Detecting Natural Disasters Through Tweets: An Application of NLP

Tasnia Sarwar, Tylar Watson, Kipp Williams, Brennora Cameron

May 5, 2020

1 Background

1.1 Introduction

Even with only 280 characters per tweet, Twitter holds a wealth of useful information that awaits to be extracted. Following along with tweets in a particular locality can give insights into happenings in that region, ranging from the weather, crime reportings, and also disasters that might be occurring in real-time. Because of this, citizens, news agencies, and even first-responders can better react to important events happening around their community by simply tracking tweets. Over the past few years, the term "fake news" has gained traction, especially pertaining to the ambiguous credibility of news, information, and updates seen in social media platforms such as Twitter. This project looks into the concept of "fake news," specifically aiming to detect instances of real-life natural disasters in tweets by applying various machine learning techniques, especially as they pertain to NLP.

1.2 Data

The data set used is supplied by figure-eight and is available through Kaggle as csv files. Included is a training set of size 7613 tweets x 5 columns, and a test set of size 3263 tweets x 5 columns. The training set columns consist of

- **id**: a unique number identifying the tweet
- **text**: the content of the tweet
- **keyword**: any identified keywords in the tweet (often empty)
- **location**: the location of the tweet's author (often empty)
- **target**: 1 for disaster, 0 for not disaster

and the testing set includes all but the target variable. Given these tweets, we are performing a binary classification to predict whether a specific tweet is about a real disaster (a value of 1) or not (a value of 0).

It is not always clear-cut if a user's words are actually referring to a disaster. Take for example this tweet stating, "On plus side LOOK AT THE SKY LAST NIGHT IT WAS ABLAZE," attached with a picture of the sky. Though the user uses the keyword "ablaze," they are not actually referring to any disasters and the keyword is being used metaphorically.

1.3 Prior Research

We surveyed other projects and resources pertaining to a similar problem space before proceeding with our choice of methods. Most applications of natural language processing include a pre-processing step to clean and refine text data before performing tokenization. This step makes the data more consistent and easier to handle. Common in this cleaning step is making characters lowercase, trimming whitespace and removing non-standard characters (Dukare).

For the vectorization step, we found that two common tokenization techniques were a basic count vectorization, which applies a simple bag-of-words vectorization, and TF-IDF vectorization. During vectorization, we found that it is also helpful to remove so-called "stopwords" from the list of phrases considered in the model (Sun, et. al). These stopwords, which include words like "me" , "I" , and "that" , are common phrases in a natural language and add negligible meaning to a sentence from a machine learning standpoint, often adding noise to models (Leskovec).

There were a multitude of machine learning models people used, as this is a relatively straightforward classification problem, identifying if a tweet is about a natural disaster, or not. When surveying previous work done for this problem on the Kaggle platform, some of the most used models were logistic regression and SVM. We chose to do these models not because they were most used, but because they made sense and provided a learning opportunity in applying classification models. With the type of data we have and the problem we were trying to solve, a supervised learning model was a logical choice, and these two types of models are both useful for classification.

2 Methods

2.1 Pre-Processing Data

After reading in the testing set and training set and storing them in a `pandas` dataframe, we performed a cleaning step on the tweets to refine our data and used two different vectorization techniques for use in implementing our choice of models.

2.1.1 Cleaning Data

We implemented a custom cleaning function that

- makes text lowercase
- trims whitespace from the front and end of tweets
- cleans urls that contain variances of "https://" or "www."
- "uncontracts" contractions

- removes non-standard characters (newlines, hashtags, punctuation, etc.)
- remove numbers

For example, a tweet that was originally " I'm feeling # great today :)" would become "i am feeling great today" after cleaning.

2.1.2 Vectorizing Data (TF-IDF & Count Vectorizers)

We used the standard `TfidfVectorizer` and `CountVectorizer` supplied by scikit-learn to perform two different forms of tokenization on our data set ("Feature Extraction"). For each method, we removed stopwords as defined in NLTK Corpus' list of stopwords ("Accessing Text Corpora and Lexical Resources"). `CountVectorizer` performs a standard bag-of-words vectorization on a set of phrases. `TfidfVectorizer`, using Term Frequency-Inverse Document Frequency, does an extra step to normalize the set of phrases and their frequencies. Given a phrase i in a tweet j, TF-IDF assigns the phrase a numerical feature value as

$$x_{i,j} = \frac{\text{occurrences of} i \text{ in } j}{\text{words in } j} * [1 + \log \frac{\text{tweets}}{\text{tweets where } i \text{ occurs}}]$$

For both of these techniques, we chose to extract both monograms (single words) and bigrams (double-word phrases) as our features, resulting in $p = 62886$ features per tweet in the set.

2.2 Implementing Machine Learning Models

2.2.1 Logistic Regression

There are many libraries available to create and test logistic regression models; we decided to use scikit-learn. We first used the scikit-learn `train_test_split` method to divide the provided training data into "train" and "test" data. Because the TF-IDF form of tokenization was more detailed than a standard "bag of words," the `TfidfVectorizer` dataset was used. After splitting the data into `x_train`, `x_test`, `y_train`, and `y_test` we created a model using the `LogisticRegression()` method and fit the model to the `x_train` and `y_train` datasets. We then made predictions using the `predict()` method on the `x_test` data.

To test the model, we used the `classification_report()` method to get the precision, recall, f1-score, support, and accuracy values to evaluate the model. We also used `roc_auc_score` to get the AUC score, and used `pyplot` to graph the ROC curve.

Lastly, `metrics.confusion_matrix()` was used to get a confusion matrix for the logistic regression model, and seaborn's `seaborn.heatmap()` function, as well as pyplot, were used to create a visualization of the confusion matrix.

2.2.2 Support Vector Machines

We used scikit-learn to implement the Support Vector Machines. To split the labeled data into training and testing sets we used `train_test_split`. We used the `TfidfVectorizer` vectorized dataset for SVM. Then we used `GridSearchCV` to tune hyperparameters with cross-validation. We evaluated SVM models based on precision and recall for linear and gaussian kernels, trying different values for C and gamma. The two best combinations of parameters were linear SVM with $C = 1$ and gaussian SVM with $C = 1000$ and $gamma = 0.001$. We created a model for reach parameter combination with `SVC()` and fit the model to the `x_train` and `y_train` datasets. We then made predictions using the `predict()` method on the `x_test` data.

The `classification_report()` method was used while testing the model to get the precision, recall, and f1-score values. We also used `roc_auc_score` to get the area under the ROC curve. Next, `metrics.confusion_matrix()` was used to get a confusion matrix. Finally, `pyplot` was used to graph the ROC curves of both models.

3 Results & Conclusion

3.1 Logistic Regression

The `classification_report()` method gave the precision, recall, and f1-score of both classes: non-disaster tweet and disaster tweet. The precision for the disaster tweet class (DTC) was 90%, whereas the precision for the non-disaster tweet class (NDTC) was only 74%. However, the recall for NDTC was 95%, much higher than the recall for DTC which was only 54%. Looking at both of these metrics together, it's clear that although the logistic regression model did not have many false positives, it did better at classifying NDTC than DTC. That said, overall the accuracy of this model was 78%, which we find to be acceptable.

In addition to the classification report, an ROC curve was also plotted, and an AUC score was calculated:

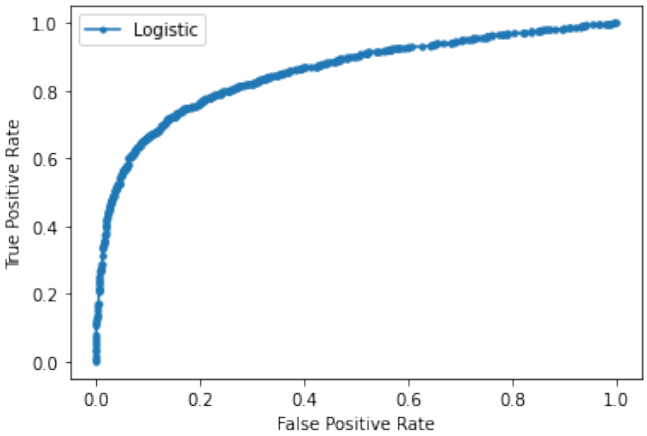

Figure 1: Logistic Regression ROC Curve

Figure 1 shows a plot of the ROC curve; the AUC score was calculated to be 85.4%, indicating that this logistic regression model makes good predictions. However, to get a closer look at the model, we also created a heatmap of the confusion matrix:

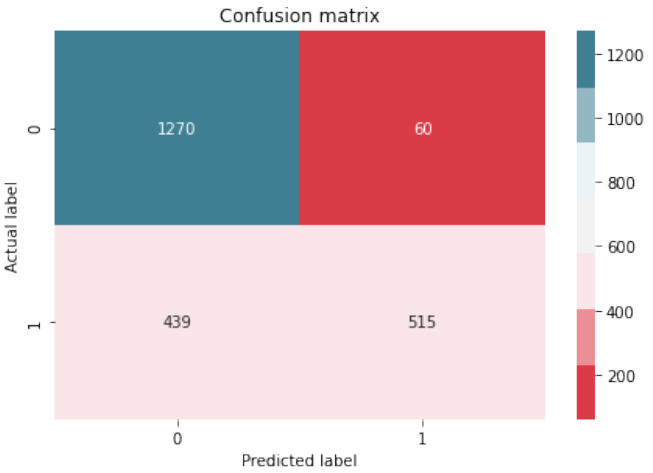

Figure 2: Logistic Regression Confusion Matrix Heatmap

This confusion matrix supports what we found with the classification report precision and recall values: this logistic regression is good at correctly identifying tweets belonging to the NDTC, but is not as good at correctly identifying tweets belonging to DTC.

3.2 Support Vector Machines

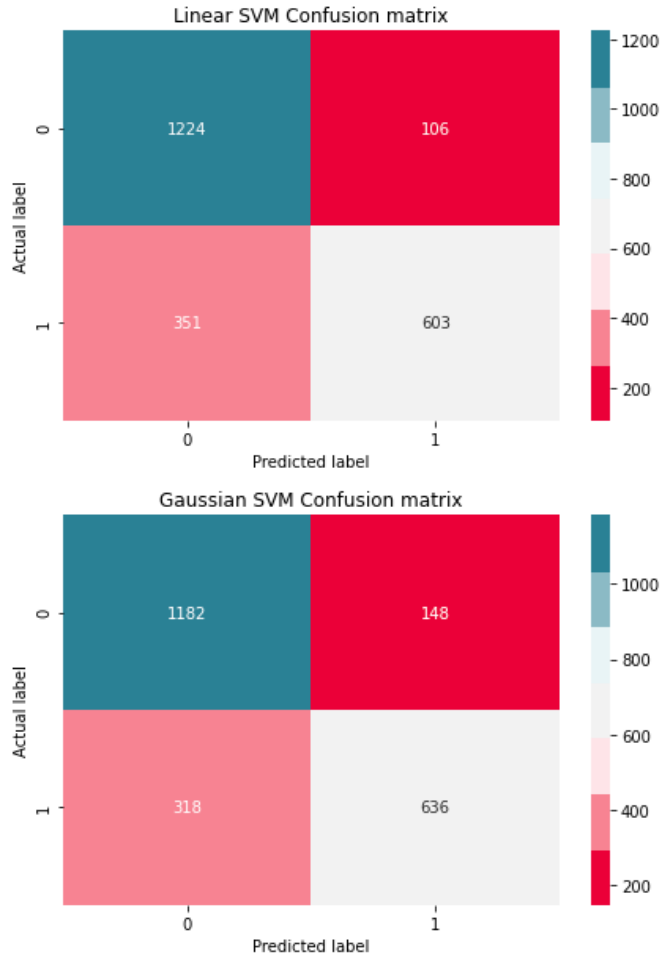

Figure 3: Confusion Matrices for Linear and Gaussian SVMs

The precision, recall, and f1-score for each model (Linear, Gaussian) of both non-disaster tweet and disaster tweet classes. The precision for the disaster tweet class (DTC) was 85% for the Linear SVM and 81% for the Gaussian SVM, whereas the precision for the non-disaster tweet class (NDTC) was 78% for Linear SVM and 79% for Gaussian SVM. The recall for NDTC was much higher than the recall for the DTC in both models, with Linear SVM showing 92% NDTC recall versus 63% DTC recall and Gaussian SVM showing 89% NDTC recall versus 67% DTC recall.

Shown in the figure above, the Linear SVM had more true negatives and less false negatives than the Gaussian SVM, but it also had more false positives and less true positives. Although each model has tradeoffs between level of precision and recall, they both did better at classifying the NDTC than DTC.

Figure 4 below shows a plot of the ROC curve; the AUC score was calculated to be 78% for both the Linear SVM and Gaussian SVM, indicating that these models make decent classifications. The ROC curves of each model also demonstrate, visually, how difficult it is to choose a superior and inferior model between the two.

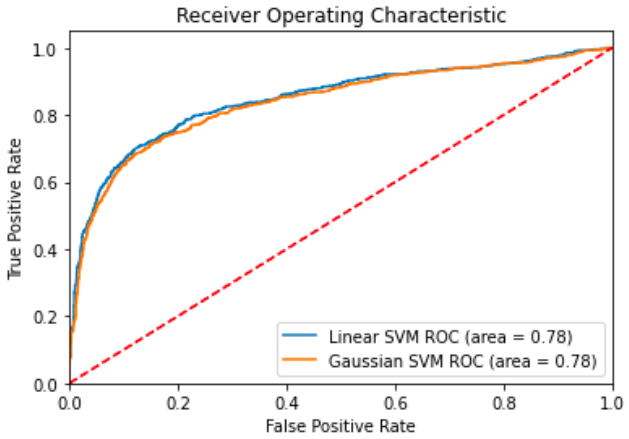

Figure 4: Linear and Gaussian SVM ROC Curves

The f-1 scores do not help determine which SVM model is better as they are equivalent for each: 0.84 for the NDTC and 0.73 for the DTC.

3.3 Conclusion

To compare the Logistic Regression, Linear SVM, and Gaussian SVM models, we can look at the f-1 scores: for the NDTC all three models have an f-1 score of 0.84 and for the DTC the two SVM models have a higher f-1 score of 0.73 in comparison to the 0.67 score for Logistic Regression. While there is room for improvement in each model, the two SVM models show more promise than the Logistic Regression model in classifying whether tweets are discussing natural disasters.

It appears that every model is best at correctly classifying non-disaster related tweets. This indicates the SVM models would be appropriate for news stations looking for sto-

ries on natural disasters since false negatives (disaster related tweets classified as non-disaster related) have low consequences, meaning that reporters miss out on a few potential stories. On the contrary, if a government agency is using the model to monitor the disaster or find people in need of emergency services, a false negative means that a person in need of assistance may go unheard and unhelped. Our results indicate that Linear or Gaussian SVM models are accurate enough to be useful for purposes such as helping news stations find stories about disasters in their communities, however, not accurate enough to be useful for government agencies trying to use social media as a monitoring platform during natural disasters. That being said, using machine learning to monitor natural disasters presents an amazing opportunity for improving disaster reporting and governmental response. Further research should explore using Recurrent Neural Networks to improve accuracy in classifying tweets about disasters for this important purpose.

4 References

"Feature Extraction." *Scikit-Learn*, scikit-learn.org/stable/modules/feature_extraction.html

"Classification Report." *Scikit-Learn*, https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html

"Accessing Text Corpora and Lexical Resources." *Natural Language Processing with Python*, by Steven Bird, O'Reilly Media, 2016.

Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. "Data Mining." *Mining of Massive Datasets.* Cambridge: Cambridge UP, 2014. 1-18. Print.

Dukare, Agasti Kishor. "Data Cleaning for NLP of Social Media Text in 2 Simple Steps." *Medium*, Towards Data Science, 11 Feb. 2020, towardsdatascience.com/data-cleaning-for-nlp-of-social-media-text-in-2-simple-steps-6ca48fa99c17.

Sun, Xiaobing, et al. "Empirical Studies on the NLP Techniques for Source Code Data Preprocessing." *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies - EAST 2014*, 2014, doi:10.1145/2627508.2627514.