. The final hyper parameter value was 32 variables chosen randomly for each split:

```
            predictions
test_labels   0   1
          0 323  55
          1  69 209
[1] "Accuracy: 0.810976"
[1] "Precision: 0.791667"
[1] "Recall: 0.751799"
[1] "F1: 0.771218"
```

Variable importance :

| | Importance <dbl> |
|---|---|
| V15 | 100.00000 |
| V44 | 95.95642 |
| V30 | 91.42512 |
| V129 | 91.28689 |
| V3 | 85.18497 |
| V45 | 84.18832 |
| V50 | 81.88989 |
| V98 | 77.23702 |
| V421 | 76.18168 |
| V242 | 75.20355 |

These variables all correspond to tf-idf tokens from the 'text' column.

Accuracy: 81.1%

AUC: 0.8031

**Gradient Boosted Machine model** – Accuracy was used as the metric for auto tuning. The final hyper parameter value were n.trees = 150, interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.

```
             predictions
test_labels   0   1
          0 349  29
          1 120 158
[1] "Accuracy: 0.772866"
[1] "Precision: 0.844920"
[1] "Recall: 0.568345"
[1] "F1: 0.679570"
```

Accuracy: 77.3%

AUC: 0.7458

**Support Vector Machine model** – Accuracy was used as the metric for auto tuning. The final hyper parameter value was C = 1. A linear kernel was used.

```
             predictions
test_labels   0   1
          0 326  52
          1  88 190
[1] "Accuracy: 0.786585"
[1] "Precision: 0.785124"
[1] "Recall: 0.683453"
[1] "F1: 0.730769"
```

Accuracy: 78.7%

AUC: 0.7729

**Feed Forward Neural Network model** – Validation loss was used as a metric to choose the best model. The Adam optimizer with binary cross entropy loss function was used. The final hyper parameters chosen were:

- First layer: 64 nodes with sigmoid activation
- Second layer: 64 nodes with relu activation
- Third layer: 16 nodes with sigmoid activation
- Learning rate: 0.01
- Batch size: 64
- Dropout: 0.1 for all three layers

```
            predictions
test_labels   0   1
          0 287  91
          1  79 199
[1] "Accuracy: 0.740854"
[1] "Precision: 0.686207"
[1] "Recall: 0.715827"
[1] "F1: 0.700704"
```

Accuracy: 74.1%

AUC: 0.7375

**Majority Vote model** – A simple majority vote model was used at the end, combining results from all 7 models.

```
            predictions
test_labels   0   1
          0 338  40
          1  92 186
[1] "Accuracy: 0.798780"
[1] "Precision: 0.823009"
[1] "Recall: 0.669065"
[1] "F1: 0.738095"
```

Accuracy: 79.88%

AUC: 0.7816

**FINAL RESULTS**

The final results are shown below. A majority class heuristic is also shown as a baseline.

| MODEL | ACCURACY % | AUC |
|---|---|---|
| KNN | 74.5 | 0.7182 |
| Naive Bayes | 71.8 | 0.6882 |
| Elastic Net | 78.4 | 0.7689 |
| Random Forest | 81.1 | 0.8031 |
| Gradient Boosted Machine | 77.3 | 0.7458 |
| SVM | 78.7 | 0.7729 |
| Neural Network | 74.1 | 0.7375 |
| Majority Vote | 79.9 | 0.7816 |
| Majority Class Heuristic | 57.6 | - |

All models beat the majority class heuristic by quite a margin, and all models were within 10 percentage points of each other for accuracy. The winner in both accuracy and auc was the random forest model, with the majority vote ensemble close behind.

## CONCLUSION

Overall, even the worst model, naïve bayes, performed decently. The best model, Random Forest, performed quite well. This is a difficult machine learning task. Inadequate documentation on how the keyword variable was assigned as well as the location variable being user chosen and optional made the dataset difficult to analyze. Despite these challenges an accuracy rate of 80% was achieved. With further knowledge in natural language processing and deep learning, an acceptable model for production use could be generated.

Kaggle uses F1 scores to grade submissions. Assuming the best model performed similarly on the Kaggle test set, it would be ranked 858 / 1154 on Kaggle's leaderboard for this challenge. I believe this project has shown how well even basic machine learning techniques can perform in the modern era on challenging tasks.

## REFERENCES

[1] Anna Kruspe, Jens Kersten, and Friederike Klan. (2020). Review article: Detection of actionable tweets in crisis events. Copernicus. https://nhess.copernicus.org/preprints/nhess-2020-214/nhess-2020-214-manuscript-version6.pdf

[2] Addison Howard, devrishi, Phil Culliton, Yufeng Guo. (2019). Natural Language Processing with Disaster Tweets. Kaggle. https://kaggle.com/competitions/nlp-getting-started

[3] Misri, I. (2021). Emergency tweet classification and model deployment. Medium. https://medium.com/@imisri1/emergency-tweet-classification-and-model-deployment-94c88e86981c