TOXIC COMMENT CLASSIFICATION CHALLENGE

Rowan Curry
STA 142A Final Project
Balasubramanian
Winter Quarter 2021

INTRODUCTION

The Toxic Comment Classification Challenge asks the participants to take a large dataset that consists of Wikipedia comments that have been labelled by human raters for toxic behavior and then use those comments to build a model that predicts the probability of each type of toxicity for each comment. This problem is of interest to me personally because– as someone who enjoys participating in body positive, inclusive online communities– an effective model that marks and monitors users engaging in hateful, bullying behaviors would be game-changing. This project uses the Support Vector Machines Algorithm to identify these toxic comments.

METHODOLOGY

i. DATA DESCRIPTION: The data is provided in the form of four files– a file of training data, a file of test data, and, since the competition is closed, a file that contains labels for the test data. There are six different labels for toxicity ranging from labels that describe potentially rude comments to labels that describe highly offensive, identity-hate comments. These labels are: toxic, severe toxic, obscene, threat, insult, and identity hate. The training data consists of 159, 572 unprocessed Wikipedia comments and their corresponding binary labels for each of the six potential indicators of an offensive comment. The test data consists of 153,165 unprocessed Wikipedia comments. The test labels file consists of the binary labels corresponding only to the comments that were used for evaluation– a value of -1 indicates that the comments were not used for scoring. The data set is in the Creative Commons, and therefore is in the public domain. These comments were randomly selected from from Wikipedia's talk page edits.

ii. SUPPORT VECTOR MACHINES ALGORITHM: A support vector machine takes data points, 'plots' them, determines the hyperplane that best separates the points, and then uses this decision boundary for classification. The Support Vector Machines Algorithm, which I will abbreviate to SVM for the rest of this report, works to maximize the margins of the decision boundary in order to find the best possible method for classification. SVM does this by using estimates of vectors perpendicular to the decision boundary in order to find the decision boundary that has the shortest distance between the closest values of x and the hyperplane. The mathematical equation for the unconstrained version of the SVM algorithm is

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^n}{\arg\min} \sum_{i=1}^{n} (1 - y_i \beta^T x_i)_+ + \lambda ||\beta||_2^2$$

where $\hat{\beta}$ is the estimate of the vector perpendicular to the decision boundary, x_i is the input, y_i is the binary label, λ is the regularization parameter, and $(a)_+ = \max(0,a)$. This report uses the **svm** function from the package **sklearn** to apply this algorithm.

IMPLEMENTATION DETAILS

i. DATA PREPROCESSING: Since the SVM algorithm takes numerical data, I approached this step with the intention of transforming the processed comments into numerical vectors (which will be done

in the next step). I first dropped all null rows from the data set and transformed all of the comments to lowercase text. Then, using the **word_tokenization** function from the **nltk.tokenize** package, I broke each comment up into potentially meaningful words and phrases. Next, I created a for loop to perform word stemming, which is a term that describes the process of reducing each word into a common base or root. In this loop, I also removed numeric characters and stop words, using the **WordNetLemmatizer** object from the **nltk.stem** package, as well as the **pos_tag**, **stopwords**, and **wn** functions available in various **nltk** packages. I also used the **defaultdict** function from the **collections** package. I applied this data preprocessing method to both the training and test csv files.

ii. WORD VECTORIZATION: For this step, I used the Term Frequency–Inverse Document method, which I will abbreviate as TF-IDF in this report. This method summarizes how often a term appears in a document (term frequency) and then scales down words that appear often across documents. For this step I used the object **TfidfVectorizer()** from the **sklearn.feature_extraction.text** package. I fit this object to the processed training data and then used the resulting model to transform both the processed training data and the processed test data.

iii. ENCODING: At this step, I needed to transform the categorical binary labels provided in the data sets into something that the SVM model would understand. I used the **LabelEncoder()** object from the **sklearn.preprocessing** package in order to individually transform each training label.

iv. SVM MODEL: To run the SVM algorithm and make the actual predictions, I used the **svm** function from the **sklearn** package. Since there were six different models, I created six different SVM objects: **SVM_toxic**, **SVM_severe_toxic**, **SVM_obscene**, **SVM_threat**, **SVM_insult**, and **SVM_identity_hate**. I then fit these models on the vectorized training data and the encoded training labels. After this step, I used the fitted model to predict the model's corresponding label by passing the vectorized test data through the predict() aspect of the SVM model. Because it's such a large data set, I specified the kernel as linear when initiating each new SVM object. When all of the predictions were procured, I compiled them into a data frame along with the original, unprocessed test comments. To calculate the accuracy of the SVM model, I first saved the indices of all rows corresponding to labels of -1 in the data frame of test labels, and then removed all of those rows from both the test labels data frame and the results data frame. After step, I was able to isolate each of the six labels from the test labels data frame, perform encoding, and then calculate the SVM accuracy score using the **accuracy_score** function from the **sklearn.metrics** package.

RESULTS AND INTERPRETATION

The tables below display the results of my SVM model in terms of the model's accuracy score for each of the six labels. Table 1 displays the results of building an SVM model with the coefficient C = 1. This model took about 30 minutes to run. As you can see, the accuracy scores were not what I hoped they would be– I decided to try a larger value of C to see if that would improve my model's accuracy. However, I was limited by the capabilities of my computer– the results of building an SVM model with the coefficient C = 100 are displayed in Table 2. This model had a running time of about 12 hours (I had to leave it overnight).

toxic	severe toxic	obscene	threat	insult	identity hate
46.03%	58.13%	50.36%	58.21%	52.26%	57.82%

Table 1: SVM Accuracy Scores for C = 1

toxic	severe toxic	obscene	threat	insult	identity hate
46.47%	57.25%	50.67%	58.06%	52.78%	57.44%

Table 2: SVM Accuracy Scores for C = 100

As we can see, increasing the value of C did not have the desired effect on the accuracy scores. In fact, it seems to have lowered the accuracy scores in the case of the severe toxic label, the threat label, and the identity hate label. Ideally, I would be able to run a cross-validation of coefficients for the SVM model in

order to determine the value of C that would return the highest accuracy scores. However, finishing that cross-validation in a timely manner is beyond the capabilities of the computer I currently have access to.

Two pie charts are provided in order to better visualize the effectiveness of my SVM model. Figure 1 is a pie chart of the test labels, and Figure 2 is a pie chart of the predictions for C = 1 (a pie chart of the predictions for C = 100 was not provided because the results are so similar).

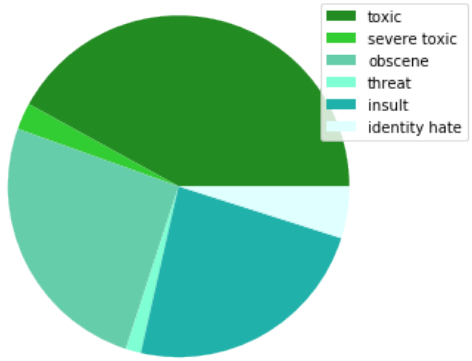

Figure 1: Pie Chart of Test Labels

We can observe that the SVM model over-predicted for the toxic label, and under-predicted for the labels identity hate and severe toxic.

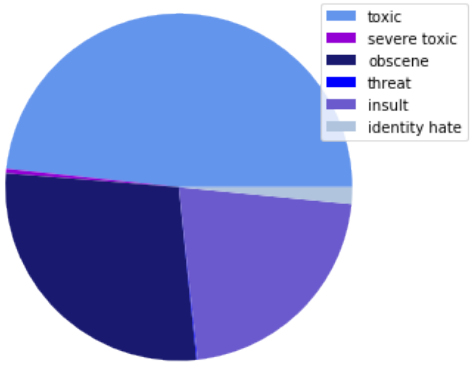

Figure 2: Pie Chart of Predictions for C = 1

Future steps I might take in order to improve the accuracy of my SVM beyond cross validation of the parameter C include adjusting the data preprocessing steps, trying other word vectorization techniques, and examining the other parameters of my SVM model beyond the value of C. I also might try a different classification method, such as the Naive Bayes Classifier Algorithm, or Deep Neural Networks.

ARTICLES USED AS REFERENCE

https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

https://www.machinecurve.com/index.php/2020/11/12/how-to-create-a-multilabel-svm-classifier-with-scikit-learn/