Hate Speech Classifier

Leon Zhou

Project Design

The goal of this project was to create a classifier to identify tweets containing hate speech. Incoming tweets would be classified as either hateful, offensive but not hateful, or neither offensive or hateful.

This was approached by normalizing and tokenizing the tweets, and corresponding parts of speech, computing the tf-idf weight vectors of each tweet, and inputting it into a variety of classification algorithms.

Tools

This project relied heavily upon the Natural Language Toolkit (`nltk`) and `sklearn` packages. The `nltk` package provided extensive corpuses of text with part-of-speech labeling, necessary for training a part-of-speech classifier. In addition to the myriad of classification algorithms present within `sklearn`, it also allowed for the easy conversion of text tokens to tf-idf weights.

Other incorporated packages include `pandas` and `numpy` for data manipulation and matrix operations, as well as `keras`, in creation of the convolutional neural network model.

Data

The data contained over 20,000 individual tweets, with a human-classified label identifying the tweet as hateful, merely offensive, or neither. Approximately 80% of the data was classified in the "offensive" category, making the "hateful" class a small minority of the available data. Exacerbating this imbalance was the fact that the "hateful" and "offensive" categories were had many semantic similarities, which would make it comparatively difficult for a classification algorithm to correctly distinguish between the two.

The text of the tweets were normalized to correct for spelling, case, and special characters. Words were stemmed using the Porter stemming algorithm to reduce the variation in similar words, before being divided into unigrams, bigrams, and trigrams before being vectorized for input into the classification model.

Algorithms and Approaches

Part-of-Speech Classification

The first classification algorithm used was the part-of-speech tagger. This was a Naive Bayes classifier trained upon the Brown tagged corpus. The reported accuracy and F1-score of this classifier was approximately 85%. In real-world use, the classifier will likely not perform as well, as much of the slang and profanity in use on the internet today is not likely to be in the Brown corpus.

Basic Classifiers

For the primary classification task, five different types of models were trained on one of six different kinds of extracted features. These models were decision trees, Naive Bayes, gradient boosted trees, logistic regression, and stochastic gradient descent. The features were either unigrams, bigrams, or trigrams of the words themselves, or the corresponding parts of speech.

At first glance, the majority of these models seemed to perform well, generating an appreciable F1 score. However, due to the aforementioned heavy class imbalance, this is a highly deceptive result - the F1 score for hate speech, the class of interest, ranged from an abysmal 0% to around 25%, with the decision tree variant models showing the highest performance.

Average F1 Score, 3 Classes - Stratified Sampling

Model	Unigram	Uni-POS	Bigram	Bi-POS	Trigram	Tri-POS
Decision Tree	0.8625	0.5746	0.8605	0.6707	0.6663	0.8546
Gradient Boosted Trees	0.7109	0.8378	0.8761	0.7092	0.8635	0.7081
Logistic Regression	0.6981	0.8560	0.7002	0.8763	0.8553	0.6985
Naive Bayes	0.3677	0.1172	0.1210	0.3725	0.1248	0.3716
Stochastic Gradient Desc.	0.7791	0.6845	0.7876	0.6886	0.8072	0.6850

Param

Avg. Avgf: 0.1172 – 0.8678

Class 0 (Hate Speech) F1 Score, 3 Classes - Stratified

Model	Unigram	Uni-POS	Bigram	Bi-POS	Trigram	Tri-POS
Decision Tree	0.2650	0.0916	0.2519	0.0758	0.0918	0.2298
Gradient Boosted Trees	0.0338	0.2593	0.2473	0.0343	0.2393	0.0259
Logistic Regression	0.0000	0.2011	0.0000	0.2505	0.1953	0.0000
Naive Bayes	0.1134	0.1118	0.1122	0.1186	0.1127	0.1133
Stochastic Gradient Desc.	0.0707	0.0000	0.0664	0.0000	0.1924	0.0000

Param

Avg. COF: 0.0000 – 0.2650

Figure 1. F1-Score taking the average (top), or “hate speech” class individually (bottom).

Compound Models

With this discouraging first result, the viability of an ensemble model was examined next. The three best-performing decision tree models and two gradient boosted tree models were used to classify observations, and the majority opinion was taken as the decision result. This methodology resulted in a F1-score of 0.21 for the “hate speech” class - a result lower than that of any one individual model.

For the an alternative approach, instead of taking the majority opinion, an additional Naive Bayes classifier was trained to predict the final class from the individual model outputs, the rationale being that each individual model might hopefully, at least, misclassify consistently, allowing an additional model to adjust the final classification accordingly. Indeed, this was the case, and a F1-score of 0.31 resulted.

Convolutional Neural Networks

Further experimentation was conducted using deep learning methods. A basic one-dimensional convolutional neural network was constructed, with two layers of convolution and max-pooling, followed by a dense and classification layer. The first convolution layer had 32 filters of size 50, and the second convolution layer had 64 filters of size 20. The pooling layers moved in strides of 10. Dropout was added after each pooling step with a probability of 25%, and before the final output layer, with a probability of 40%.

The model was run for 30 epochs, and investigation of the validation accuracy and loss showed severe overtraining, with the optimal training duration at about 10 epochs. However, investigation of the end-of-epoch F1-score for class 0 showed no improvement, varying from 0.00 to 0.34 on the validation set.

Miscellaneous Approaches

An initial approach was to collapse the three available classes into two, simplifying the question to a binary classification problem. However, due to the overwhelming majority of offensive category tweets, no matter which category it was placed with, it dominated the result of the classification.

Hateful + Offensive Classification F1 Score (Binary) - Stratified

Model	Param: Unigram	Uni-POS	Bigram	Bi-POS	Trigram	Tri-POS
Decision Tree	0.9594	0.8555	0.9609	0.8545	0.8520	0.9575
Gradient Boosted Trees	0.9051	0.9689	0.9632	0.9056	0.9645	0.9061
Logistic Regression	0.9087	0.9630	0.9085	0.9664	0.9574	0.9086
Naive Bayes	0.4713	0.1666	0.1822	0.4802	0.1754	0.4751
Stochastic Gradient Desc.	0.9328	0.9076	0.9274	0.9089	0.9330	0.9091

COF: 0.1666 — 0.9689

Figure 2. Binary classification problem, combining “hate” and “offensive” classes.

With a vocabulary size of approximately 10000 unique tokens, overfitting due to high dimensionality was a concern. PCA was performed on the feature sets, reducing the vectors to 50 or 250 components. Classification on these transformed vectors did not improve results, however.

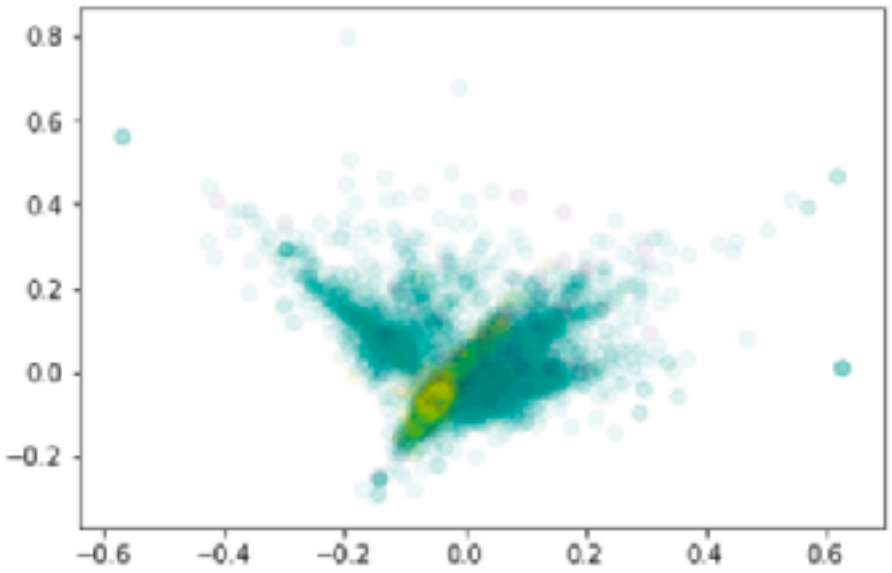

Figure 3. PCA with 2 components. Coloring corresponds to class of tweet.

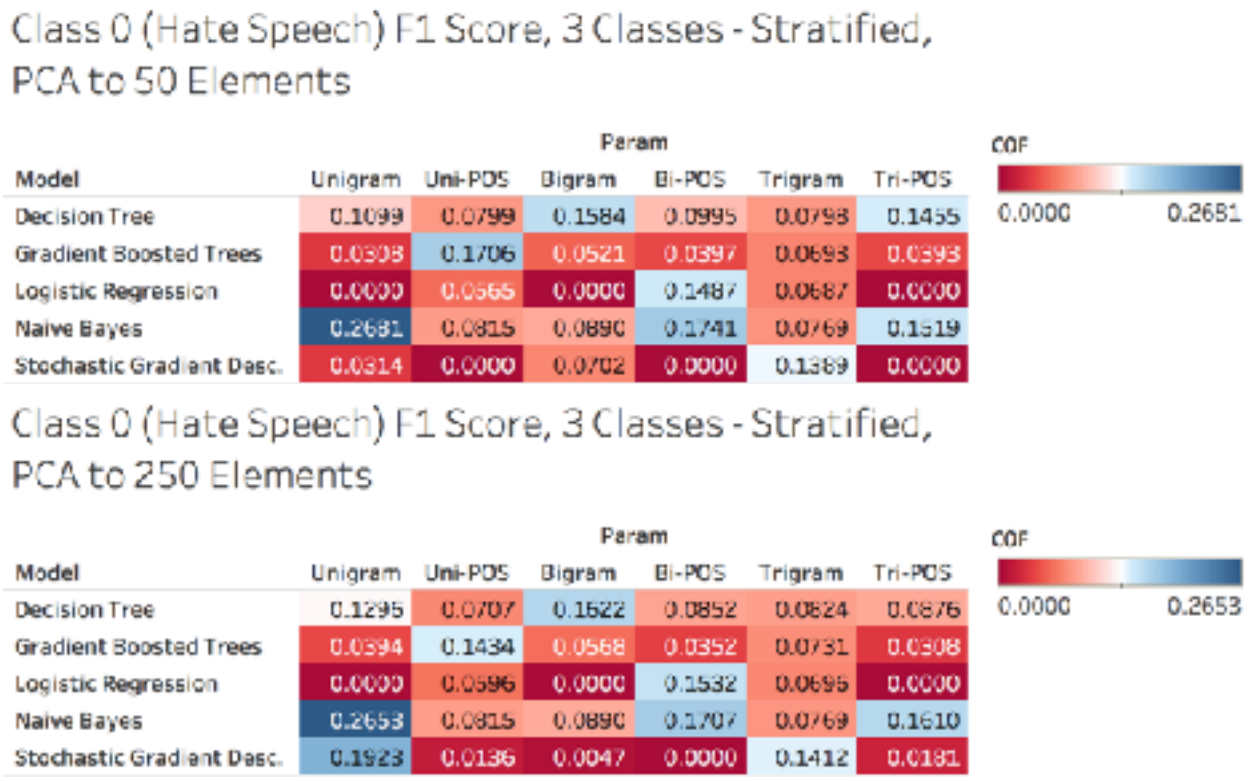

Figure 4. Result of classification on decomposed features, to 50 (above) or 250 (below).

Doing it Over Again

Refactoring

There were many aspects of code structure and style that could have been improved and would have resulted in a more robust and clean product; these were only discovered through trial by error, after a point in which refactoring code would be cost-prohibitive.

Refactoring my '`generate_tfidf_split()`' function to return the `TfidfVectorizer` object would be a significant improvement. Without the vectorizer object, the trained models cannot be used to predict new data. This was not realized until very far in the process, after which the function was extensively implemented and embedded in existing code. In the future, instead of returning a train-test split, the vectorizer (and, where necessary, a scaler) ought to be returned by the function so that the trained models may continue to be used.

The code structure for this project waffled between providing a generalizable and customizable interface and providing more optimized code specific to the project application. One style should be stuck with, as this mishmash of user interactivity is confusing and the unnecessary control structures needed to implement such an interface interferes with code readability.

Rebalancing

Class distribution and imbalance was not accounted for in the initial modeling. Downsampling of the offensive tweet class should allow the model space to more cleanly distinguish between the classes. Alternatively, if the priority is to retain all the original data, an classifying algorithm specialized for class imbalance could be utilized.

Neural Network Optimizations

Neural networks more specialized at handling text sequences, such as those within the recurrent neural network family may be better positioned to distinguish between the classes.