Alan Kessler
MSDS 453, Section 55 Summer 2019

Assignment 2/3 – Final Project Status Update

Intro and Problem Statement

The goal for this final project is to take concepts discussed in the course and scale them to a larger corpus and use this corpus to create a useful classifier. The dataset is a collection of Wikipedia comments. Each comment is labeled in terms of whether it is toxic or not in six different categories. The dataset is available through Kaggle.com, and the goal of the contest was to build a classifier for these labels using the text data. Many of the successful solutions discussed on the site involve using pre-trained embeddings and deep learning, but I want to analyze the use of term frequency inverse term frequency (TF-IDF) vectorizer features to create an interpretable way to analyze a given comment. For this status update, much of the time spent on the project has covered the processing of the text data prior to feature engineering. Future work will include enhancing both the processing and the classifier to improve on a baseline approach.

The comments themselves and the features used to classify them contain offensive and hateful language. This analysis is not a comment on the content of the comments but an exercise on how to use them for classification. All of the code for the project is included in a GitHub repository if needed as discussed in the assignment instructions. First, I will share aspects of the data structure how it is cleaned. Then I will discuss the modeling methodology used along with some next steps for the final project.

Data

The data for the project is a collection of comments from Wikipedia. There is a total of 159,571 comments in the training data and 153,165 comments in the testing data. Each comment is labeled for whether it demonstrates toxic behavior. There is a total of six categories of toxicity: “toxic”, “severe_toxic”, “obscene”, “threat”, “insult”, and “identity_hate”. Each comment may hit multiple categories or none at all. None of the raw data has missing or blank comments. Figure 1 shows the occurrence rate of each label in the training data. “Toxic” is the most common category with others falling into what are not exactly sub-categories of types of toxicity. These values are calculated using “label_eda.py”.

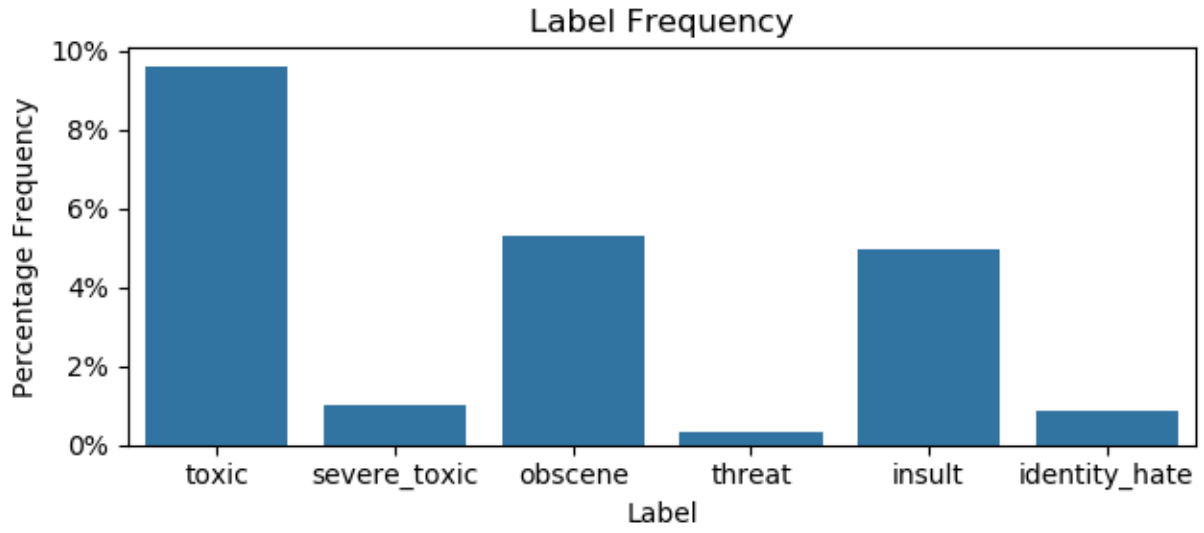

Figure 1 – Label Occurrence in the Training Data

The size of this data is significantly greater than the other examples in the course. This presents a challenge where processing all of the comments takes a significant amount of time and it is unreasonable to manually analyze a large percentage of the comments. To develop a text processing or cleaning process, I sample 100 comments from the training data to analyze more thoroughly. This sampling occurs in “mini_corpus.py”. Figure 2 shows examples of the unprocessed comments. Punctuation and formatting are common in these examples.

Example Comments Prior to Processing
Eelam War IV \n\nCould you please see Eelam War IV.\nThanks.
See \n\nYou soon, loser.
He is bad man! He makes much vandalism! 177.154.145.102
\nNoted. Just trying to be helpful) | talk

Figure 2 – Example Comments (Unprocessed)

The first processing steps were taken from the code provided in the course. Punctuation is removed along with non-alphabetical tokens. Short tokens are removed as well but the threshold is reduced to two characters to avoid missing out on the curse words and slurs that will likely be predictive in the model. While capitalization could be significant given the context of toxicity, it introduces too many variations such that all tokens are made lowercase. Finally, English stop words are removed.

For this exercise, I am taking additional steps in processing the data. First, the comments tend to reference URLs that are usually long tokens and should be removed. I also perform lemmatization. An enhancement to this process is that I identify the part of speech to feed into the lemmatization as additional context. Finally, I add negations from the NLTK package and remove the word “talk” if it is the last word of the comment. From my understanding, that is a function of how Wikipedia works and not really content of the comment itself. The same examples after processing are included in Figure 3. These comments are more easily incorporated into an algorithm like TF-IDF. A future enhancement would be to analyze the order of the processing to see if that improves results.

Example Comments After Processing
eelam war could please see eelam war thanks
see soon loser
bad man make much vandalism
note try helpful

Figure 3 – Example Comments (Processed)

The next step is to apply this processing to both the training and testing data sets. This is particularly challenging because it involves processing over 300,000 records individually. The code shared in the course processes a given corpus as a list to run through the cleaning function. To increase efficiency, I apply the function directly to the dataframe. This enables efficiencies such as iterators within the Pandas module. The next step in making this process more efficient is the use of the Dask module. This splits up the dataframe into six partitions to take advantage of more of the available CPU. The result is a process that takes approximately 47 minutes on an average desktop computer, and due to the time it takes to run, the results are persisted as new datasets. The full script to execute the cleaning is “full_processing.py”.

Methodology

The baseline modeling methodology is included in “baseline_classification.py”. The first step in the modeling process is to load the persisted cleaned text data. The algorithm used to create features for modeling is TF-IDF. The comments from both the training and testing data are combined prior to calculating the TF-IDF values. This is required in order to generate scores for the out of sample testing data because the features need to be present for both set of comments. Typically when using testing data

in a context like this, a concern would be introducing leakage, but in this case the TF-IDF algorithm runs independently on each comment.

The resulting vectorizer is used to transform both the training and testing data generating feature sets for both. Only 1,000 single-token terms are considered in order to reduce the amount of time required to train the models for a baseline approach. A sublinear term frequency is used to account for the longer comments such that term frequency does not reflect the weight of a term in a one-to-one relationship. The formula for this approach is shown below.

$$tf'_i = 1 + \log(tf_i)$$

A logistic regression is trained to the features independently for each class. The regression uses Lasso (L1) regularization with the Stochastic Average Gradient (SAGA) solver. Lasso regression has the ability to give features a weight of zero which is useful for analyzing importance and the SAGA solver is helpful in achieving model convergence for large data. The model is used to generate feature importance, cross-validation results, and predicted probabilities on the test data. The full process as described above is shown in Figure 4.

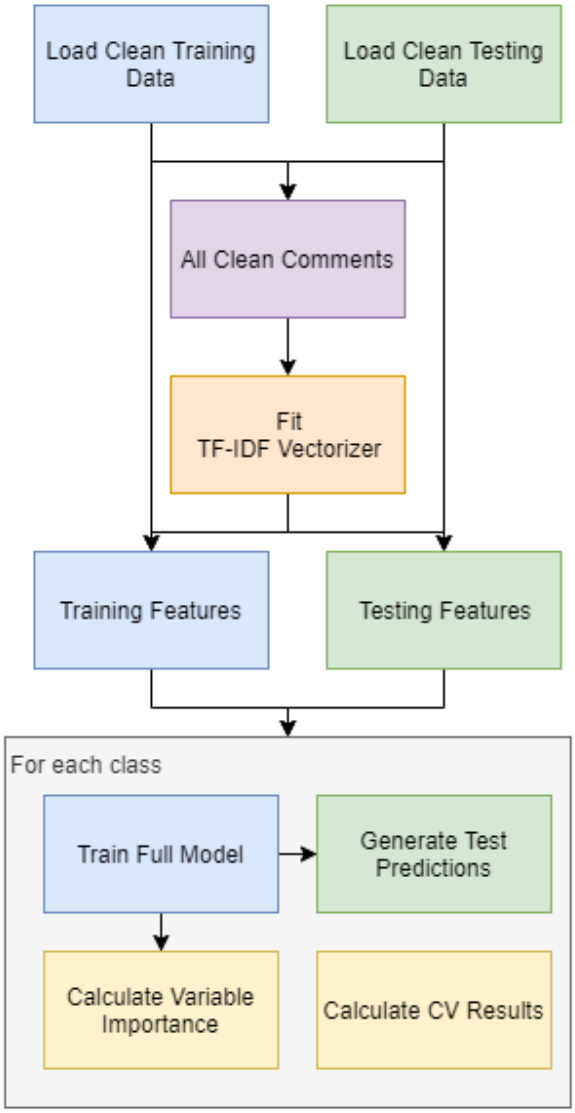

Figure 4 – Modeling Methodology

Preliminary Results

The preliminary results show that as expected the baseline model falls well short of top of the Kaggle leaderboard. The challenge evaluates submissions based on the average of each class's AUC. The private leaderboard AUC is **0.94951** while the public score is **0.95071**. While that performance would not win any awards, it is a significant step for a baseline model, and the AUC shows that the model is excellent at ranking comments in terms of their toxicity.

The baseline training is also performed with five-fold cross validation. This allows for an analysis by class as well. The resulting out-of-fold AUC by class is shown in Figure 5. These results show that the baseline modeling approach is best at ranking the "severe_toxic" class and worst at ranking the "threat" class. Intuitively, this makes sense because a threat may require additional context in the comment where a curse word-filled comment is a more obvious to an algorithm like TF-IDF based purely on terms alone.

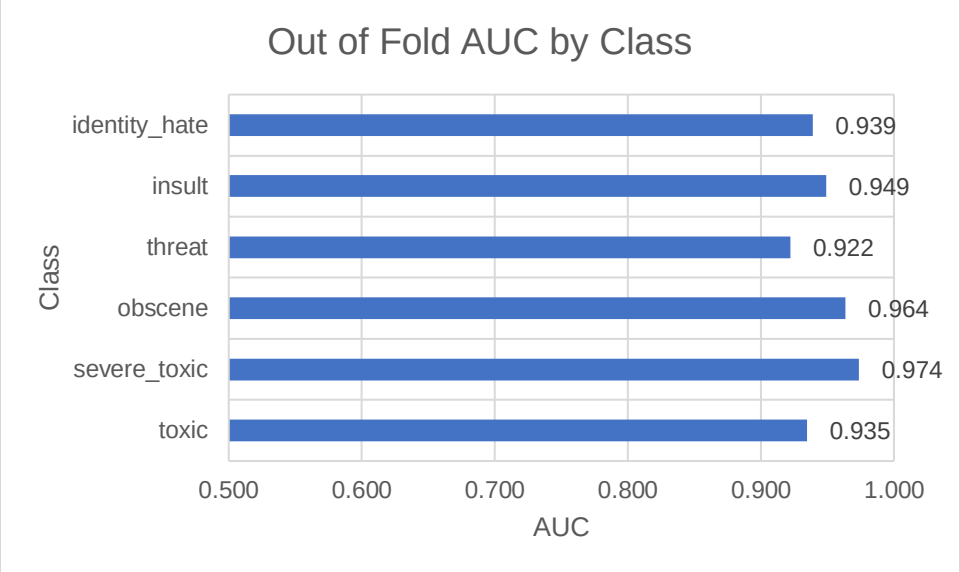

Figure 5 – Cross-Validation Results

One of the advantages of the TF-IDF approach coupled with the use of logistic regression, is that features and their impacts are readily apparent. Figure 6 shows the feature importance for the "threat" class logistic regression for the 20 most important features. The use of regularization produces standardized coefficients whose absolute values provide the basis for calculating feature importance. The columns in orange indicate features that reduce the likelihood of the comment being identified as a threat while blue is the opposite. Each of the features represent the TF-IDF value of the term. The features are very intuitive where terms like "kill" are seen as threatening and "source" is not. It also provides a chance to see the impact that features with negations have on the model. In some cases, such as the term "ill", both the negation and the regular term are significant and show the same direction but anecdotally, it appears that the negated version is more significant to the model. The other models show similar relationships where slurs and curse words are the most significant features. They are not included as they could be viewed as particularly offensive.

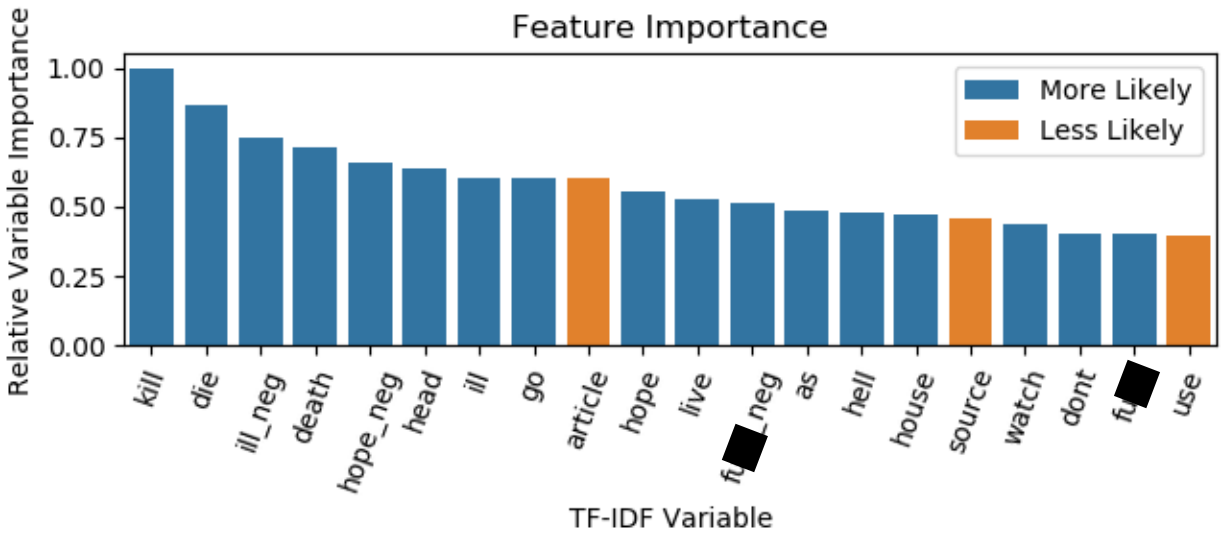

Figure 6 – Feature Importance "Threat"

Future Steps for the Final Project

The main objective for the remainder of the final project is to improve on the baseline and ultimately produce a model more in line with those making up the body of the Kaggle leaderboard. The majority of the time spent on the project has gone towards cleaning the comment data. There are plenty of ways to improve on the baseline while keeping much of its structure consistent. For the sake of this project, the extensive use of deep learning with pre-trained embeddings is out of scope given the amount that can be improved on with this baseline. The questions that I want to answer for the final project include:

- Does changing the order of some of the processing improve fit?
- Do the negations benefit the model score or just add complexity?
- Does increasing the number of TF-IDF features improve the fit?
- Will adding n-grams of greater size (e.g. two to three terms) improve the model?
- Does the type of generalization and generalization power impact the results?
- Is it possible to use the estimates of one class to improve others?

My plan is to work through each of the options one at a time. While ideally, I would optimize more of these at once, it is reasonable to assume that improvement will also be generated through this straightforward approach.