Ethan Frias: 77331598

Wilson Su: 11481462

Brian Avalos Herrera: 37738731

Professor Alexander Ihler

Machine Learning & Data Mining

14 December 2023

# Toxic Comment Classification

For our project we chose to experiment with the toxic comment dataset. As text classification and analysis were not covered much in this course, we needed to research different ways to experiment and train language models. With the current relevance of websites needing to automatically moderate user content and the amusing nature of testing a model with profanity, we felt choosing this dataset was an obvious choice. Each of our members decided to focus on different models. Ethan created the template code and explored the vectorizers, Wilson explored the parameters for LogisticRegression(), and Brian explored the parameters for SGDClassifier(). Our main focus was on linear classifiers due to their simplicity and ease of training regardless of data size and dimensionality. Using this, we were able to learn a lot about our data set in the process by vectorizing the text data and analyzing accuracy via multiple kinds of metrics. After several parameters testing in each model, we reached our conclusions as outlined in this paper.

Our first objective was to represent the text data in a way that allows for our models to process the input data. In class we covered the idea of a bag of words, a model that counts the occurrence of words in each text document and transforms them into high dimensional data points that can be used with many common models. We were able to perform this operation using sklearn's CountVectorizer() which is fitted on the training data to build a vocabulary of words. However, the training data provided had around 65 megabytes of data, which would result in a 190,000 word vocabulary. Luckily CountVectorizer() has a parameter (max_features) to adjust how many words we would like in our vocabulary. As shown in Figure 1, we found that only 1,000 words were needed to obtain performance that very closely matched that of using all 190,000 words. Since words that can help detect toxicity are not very common, we were initially worried that doing this might make our model significantly less accurate as so many words are stripped away. Luckily that was not the case and we believe it might be due to our training data, which could be increasing the frequency of toxic words appearing. Though 1,000 words seemed to be the practical minimum limit as there were very notable losses in accuracy for the testing data if the limit were any lower.

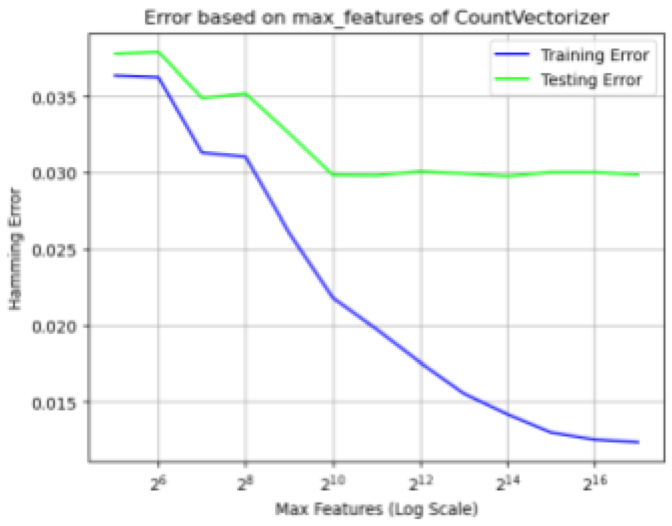

Figure 1

However,  CountVectorizer() did present some issues. By only counting the quantity of words in text data, “long” data with many words can result in some data points having scaled results that can negatively affect classification accuracy. For example, if one piece of text data was three times as long as another, we would expect the vectorization of it to be scaled three times as much. Due to this, it was harder to fit models, such as a logistic regression model that needs to converge to a final state. Many models would end up hitting the maximum iteration limit before termination, which was not ideal for accuracy and training time. In addition, Figure 1 indicates that training notably increased if more than 1000 words were used in the vocabulary, which implied that less common words still had some statistical importance despite their rarity. Thus we concluded it would be more preferable to vectorize text data based on frequency of each word so they would still evaluate to similar vectorized data points while avoiding the convergence problem and emphasize rarer words to help with classification

A method called tf-idf solves both issues. This method transforms data based on the term’s frequency in a single datapoint (term frequency) and multiplies it with its inverse frequency in all data points (inverse document frequency). Like CountVectorizer(), sklearn has a dedicated object called TfidfVectorizer(), which also fits the data by building a vocabulary. Instead of counting words however, TfidVectorizer uses the tf-idf operation, blending its frequency in individual documents and across all documents. This effectively solved both of our problems with CountVectorizer() and our models converged much quicker with smaller gains in training performance using more than 1000 words in a vocabulary, as seen in Figure 2. Though using less than 1000 words still had a noticeable decrease in performance, using more words yielded a higher performance, so the group decided on 2,500 words in the vocabulary. Further experimentation with frequency parameters indicated that rarer words had a larger weight when classifying data compared to common words.

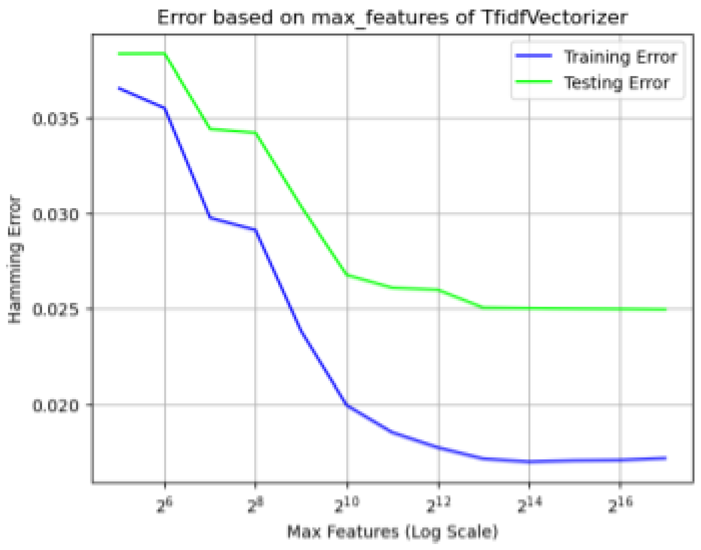

Figure 2

Lastly, we also explored the importance of word capitalization and the consideration of accented characters for accuracy. Capitalization can be used online as an indicator of tone. For instance, “go away” and “GO AWAY” have significantly different connotations. Characters with accent marks such as “á” may be used in profanity in other languages or to bypass moderation. However, both of these considerations in our model just increased the size of the vocabulary without any performance benefit, depicted below in Figure 3.

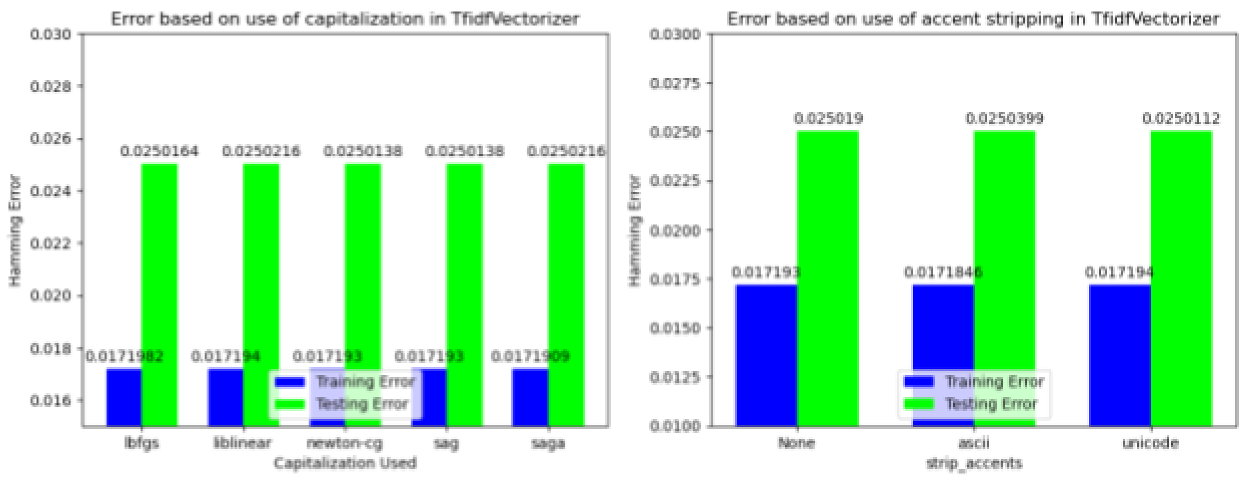

Figure 3

This may be due to the data set provided as it was gathered from an English speaking community on Wikipedia, which may not use capitalization or accented characters in the same way as other online spaces such as social media or video game chat rooms.

Another complication we needed to tackle was that the data was a multilabel set, with each datapoint having 6 different labels that may or may not be applicable. Using sklearn's MultiOutputClassifier(), we were able to address this issue. However, since MultiOutputClassifier() returns an array of predictions for each label, a different metric was needed as a zero-one loss function would return an incorrect prediction if two arrays did not perfectly match. Hamming loss was used instead which compared each individual label and returned the proportion that did not match. An interesting observation was that a basic model without optimization had a hamming loss of around 3-4%, but only 10% were labeled with any form of toxicity. Initially we assumed that the Toxic Comment dataset would have a larger proportion of toxic comments, but with a large proportion of online interactions benign in nature, it is logical only a small amount of data would be labeled toxic.

Another performance evaluation metric used were confusion matrices. They were particularly useful to evaluate the model's performance for each individual label. Confusion matrices helped identify false positive and false negative predictions. It is important for automatic content moderation to limit the amount of false positive predictions, else they significantly interfere with benign user activity. False negatives meanwhile are generally preferred as content can be manually moderated without inconveniencing the user base.
This can be achieved by using weights to prioritize decreasing the false positive rate.
The confusion matrices also showed the ease in which the models fit certain labels.
For example, rare labels such as "threat" or"severe_toxic", which only about 1% of data was labeled as such, were difficult to fit accurately, likely due to little data. The general "toxic" label was common but tended to have the most variation in errors, which was likely due to how slight variation in human classification and word order, which is not preserved during vectorization, might make toxicity a bit vague to the model.

Finally, by using different parameters, we observed that overfitting and underfitting had unique characteristics with manual performance. Manual testing with text classification showed the general behavior of models that fit poorly. Underfit models were conservative with their estimates, rarely labeling data as toxic, causing many prediction errors. Similarly, overfit models were liberal with classification and labeled many non-toxic data as toxic. Initial models showed more underfitted behavior as they were not yet optimized. However, observing overfitting was more difficult. Using a StandardScaler() with a complete vocabulary in the vectorizer gave common words similar importance as less common ones. As shown in Figure 4, this practically eliminated the training error while the testing error doubled to 5.1%, a symptom of overfitting. When evaluating the manual performance we observed  the overconfidence traits specified earlier. We attempted to use bagging to reduce this overfitting however this proved to be both expensive and ineffective in preventing overfitting.. Using a SGDClassifier() using a perceptron solver also resulted in overfitting.

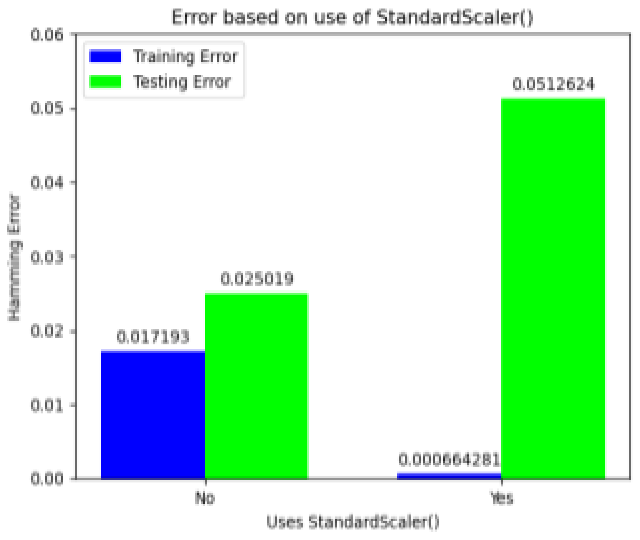

Figure 4

After further testing and experimentation, we found our most desirable performance was using a TfidfVectorizer() with 2500 max_features, and a LogisticRegression() model with class_weight={0:3, 1:1} and 200 max iterations, achieving a hamming loss of about 2.53%. It is worth nothing that there were still many methods that have not yet been explored that could further improve the model's performance. For example, the simplification of implementing a MultiOutputClassifer() to handle all 6 unique labels is not ideal as every label uses the same LogisticRegression() model and parameters. Instead,  manually controlling each model and their parameters for each label could achieve more accurate results. Neural networks were also not explored due to the huge dimensionality and quantity of the data, as training and testing would be much more time consuming, which was difficult to perform due to the time-limited nature of this project. For instance, training the LogisticRegression() models using the maximum vocabulary took around a minute while attempting to train a MLPClassifier() with the same vocabulary took multiple hours without successful completion. Lastly, the dataset provided was sampled from Wikipedia in 2018. This limits the scope significantly and was not able to effectively extrapolate many kinds of toxicity present in modern internet activity, which caused some manual performance tests to not be labeled correctly.

In conclusion, if we were to revisit this project we would likely explore these aspects further. Despite this we still reached a deeper understanding of this dataset and text-classification methods such as vectorizing text data and evaluating performance that could be used in future language processing projects.