**COMPSCI 9680A**

**Advanced Machine Learning**

**Final Report**

**Due: December 6, 2020**

**Submitted By:**

Kolby Sarson

**Abstract**

This research aims to produce a machine learning project which can accurately classify disaster tweets. We utilize a dataset which contains nearly 11,000 tweets, with both disaster tweets and non-disaster tweets. Given that the dataset consists of tweets, which are short passages of text, symbols, and various other characters, feature engineering acts as the primary method for improving performance of our classification. Properly utilizing tweets in classification requires adequate preprocessing, which acts as the core of this project. To actually determine whether a given tweet is a disaster tweet or not, we utilize a variety of models, including logistic regression, perceptron, random forest, and others.

This report will begin with an introduction, in which I discuss my motivation for this project, as well as a general outline of the work done throughout. This is followed by a discussion of the data itself; where it comes from, and its purpose. Next, we take a look at the methods used, both in preprocessing and in classification. Afterwards, we outline the results of our implemented models and their performance. Finally, we discuss these results in detail, considering their quality overall.

**Introduction**

As a new graduate student, I took this class hoping to broaden my knowledge in the field of machine learning. I also chose to take on the final project as an individual student. As a result, I decided to pursue a Kaggle competition as the basis for my final project. I felt that the work involved with a Kaggle competition would be suitable for an individual, as well as being very good for guided learning. Additionally, I wanted to pick a competition that would broaden my knowledge, and since we did not get any hands on work with Natural Language Processing (NLP), that is the topic I decided on for this project. This led me to the Kaggle competition titled, ‘Real or Not? NLP with Disaster Tweets.’

This project is a simple one, with the task at hand being the classification of tweets as either disaster tweets, or non-disaster tweets. When we refer to disaster tweets in this paper, we are referring to tweets referencing disasters in the real world, such as earthquakes or wildfires. Though the applications of a model that can accurately classify these tweets would be limited to disaster related fields, such as disaster relief or for use by news agencies, the ideas behind the project could be tweaked to apply to a wide range of NLP problems. The topics investigated are ones I could implement in future works.

The approach we will take to solve this problem begins with a quick look at the data, which is mostly just the tweet itself. After looking at the data we have to decide on how to proceed. Given that the dataset has so few features outside of the tweet, we know that we will need to do some feature engineering to prepare the data for a proper classification. Before doing this though we can implement a few machine learning models on the data as is to see some baseline scores. For a scoring metric we will use accuracy with cross validation. After we implement and test these models, we can then proceed to our feature engineering. We adjust the tweet to account for different issues such as consistency, unimportant pieces of text, and emojis or symbols. After concluding this feature engineering, we will reapply the previously initialized models on the newly adjusted dataset and analyze the new results.

**Data**

The dataset used in this competition is a dataset originally provided by Figure-Eight, a machine learning and artificial intelligence company. They offer a variety of open-source datasets, one of which being the one used in this competition. Unfortunately, the original dataset is no longer available through their datasets portal.

The dataset itself contains few features, three to be precise. These features include a single keyword from the tweet, a location which the tweet was written at, and the tweet itself. The only other information in the dataset is an arbitrary id, and the labels of whether the tweet was a disaster tweet or not.

**Methods**

This section will be divided into two subsections. The first subsection, preprocessing, will look at the data manipulation and feature engineering performed prior to applying our models. The second section will discuss the models we utilized to classify the tweets.

***Preprocessing***

In this stage, we take a look at the dataset and any ways in which we can improve it prior to classification. This can take place in many ways, but specifically for NLP, we typically look at adjusting the text itself. There are four adjustments that we can make on the tweets we are provided.

The first adjustment that we can perform is that we can rewrite contractions as their parts, such as rewriting “can’t” as “can not”, or taking “‘re” off the end of a contraction and replacing it with “are.” Making these alterations can create a consistency in varied texts which ultimately helps with classification through simplifying associations. This is handled simply through utilizing python’s ‘.sub()’ function from the ‘Regular Expression’ library to substitute text in a passage with alternate text.

The second adjustment we can make is what we can broadly refer to as normalization. In this step we utilize regex to change the text to meet standards common across all of the tweets. Some of the regex expressions we can apply include removing non-word characters, punctuation, brackets, and more. We can also perform a ‘to-lowercase’ operation on the tweets as part of our normalization step. By doing this we can make the text more consistent between tweets as well as removing noise at the same time.

The next alteration we can make on the tweets is to remove emojis. This step also includes the removal of other similar symbols and characters. To achieve this we can utilize unicode values to create a set of characters and symbols to be removed, once again using the ‘.sub()’ function. This idea was inspired by Kamil Slowikowski’s emoji removal script, shared openly on his GitHub. This step is similar to the normalization we did in the prior step, just shifting focus to the non-simplistic characters that regex cannot handle.

The final step we can take in our preprocessing is to remove a set of stopwords, which are words that are deemed as unimportant to the understanding of a passage. This is a very common step in NLP problems as it is a great way to clean the data during preprocessing. In this project we can use python’s ‘Natural Language Toolkit’ library and its english stopwords ‘corpus,’ or dictionary. We then use a count vectorizer, to convert the tweets to vectors of words, while simultaneously removing stopwords.

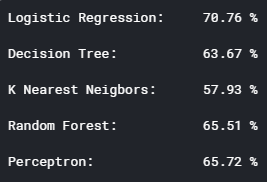
After applying these four preprocessing adjustments to our tweets, we find ourselves with a dataset that is much cleaner and less noisy. With this dataset in hand, we can proceed to the second phase of the project, which is the classification stage using machine learning models.

***Machine Learning Models***

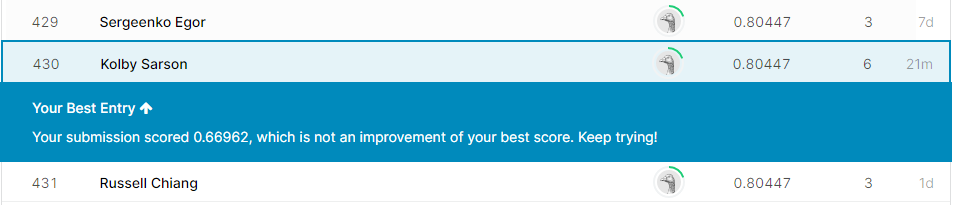
In the second step of our project, we look at the classification of the now preprocessed tweets. We have implemented five models in this project including logistic regression, decision tree, random forest, K nearest neighbours, and perceptron. For each of the five models, we fit them to a training vector set created from the preprocessed data so that we can then apply the trained model to the test vector set. Each of these models was run with their default parameters for simplicity, as well as with the belief that for this problem, the preprocessing was the more significant step for improving performance. This can be seen with an application of ‘gridSearchCV()’ applied to the logistic regression model. Upon running the grid search and finding the best parameters, applying the logistic regression model a second time with the best parameters showed no improvement on the original score when run with the default parameters.

**Results**

In this section we will outline the results obtained from our models. The scoring metric implemented to evaluate performance was accuracy with cross validation. Each cross validation was performed with 5 folds and a mean accuracy was taken and used as the final score. Given the format of Kaggle competitions, the training data has labels while the test data does not. As a result, when designing models, you can only obtain training set performance scores, whereas to obtain test set performance scores you must make a submission. Below you can see the cross validated accuracy scores of the five models applied to the preprocessed training data.

The scores achieved by the models on the test set are as follows: Logistic Regression score 80.45%, Decision Tree scored 76.86%, K Nearest Neighbours scored 66.96%, Random Forest scored 78.18%, and Perceptron score 77.81%. These scores are all 9-13% better than the training data leads us to believe prior to submission.

Evidently, the highest scoring model implemented was the logistic regression, achieving a score of 80.45% on the test data as per Kaggle. This score placed me in the top 500 on the leaderboard. Additionally, only the top 72 submissions were able to exceed a score of 85%, indicating that a score of 80% is still very strong overall. Below is a screenshot of this model’s position in the leaderboard.



**Conclusion**

Throughout this competition, we conducted research and followed the typical requirements that would be expected in a machine learning project. We investigated the data, collected baseline scores, preprocessed the data, and applied various machine learning models. We then analyze our results to evaluate the performance of our proposed model. We were able to produce a top performing classification on the disaster tweets. This kind of accuracy could be considered sufficient for use in practice. For example, a disaster relief organization could use this data to improve response times during disasters occuring around the world.

We expanded on our previous knowledge and experience working with machine learning models for classification by adding the dimension of NLP to the problem. As we saw in previous works during the semester, working with and adjusting machine learning models can be a solid way to improve performance. But with NLP problems, the preprocessing of the data appears to require much more attention.

**References**

“Remove All Traces of Emoji from a Text File.” *Gist*, gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b.

“All You Need to Know about Text Preprocessing for NLP and Machine Learning.” *KDnuggets*, www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html.

“Guide to Text Classification with Machine Learning.” *MonkeyLearn*, monkeylearn.com/text-classification/.

Koenig, Rachel. “NLP for Beginners: Cleaning & Preprocessing Text Data.” *Medium*, Towards Data Science, 29 July 2019, towardsdatascience.com/nlp-for-beginners-cleaning-preprocessing-text-data-ae8e306bef0f.

Weng, Jiahao. “NLP Text Preprocessing: A Practical Guide and Template.” *Medium*, Towards Data Science, 1 Sept. 2019, towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79.