Hunter Bettin & Parker Carlson

Final Project Report

CSCI404

Toxic Comment Classification

Abstract

For our final project, we decided to implement a binary classification algorithm to classify toxicity within a massive database of real online comments. This essay will provide you with the motivation behind our work, a detailed description of our classification system, including our datasets, NLP algorithms utilized, and experiment setup, as well as the results for our system.

1. Introduction

At first we decided to develop a system for toxicity classification purely out of self interest. It seemed funny and entertaining to deal with potentially extremely vulgar online comments. Parker and I being gamers and avid social media users we were well accustomed to online toxicity so we thought we would be well prepared for the job. After delving a little deeper into the importance of work like this we found that classifying online comments can be very useful for a variety of online applications, such as all social media websites and even some online videogames where in-game text chat is enabled. A system like this would be an efficient and automated way to weed out the toxic and/or rude users, in order to issue some sort of suspension or ban on a toxic identified account. We ended up finding a large pre-existing dataset on kaggle.com for online comments. The dataset contained about 160,000 classified

comments for training data, and about 150,000 comments for testing. After finding the dataset online, we were set on developing a system to help classify these comments and we had a good idea on how to do it: Naive Bayes Classifiers.

2. System Overview

2.1) Data

We got our dataset kaggle.com/c/jigsaw-toxic-comment-classification-challenge/
The website provided us with both a train and test dataset with about 160,000 and 153,000 comments respectively. The pre-classified data supplied is with classification of toxic, severe_toxic, obscene, thread, insult and identity_hate. With a value of 1 meaning they do fit the given category or 0, if they do not. Due to this data being used for competition, about 100,000 comments within the test dataset were not classified, instead those comments were replaced by a series of -1s for their classification values. The comments were mostly english, but did have some foreign languages scattered throughout the dataset. The dataset was also not cleaned beforehand. The only help given was that most of the data was pre-classified for us.

2.2) Feature description

Our system involves five major components, those being: csv parsing, word cleaning, binary classification conversion, feature vector creation and probability calculations.
The program starts by parsing the csv files to store the row data into a massive dictionary containing the column title as the keys and a list of all row entries for that specific column as their value. We take this this dictionary and pass it through what we call a binary converter,

which basically just looks to see if any of the classification values (toxic, severe_toxic, obscene, thread, insult and identity_hate) are true, then we take that comment and generally classify it as “toxic” rather than a specific form of toxicity. After binary conversion we take our comment list and throw it into our word cleaning function which performs tokenization, punctuation removal, stop words removal, and stemming. We then separate the cleaned comments into lists of toxic and nontoxic comments and then turn those lists into count dictionaries with the token as key and the count of the word as value. Finally we only keep the top 50,000 words within each dict and delete the others due to those being uncommon and only appearing once in about 160,000 comments. Now we parse our testing data, repeat the binary conversion, and word cleaning, but rather than getting a count of how many times each token appeared in the entire csv, we get a count of each word in relation to the individual comment. We create feature vectors for toxic and nontoxic comments with this information and then calculate the probabilities of each comment vector being either toxic or nontoxic. Depending on which probability is higher using either the training toxic or nontoxic dictionary, the given vector will be either classified as Toxic or Nontoxic. Finally, we run all of our predictions against the testing data to calculate our precision, recall and F1-scores.

2.3) NLP Algorithm

For our system we used Naive bayes binary classifiers with +1 (laplace) smoothing. Within machine learning the algorithm basically consists of formatting all of your test data into feature vectors. Where a feature vector is just a group of occurrence counts of a feature or set of tokens in our case. With accurately labeled training data you can then use these vectors and run them against the frequency of tokens in the training data to calculate the probability of that vector being a specific classifier. You then compare the two probabilities against each other and

whichever is more likely gets the more likely classifier. We used simple laplace smoothing within our probability calculations to account for tokens that appeared in the test data/feature vectors but never appeared in the training data.

2.4) Experimental Setup

For our experimental setup we tested the accuracy of our program using the F1-score method. Basically tallying up the True-positive, false-positive, true-negative and false-negatives of our results to calculate precision, recall and f-score. We tested our results with 4 different parsing parameters.

1) 10,000 training comments
2) 50,000 training comments
3) 100,000 training comments
4) All 160,000 training comments

3) Results

These were the results from our experiments:

Experiment	**1**	**2**	**3**	**4**
Precision	**49.3025**	**51.2182**	**54.4232**	**56.3025**
Recall	**67.7160**	**74.7172**	**79.4721**	**82.7160**
F1-Score	**57.0651**	**60.7753**	**64.6046**	**67.0**

As we added more training comments our results got progressively better, but it looks as though it's about to hit a curve around 160,000 training comments. But we were happy to see that there was room for improvement within our training, although the accuracy wasn't reliably high, the

recall stayed fairly high meaning that it could identify most of the non-toxic comments within a test database. This is significant as only around 10% of our data was toxic, meaning our machine could properly identify around 82% of the nontoxic data. Where it struggled however was with toxic data, and we believe this was due to having such similar wording between toxic and non-toxic data, it's not as simple as identifying harsh words you also need contextual wording around it to identify.

4) Conclusion

In conclusion, our machine was not as accurate as we were hoping, we were happy to see it could identify a large portion of the data but it could not produce an overall accuracy to satisfy constraints. We believe this had to do with the lack of contextualization within our data, and how we couldn't properly distinguish between someone using harsh wording to get a point across, and someone insulting another individual. Going forward we want to use a separate model alongside our naive bayes, we think that an SVM could produce much higher precision when compared alongside our naive bayes probabilities. Alongside that our calculations took a long time to come out, and so we would like to refactor our code and storage methods used for our data. Another option could be a neural network, but this could prove to be too expensive for how quick this process would need to be.

References

- https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview
- https://www.nltk.org/
- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
- https://stackoverflow.com/