COMP 562 Final Project

Max Christman, Jarem Saunders, Shiva Kammala,
Pradham Tanikella, and Aneesh Murthi

May 5th, 2022

1 Motivation

As social media technology increases in accessibility throughout the world, the prevalence of misinformation has inevitably risen. With this in mind, my group sought to create a model that analyzes data taken from Twitter that correspond to disaster scenarios and analyze whether a given tweet references a real disaster. There are several use cases for the classification of tweets such as being able to sort through large quantities of data to identify genuine disaster-related tweets. In a disaster scenario, it is important to quickly broadcast relevant information about an event. After creating an initial model for the disaster tweets using both random forest classifiers and multilayer perceptron classifiers, the group realized that the dataset itself was not optimal for a high accuracy model due to issues with data labeling. We then switched to a dataset of funny tweets and redesigned our model to identify whether or not a tweet was actually humorous. Classifying humorous tweets has wide applications in industry where massive amounts of communication needs to be sorted to be presented to users.

2 Related work

This analysis examines both random forests and neural networks as models for Twitter text classification. A preliminary survey of articles on text classification seems to suggest that neural networks typically outperform random forests in text classification, though this is not always the case. Both methods are relatively popular and can be highly effective if optimized.

This analysis will also examine n-grams, rather than simple bag-of-words approaches we have used earlier in this course. N-grams have been noted to improve the performance of basic NLP classification tasks by better representing relationships among words and giving more data points to sample from within a single text.

3 Disaster tweets

This project examines two different Twitter language data classification problems using random forests and neural networks. Initially, we examined the Kaggle dataset on identifying tweets about disasters and built binary classification models using random forests and neural networks. Language data vectorization and the base algorithms for our models were implemented using sklearn Test and training data came from the .csv files provided for the Kaggle competitions, and included the text data for about 7000 tweets for the disaster corpus and 200,000 for the humor identification corpus.

We performed minimal standardization on the text data - strings were all cast to lowercase and hyperlinks (typically image embeds) were removed. We decided that from all remaining characters in the strings, we would only keep letters, numbers, and a few accented characters. We also kept # and @ due to their importance on Twitter. Text data was not lemmatized or stemmed.

The tweet text data was processed into vector representation as n-grams (all substrings of n adjacent tokens) using the sklearn CountVectorizer function with ngram_range parameter.These vectors were then used to create a random forest model using the sklearn RandomForestClassifier() function. Without hyperparameter tuning, the random forest model gave the following F1 cross validation scores:

$$\begin{aligned}\text{Monogram only} &\Rightarrow 0.562728\\ \text{Bigram only} &\Rightarrow 0.23036\\ \text{Monogram and bigram} &\Rightarrow 0.563051\end{aligned}$$

Using GridSearch, we looked for the optimal hyperparameter values using cross validation. Initially, we did cross-validation across maximum depth and number of estimators. However, these models all gave fairly poor accuracy. Instead, we switched to cross-validation across minimum samples for a split and minimum samples for a leaf. These hyperparameters allowed us to get better results. Through this second round of cross validation, we got the hyperparameters with the highest average accuracy as:

$$\begin{aligned}\text{min-samples-leaf} &= 2\\ \text{min-samples-split} &= 4\\ \text{n-estimators} &= 500\end{aligned}$$

3.1 Predicting test data

As the disaster tweets dataset is from a Kaggle competition, the creators chose to not make the test labels public. As such, we have the random forest model make predictions on the test data. We then submitted this data to Kaggle to get an accuracy. According to Kaggle, these parameters gave an F1 score of 0.74103.

Given the overall poor performance of our random forest, even after optimizing hyperparameters, we opted to switch to a neural network model. We used sklearn's MLPClassifier to implement a multilayer perceptron model. Using our same vector representations of text, we performed another gridsearch to learn hyperparameters, testing out $\alpha \in \{0.0001, 0.001, 0.01, 0.1\}$. The grid search yielded $\alpha = 0.0001$ as the best value. We then used the MLPClassifier to get predictions for the bigrams test dataset. After submitting the predicted test data to Kaggle, we received an F1 score of 0.76156. Due to undesirably low accuracy, we tried to train a multilayer perceptron classifier again, this time with tweet data represented as tri-grams rather than bi-grams. Unfortunately, this actually diminished the accuracy, classifying 0.70303 of the results correctly.

4 Detecting humor in text

After manual inspection and additional research on the disaster tweets dataset, we determined that the dataset itself was low quality, i.e. a number of the tweets were labeled incorrectly or did not in fact refer to a disaster of any kind. Instead, we decided to apply similar natural language techniques to another dataset. We found a dataset of 200,000 short passages of text (similar to tweets) which are labeled as either having humor or not having humor. Others' work on Kaggle gave us the impression that this dataset is high quality. We then proceeded much as before on this dataset.

Due to the time limitations of this project, we decided to train on only 10 percent of the data, as there are 200,000 data points. Training on, for example, 60 percent of the data would take prohibitively long (more than 10 minutes for one fit). However, evaluation time is more reasonable than training time, so we can still test on the remaining 90 percent of the data. Using cross-validation to train the model, we determined the optimal number of estimators for Random Forest Classification to be 500.

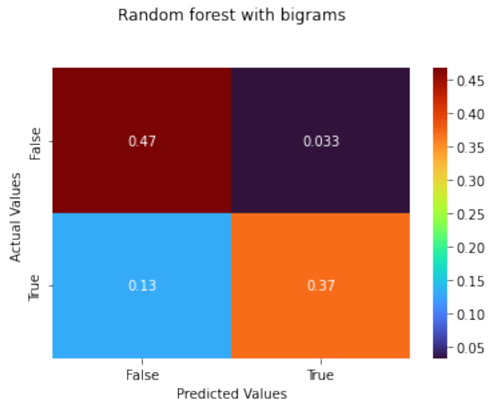
The humor dataset, unlike the disaster tweets dataset, included labels for every piece of text in the dataset. As such, we are able to directly compute accuracy/F1 scores and also to plot confusion matrices.

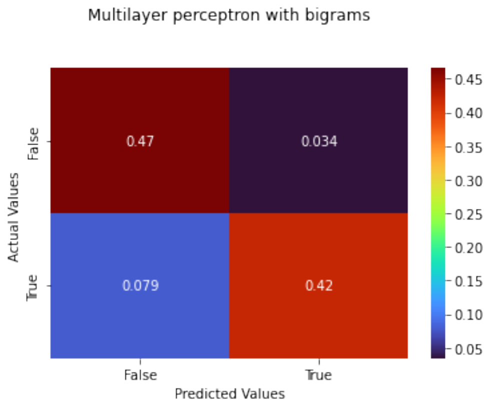
4.1 Predicting test data

Using a methodology similar to the one we used on the disaster tweets dataset, we used the CountVectorizer function to split the tweet data into bigrams for a preliminary analysis using random forests. A random forest classifier on the humor data was fairly effective, getting an accuracy of 0.836.

4.2 Neural networks and n-gram combinations

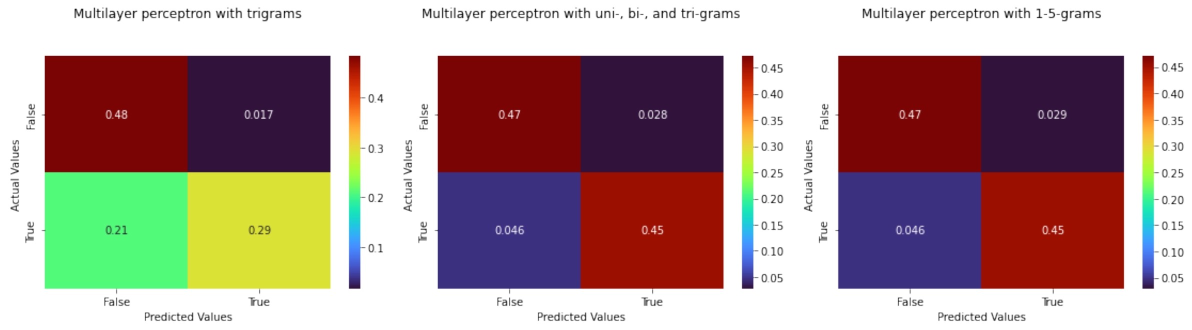
Since we had seen an improvement on a similar dataset by switching to neural networks, we decided to again try a multilayer perceptron model on the data. The MLPClassifier was used once again, and hyperparameter GridSearch gave the same value for this dataset, $\alpha = 0.0001$. The multilayer perceptron classifier once again fared even better than random forest, getting an accuracy of 0.887 on the test data.





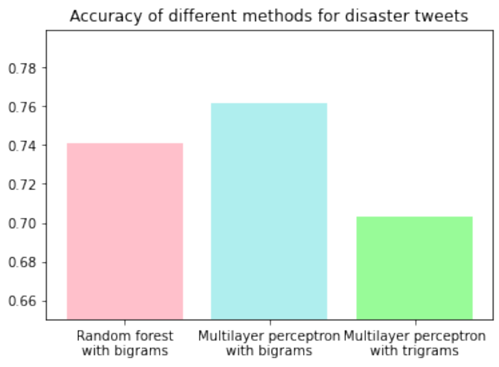
A second pass was made using MLP net, this time using trigram embedding. Interestingly, the MLP classifier performed significantly worse when trained on only trigrams, giving an accuracy of 0.774. Note, however, that its performance on fake disaster tweets was excellent, while its performance on true disaster tweets was only slightly better than chance.

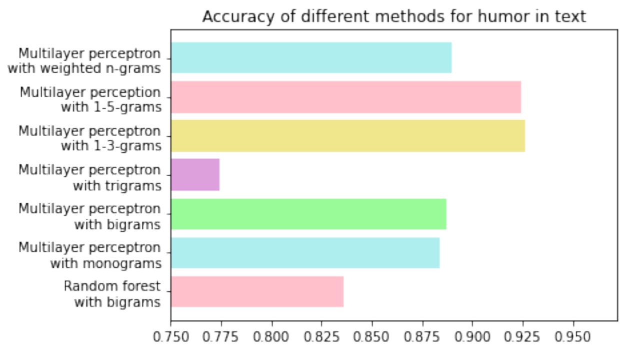
Next, the group came to the conclusion to train the model using single words, bigrams, and trigrams concurrently rather than in separate trials. Using 1, 2, and 3-grams, we were able to achieve better accuracy than with any previous method, getting 0.926. Unfortunately, it would take prohibitively long to cross-validate on additional parameters for the multigrams, as it took 15 minutes to train on a single set of parameters. To test this, we predicted values for the test-data using 1-, 2-, 3-, 4-, and 5-grams, which gave a very similar accuracy of 0.924 and demonstrated that the additional parameters were not necessary or beneficial to the model.



As an additional step, we wanted to examine whether assigning our own feature weights to monograms, bigrams, and trigrams predictions could yield improved performance over a model that treated 1-3 grams as a single pool of vectorized information. In order to meet the time requirements of this project, we pre-trained 3 separate MPL net models with 1-3 n-gram vectorization on the same 10% of the data we had previously been using as training data and then weighted the predictions of these models according to three weighting parameters. As each of these n-gram models are pre-trained on the training data, we need to further split the test data into "training" and "test" data. These "training" data will be used to evaluate the accuracy of the combined model on a particular set of weights, while the "test" data will be used to score the best set of weights after the fact.

5 Results and conclusion





We found that despite having very different domains and levels of training data noise, both data sets were better predicted by using neural networks rather than the Random Forest Classifier method. This seems consistent with the general findings on supervised text classification. We found no advantage to pre-training different n-grams and then iterating to find their weighting. In theory, this could introduce another variable, similar to a hyperparameter, that could be tuned to further optimize a pre-trained model, but in our experiment it did not yield any better results than the model that was trained on aggregated 1-3 grams. The weighting also happened to be best when the models were all equally weighted. It is possible some effect could be found if the weights for different n-gram levels were deliberately tuned during the training stage, rather than trying to weight them after training separately, though it may prove ineffective in this case as well.