DSC 478 - Winter 2023
Project Type: App Dev
Team Members: Jeff Bocek, Xuyang Ji, Anna-Lisa Vu

A Toxic Comment Identifier Application
Final Report

Executive Summary

The goal of this project was to create a Python application that identifies whether a comment is toxic. This app can be used as a third-party library for social media sites or public sites where users are allowed to leave comments. When the application starts up, it will train multiple models against an existing data set. Once the models are trained, the user will be prompted to enter a comment. The application will classify the comments using the following categories below:

- Non-Toxic
- Toxic
- Severe Toxic

A category to the provided comment will be the output from the program. Any program would be able to apply logic based on the results of this application.

We used a data set from a Kaggle competition which can be accessed here: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data?select=test.csv.zip.

We performed full exploratory analysis on the existing dataset. Based on what we discovered, we pre-processed and cleaned the data. Next, we analyzed the number of classes we needed to work with and made sure that both the train and test data were balanced (or unbalanced) appropriately. The bulk of our time was focused on training and tuning various feature transformation techniques, feature reduction methods, and models for our application to use. Lastly, we developed a simple command-line application to apply the techniques and models we tuned to categorize a comment that is provided by a user.

Final Conclusions and Outcome

We found that our top 3 models were the following in decreasing order of accuracy

- Stochastic Gradient Descent Logistic Regression using the Tfidf matrix and normalized truncated Singular Value Decomposition. → Accuracy: 72.4%
- Rocchio (Nearest Centroid) using the Tfidf matrix. → Accuracy: 69.8%
- Compliment Naive Bayes using the Tfidf matrix. → Accuracy 64.0%

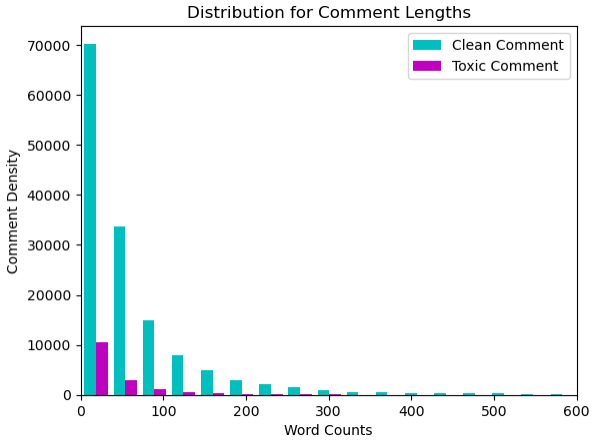
We used these top 3 models and put them in an ensemble model which achieved an accuracy score of 72%. After, we developed a command-line application which classifies user comments using the ensemble model.

Exploratory Analysis

Unfortunately, having conversation about topics one cares about can be challenging under some scenarios, such as the threat of online abuse and harassment. An insecure online environment not only causes many individuals to refrain from expressing themselves and seeking diverse opinions; but also has led to various platforms struggling to effectively facilitate discussions, resulting in many communities limiting or forcing shutting down user comment sections. With a goal to foster healthier online communities by addressing the issue, our team worked on developing a Python application which focuses on comment toxicity detection. The app can be used as a third-party library or extension for social media sites or public sites where users are allowed to leave comments. With various clustering and predictive models stored in the backend, the app allows users to detect the toxicity level of specific queries, and revise them for maintaining a more respectful online community.

The dataset comprises 159,571 comments from Wikipedia, with each comment consisting of a string data input feature and six labels that categorize the comment as toxic, severe_toxic, obscene, threat, insult, or identity_hate.

Comment Length



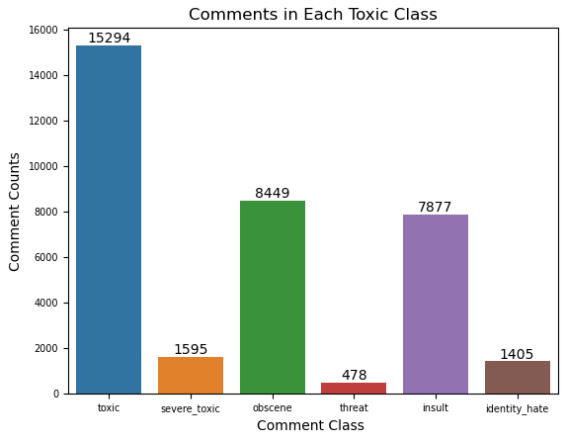
Looking at the histoplot for comment length differences between clean and toxic comments, clean comments tend to be approximately one-fourth longer than toxic comments on average. Upon examining random samples, it becomes apparent that many clean comments consist of long and well-crafted responses. On the other hand, toxic comments generally have less word counts by looking at the distribution for toxic comment length.

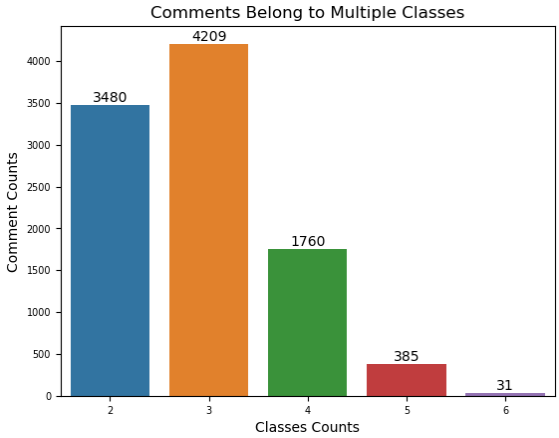
Class Imbalance

The following figure shows how these labels are distributed throughout the dataset, including multi labelled data. Although there is no missing value in the training dataset; however, based on the fact that the mean values are extremely low, it can be inferred that the majority of the comments are likely to be clean/non-toxic comments. In

other words, the comments are not evenly distributed across classes, and class imbalance is present. Upon investigation, the clean comment ratio in the training set is 89.8%, while there are 58.2% of clean comments in the test set.

The breakdown demonstrates that while most comments with other labels are also toxic, not all of them are. Only "severe_toxic" is clearly a subcategory of "toxic," which is reasonable to rule out labeling errors. The observation indicates that "toxic" is not an overseeing label, but rather a subcategory in the bigger context with considerable overlap with other labels. Regarding the issue of multi-labelling, it would most likely pose difficulty to train a classifier on specific labels in the raw dataset due to overlapping. The ambiguity surrounding the label assignments and the absence of clear explanations is the reason why we opted to use aggregate labels for general toxicity levels, called "non_toxic", "mild_toxicity", "toxic", and "severe-toxic" as the targets going forwards.

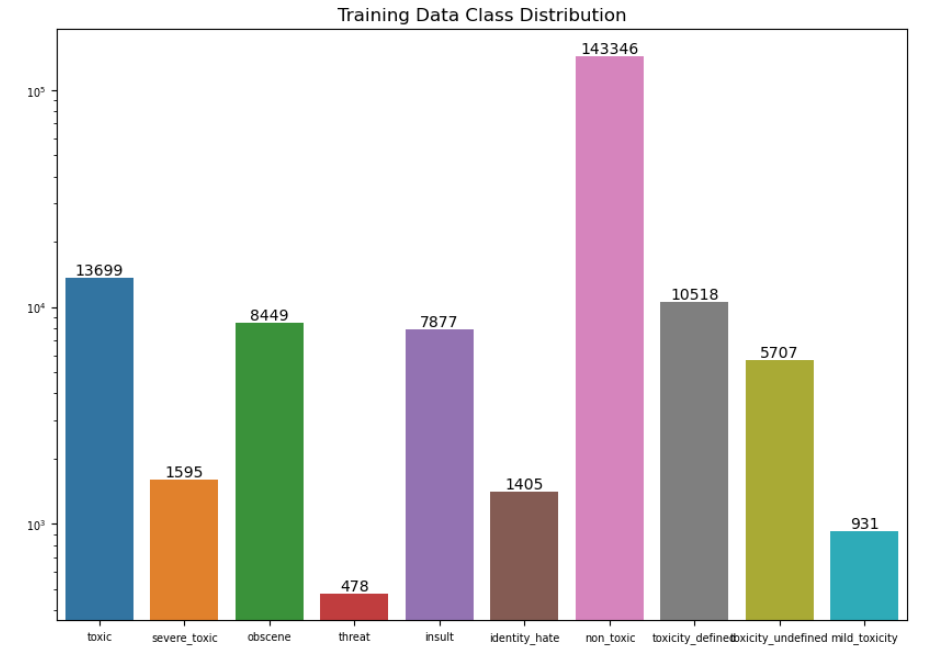


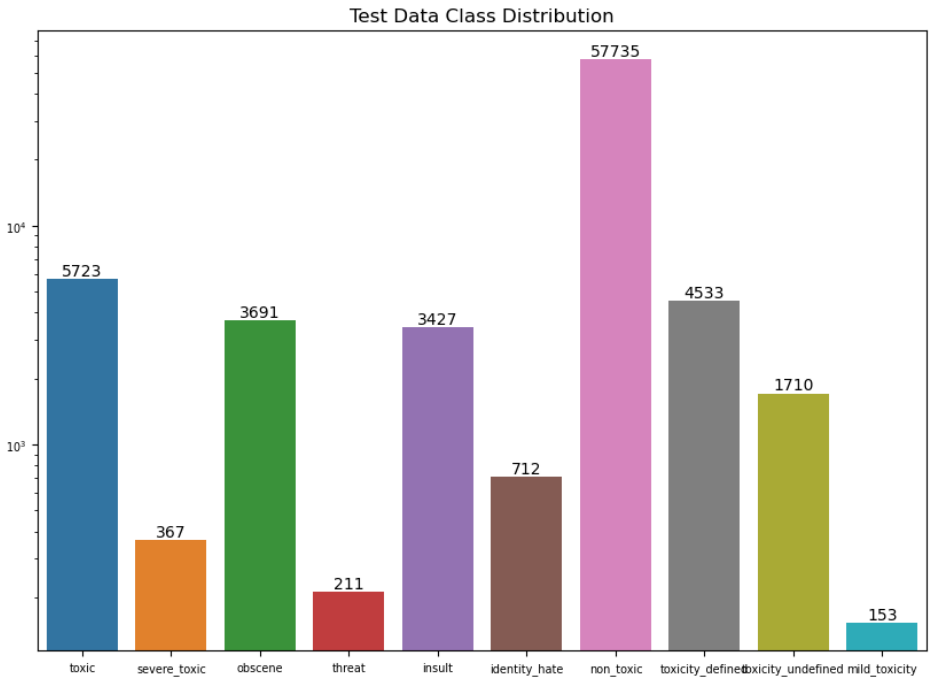


Pre-Processing of Data

Class Resampling

As the first step to ensuring data quality, the reconfigure_categories function transforms the multi-labeled dataset into 4 separate independent classes. While the semantic meaning of toxic and severe-toxic comments shows some level of graduation; we firstly fill the all toxic column of the severe-toxic comments with 1, and extract all severe-toxic comments from the toxic class. The newly created non-toxic column is for comments with a target_col value of 0. Since labels are nested under each other in some cases, comments with toxicity-type defined are identified and stored in a new column called toxicity-defined. Finally, the new mild_toxicity column is created for comments with toxicity defined but not labeled as toxic/severe toxic. The key assumption is that there is no significant correlation among the following labels. *(level 1: Non-toxic comment; level 2: Mild_toxicity comment; level 3: Toxic comment; level 4: severe_toxic comment)*





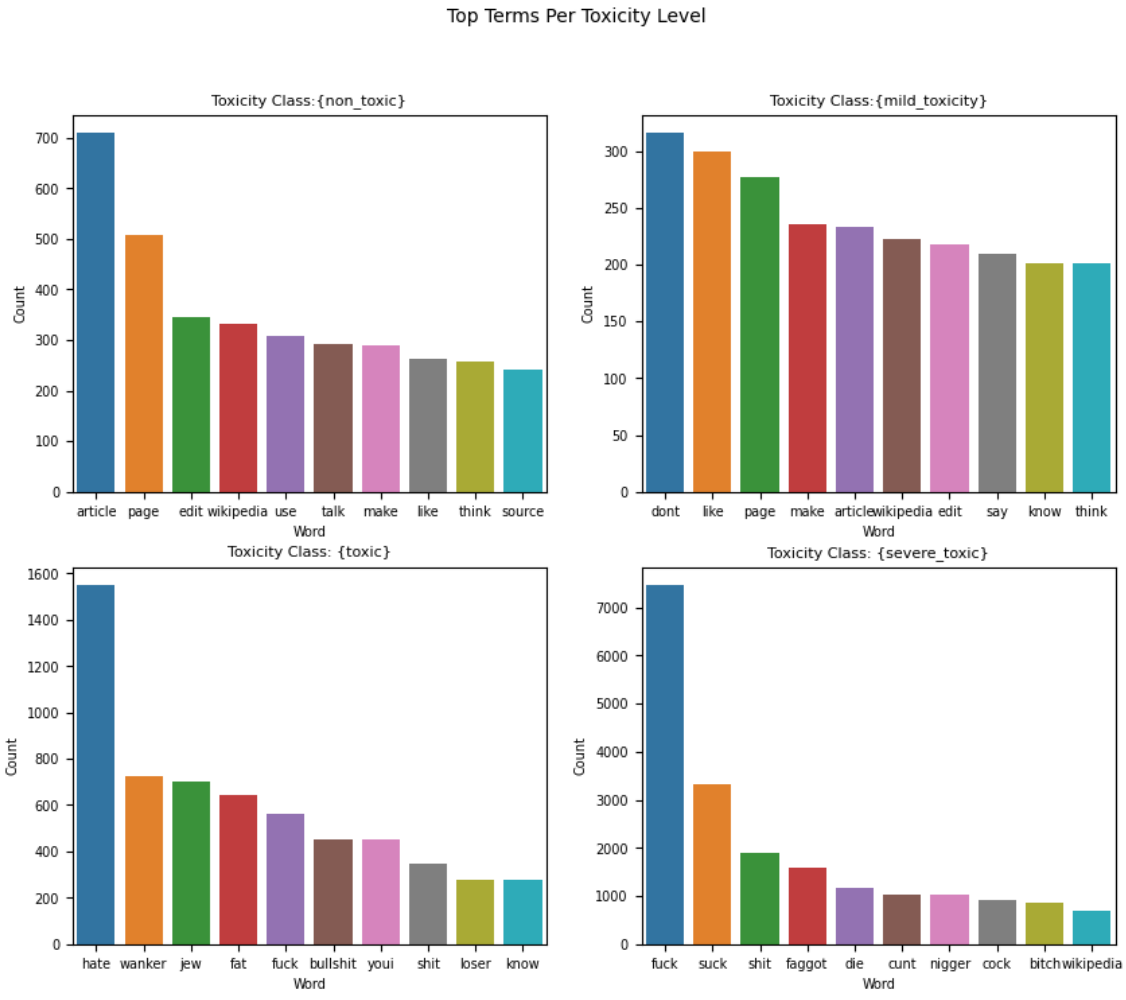
Text Preprocessing

The diversity and vastness of social media comments make it difficult to comprehend and capture the underlying trends and characteristics for comments with different toxicity levels. Keeping all words makes the dimensionality of each text extremely high, which makes classification more challenging. However, properly preprocessing the data,

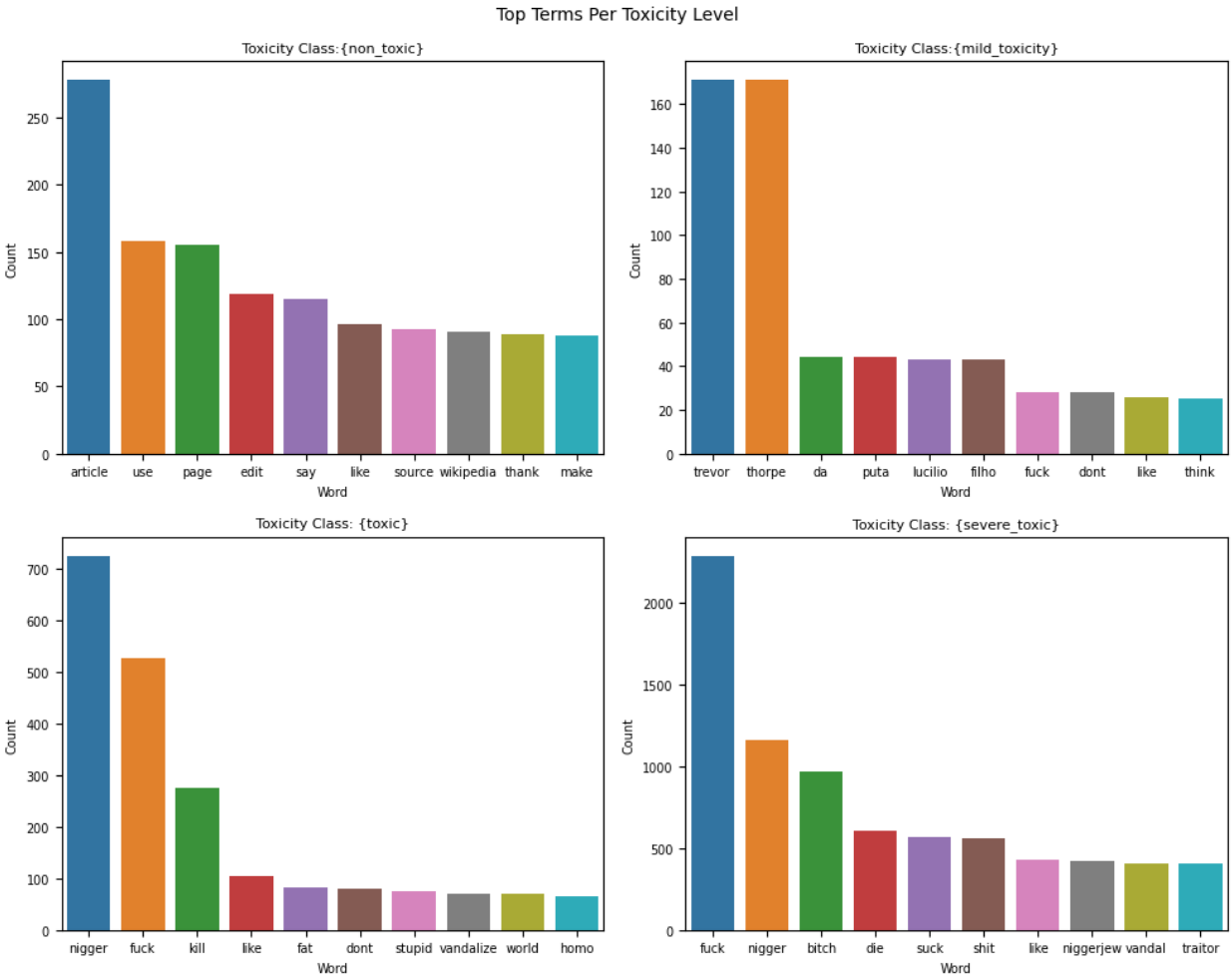
by reducing noise in the text, may improve classifier performance and speed up the classification process, thereby facilitating real-time sentiment analysis. Hence,in order to conduct data mining on online opinion data, the text preprocessing steps involve several stages, including removal of punctuation, lowering the text, removal of white spaces, stemming, removing stop words, handling negation, and finally tokenization and lemmatization.

Data Quality Assessment

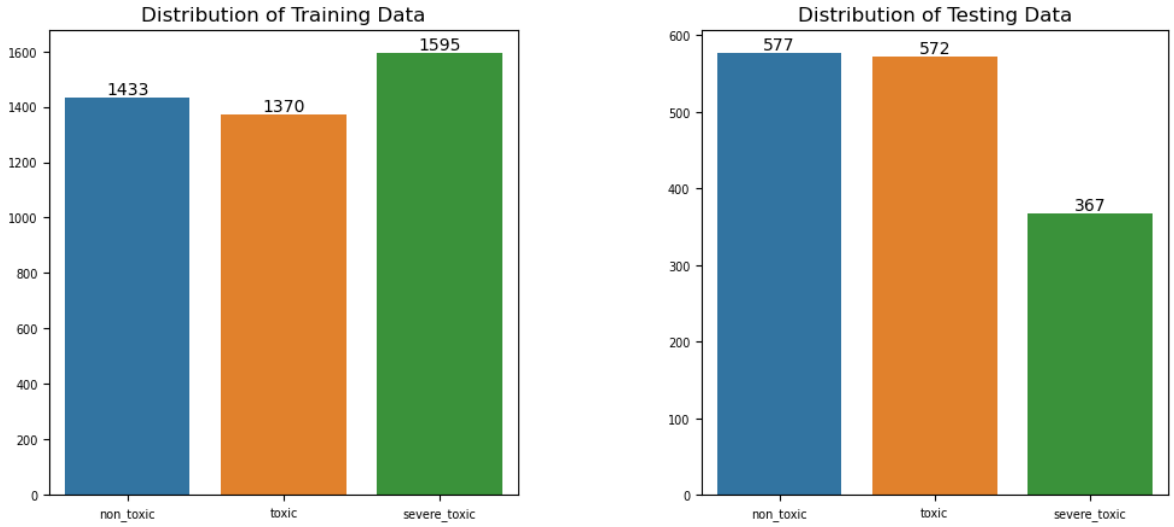
In the final step for data preprocessing, we examined data quality of the 4 classes based on the Bag of Word model, where each row represents a specific text in corpus and each column represents a word in vocabulary. Based on the following plot on top terms per toxicity level, there is no significant difference between non_toxic and mild_toxicity in both training and test set. Hence, regarding the small distribution in mild_toxicity class and the low level of uniqueness, we decided to drop mild_toxicity for model quality.



*(*Top Terms in Training Set)*



*(*Top Terms in Test Set)*



Feature Transformation

We decided to use 3 methods of feature transformations - Term-Frequency, TF*IDF and Doc2Vec. Full explanations on each method are provided below.

Term Frequency

Simply put, the weights in the matrix represent the frequency of the term in a specific comment. The underlying concept is the higher the term frequency for a specific term in a comment, the more important it is for that comment. We use scikit-learn's CountVectorizer(). Tuning occurred by adjusting the 'max_df' and 'min_df' which when building the vocabulary ignore terms that have a document frequency higher/lower respectively than the given threshold. We used a proportion of documents for the max document frequency and individual values for the min document frequency during parameter tuning.

TF*IDF

Term Frequency - Inverse Document Frequency assesses how important a term is within a comment relative to our collection of comments. It does this by vectorizing and scoring a term by multiplying the term's Term Frequency (TF; number of times the term appears in the comment over the total number of terms in the comment) by the Inverse Document Frequency (IDF; the log of; the total number of comments over the total number of comments which contain the specific term plus one). By using this formula we don't place added importance on the super common words which overshadow less common words which can show more meaning to the comments. We used scikit-learn's TfidfVectorizer() to transform our preprocessed collection of comments into a TF-IDF matrix. In aberration to the basic formula referenced above TfidfVectorizer() adds a "+1" to the numerator and denominator of the IDF score to prevent zero divisions when the term is not present to better represent the IDF scores for sparse matrices. We tuned this transformation by trying two methods of normalization; 'l1' and 'l2'. 'l1' normalized by using the sum of the absolute values of the vector elements is 1 while 'l2' is the Euclidean norm and the sum of squares of vector elements is 1. For 'l2' the cosine similarity between two vectors is their dot product. As was the case with Term Frequency tuning, tuned by adjusting the 'max_df' and 'min_df' which when building the vocabulary ignore terms that have a document frequency higher/lower respectively than the given threshold.

Doc2Vec

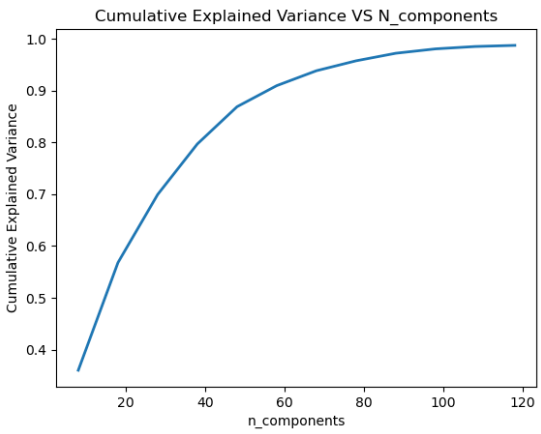
Doc2Vec is a natural language processing technique to transform a set of documents into a list of vectors. It is closely related to Word2Vec and is considered a more generalized form of it. In Word2Vec, the algorithm looks at the context of each word and surrounding words. It assigns a numerical value to the word which represents the meaning of the word given its context. For example, the chances of "Paris" and "France" being in the same sentence

are higher than “Paris” and “power”. Word2Vec takes this into consideration when creating the numerical value for each word. This example was taken from this article: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e.

Doc2Vec has generalized this technique. Instead of having a numeric representation for the words, you can achieve a numeric representation for a document as a whole. We thought it would be a good exercise to use this data transformation technique in our project. Each comment can be like a document. To create the Doc2Vec vectors, we used the doc2Vec library package from the gensim.models library. We ran this against the train and test data and fed that into our models.

Feature Reduction

We used Latent Semantic Analysis(LSA) which uses Singular Value Decomposition(SVD) in an effort to improve our models for classification. Due to our comments having lots of terms we have a very high number of features. This method reduces the complexity of our models while trying to minimize loss of information. We used scikit-learn’s TruncatedSVD() to transform our data matrix. We choose this method as it works well with sparse and non-square matrices. Since we are using SVD on term count/tf-idf matrices this method is referred to as LSA. This method helps to reduce the effects when words have multiple meanings. By running the LSA function, the suggested range is between 105 to 115 components. We then test accuracy and F1_scores for each components sets in that range in our base model, to ensure data quality.

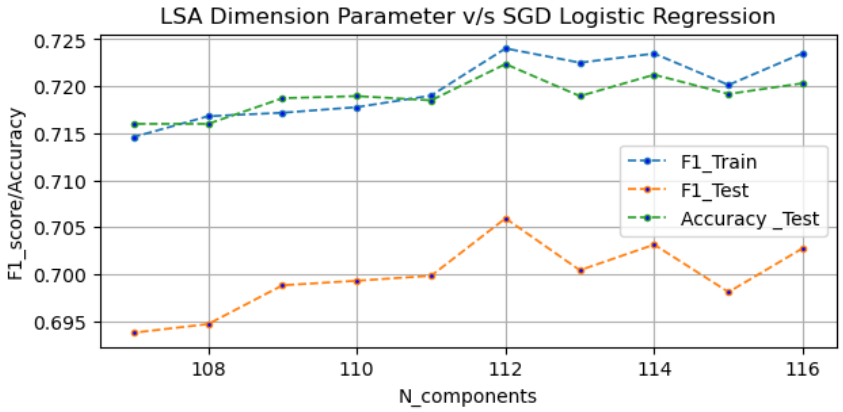


Models

Gradient Descent/Logistic Regression

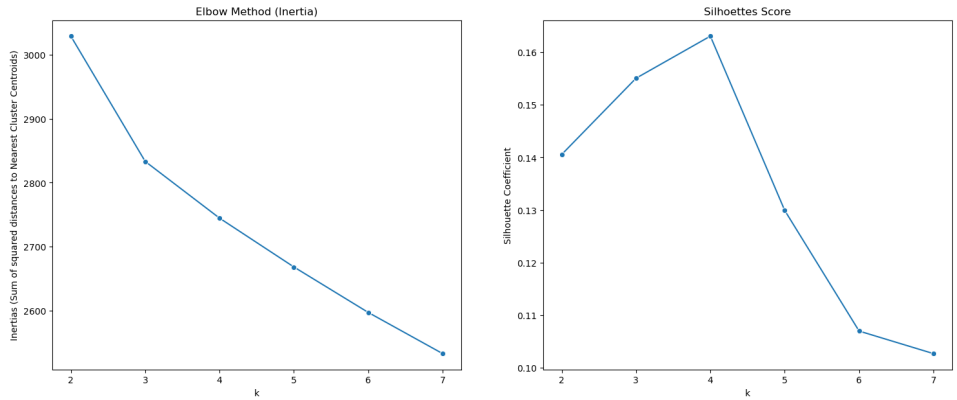
The baseline model is a Logistic Regression model fit to tf-idf vectorized comment text with using only words for tokens, with the target value being toxicity_level. In logistic regression, a cost function is defined to quantify the deviation between the predicted output and the actual target output. The aim is to minimize this cost function by adjusting the values of the model's weights (represented by w) and bias (represented by b) using gradient descent, which further enables us to use the updated weights and bias to make accurate predictions on unseen data points. After obtaining the model's output, the sigmoid function is commonly used to convert the continuous output into a probabilistic output, which ranges between 0 and 1, which helps in the classification task.

By performing a grid search using GridSearchCV between the two penalty parameters ('l2', 'elasticnet') and different values of l1_ratio, the best component to keep from the tsvd is n_componnet=112 with explained varianceof 98.53%, ,the best parameters are: {'l1_ratio': 0.8, 'loss': 'log', 'n_jobs': -1, 'penalty': 'elasticnet'}. Using the best mixing ratio with Stochastic Gradient Descent Regression, the accuracy on the test set is 72.37%. The training accuracy was 76%.

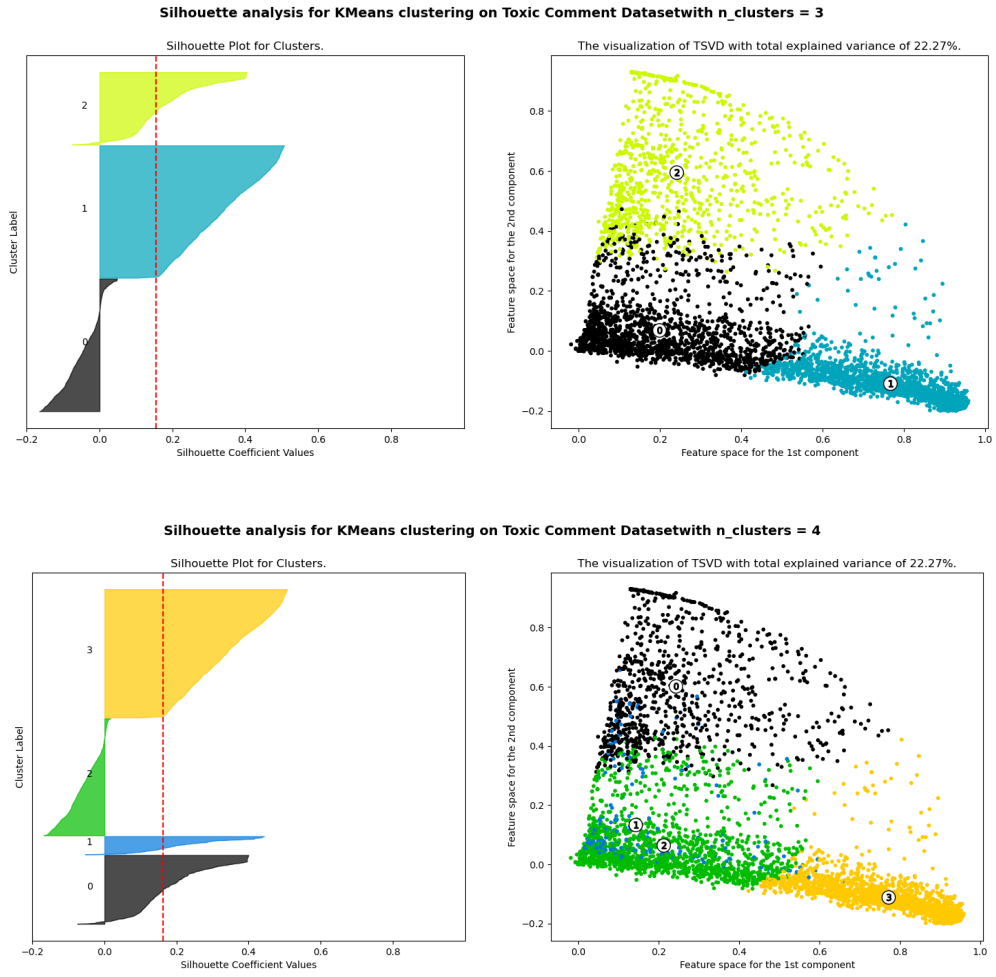


Unsupervised Model to ensure data quality - K-Means Clustering

The K-Means Clustering assigns clusters based on the distance to the cluster centroids. To streamline the parameter tuning process, we defined a utility function, silhouette_analysis_with_tsvd, that provides a fitted k-means model and additional metrics. The following SSE and the Silhouette score method give an idea on what an appropriate number of clusters would be based on the sum of squared distance between data points and their assigned clusters' centroids. The plot illustrates the line that helps in choosing the elbow point. It can be observed that the sum of squared errors (inertia) does not improve much after point K = 3, while the Silhouette score suggests a cluster of 4 . Hence, ***the number of clusters are tested using the K-Means algorithm to determine the most optimal number of cluster in the dataset***.



Based on the following output, the clusters formed are quite distinguishable and balanced without significant overlapping with k=3, while significant overlapping is shown in cluster1 and cluster2 with k=4. Hence, the most optimal value of k is chosen as 3 clusters.




```
Cluster 0: suck go bitch youre shit wikipedia article get asshole stupid
Cluster 1: article page use like people source one tag copyright make
Cluster 2: fuck go shit bitch youre asshole cunt faggot article wikipedia
```

Nearest Centroid

In the Nearest Centroid classifier, a mean centroid is calculated from the training set. Each centroid is associated with a class. The model classifies data by using the label of the closest centroid to the test instance.

We chose this model because it is a simple model to run and most often used as a baseline model. In this project, we used the Nearest Centroid implementation from the sklearn library. We tuned our model by evaluating the results using the euclidean and cosine distance metric along with various parameters data transformation. An attempt was made to also tune using the shrink_threshold parameter; however, this didn't work well with our data set since the model complained that our data set was too sparse.

Best Tune Model: We used scikit-learn's GridSearchCV() to search which of these parameters would produce the best model using stratified five-fold cross validation which preserves the percentage of samples for each class. Our best model used the Tfidf vectors generated from the TfidfVectorizer library (sklearn) using a max of 0.6 and a min of 1 , no dimensionality reduction (vs, SVD), and the cosine distance. The model using Doc2Vec instead of Tfidf had the same parameters minus the parameters needed to generate the Tfidf matrix from the TfidfVectorizer library. The training accuracy was 78% and the testing was 70%.

Naive Bayes

The Naive Bayes model uses the probabilities of the features and vectors to predict the class of our comments. The NB assumes independence between the features and a normal distribution. NB was chosen as it works well with text classification with a multiclass problem. We included the smoothing parameter in our tuning to account for small samples which use Laplace smoothing to correct for zero frequency issues. Truncated SVD (feature reduction) and using Doc2Vec were unable to be used as NB is incompatible with passing negative values. If we had more time we could have explored the correlations between the terms to try and reduce our features so we reduce the inflating of importance. While our dataset is not severely unbalanced we used Complement Naive Bayes (CNB) as it performs better on text classification than the standard Multinomial Naive Bayes. CNB uses the complement of each class to compute the model's weights. It applies a second normalization to address the tendency for longer documents to dominate parameter estimates in MNB.

Best Tune Model: We tried using both Term Frequency and Tfidf with CNB. Both models used a max document frequency ratio of 0.6 while the TF model used a minimum document frequency of 1 and Tfidf of 3. The best Tfidf model used the "l1" Euclidean norm. The training accuracy was 82% while the testing accuracy was 61% indicating that our model is potentially overfitting the data or we may not have enough training data. For example a word in the testing data was not in the training dataset.

KNN

The K Nearest Neighbors model depends on a distance or similarity function to compare how far apart instances (data points) are away from each other. In classification, for a given number of points, the majority label of these "nearest neighbors" is assigned to our point in question. To tune our KNN model we examined three parameters. The

first was the number of neighbors (*k*). A larger *k* should reduce the effects of noise but will make the classification boundary fuzzier. A smaller *k* could overfit the model. The second parameter used was the type of distance function used to measure between the points. We tried either the cosine distance which measures the angle between the two vectors of the points and the minkowski distance with $p = 2$ which is the euclidean distance function (pythagorean distance). Typically when dealing with sparse data (text data) the cosine distance helps the model to perform better. The last parameter tuned was the weights function used in the prediction of points using the model. We examined uniform weights (all the neighbors are weighed the same) versus distance weights which the closer neighbors will have a stronger impact than the ones further away.

Best tuned model: We used scikit-learn's GridSearchCV() to search which of these parameters would produce the best model using stratified five-fold cross validation which preserves the percentage of samples for each class. Our best model used the Term Frequency vectors (not Tfidf) with a , no dimensionality reduction (vs, SVD), the cosine distance, ten nearest neighbors, and the neighbor weights were uniform. The model using Doc2Vec instead of Term Frequency had the same parameters. The training accuracy was 73% and the testing accuracy was 61%.

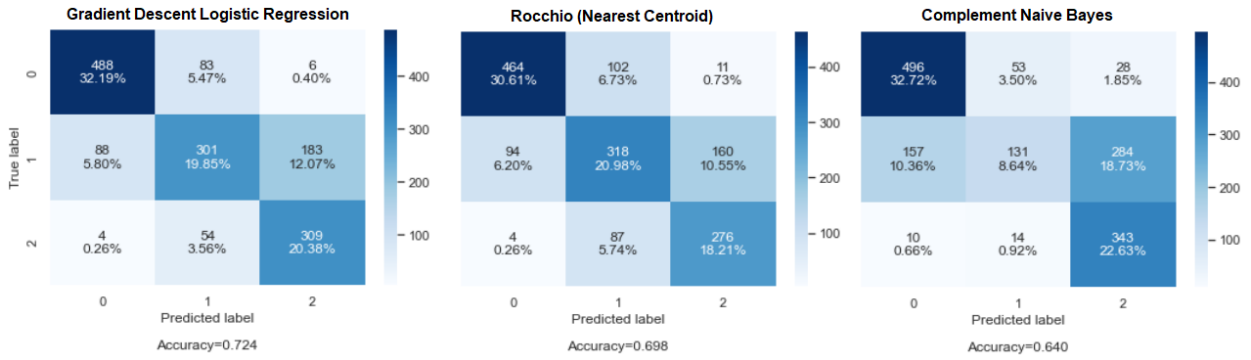
Evaluation of Models

For those models where appropriate we evaluated the confusion matrix, accuracy, and ROC AUC score for each model. As part of the confusion matrix evaluation we looked at the precision, recall, and F1 scores.

In comparing our different tuned models we found the following three the best:

1) Stochastic Gradient Descent Logistic Regression using the Tfidf matrix and normalized truncated Singular Value Decomposition.
2) Rocchio (Nearest Centroid) using the Tfidf matrix.
3) Complement Naive Bayes using the Tfidf matrix.

We made this determination primarily on the F1 score. The F1 score represents precision and recall in one metric. We are interested in classifying our comments as non-toxic, toxic, or severely toxic. Most importantly we want to minimize our False Negative rate. If a comment is toxic, we do not want to label it as non-toxic and thus not be flagged or removed by the application. Hence, we paid special attention to the recall score which focuses on the false negatives. Looking at the confusion matrices below, we see the most errors in trying to classify the toxic comments (label "1"). If we had to choose the best model, we would use Gradient Descent Logistic Regression because it has the lowest instances of classifying toxic and severely toxic comments as non toxic (5.8% and 0.26% respectively).



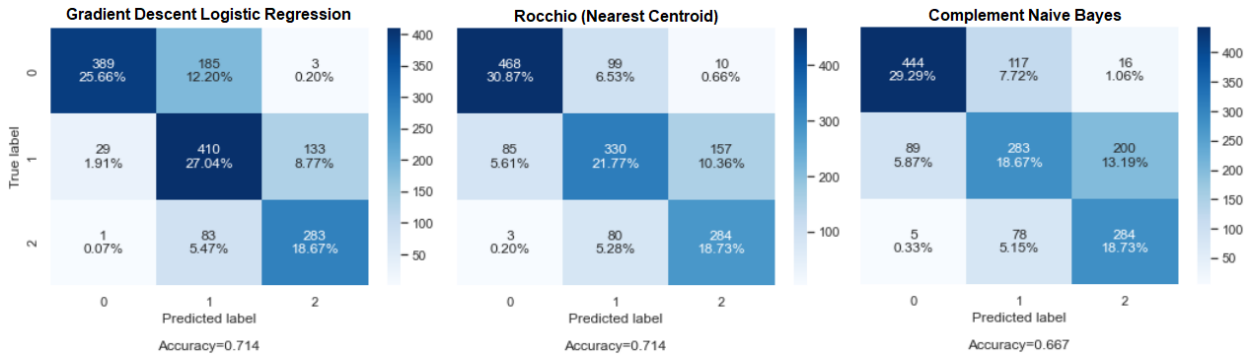
The logistic regression model also has the smallest difference in training and testing accuracies of all the models tested (4% difference). While Rocchio has a 7% difference and Naive Bayes a 20% difference. The latter two models overfit the data more.

Model Performance with Doc2Vec

We observed that all our models didn't do very well with Doc2Vec. This was surprising because we read many good things about using this type of transformation. The reason why it didn't do well is perhaps because we didn't tune the Doc2Vec model enough with the training data. Considering that Doc2Vec is a deep learning model, we found that tuning certain parameters was very time-consuming (for example, tuning epoch). Therefore, we decided to only tune a small set of parameters given the time constraints we had.

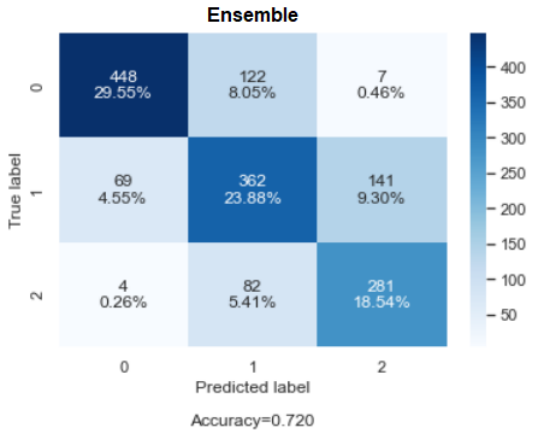
Model Performance with Unbalanced Data

To further evaluate the best three models we test the models using unbalanced test data. The original test data had instances of non-toxic (577), toxic (572), and severely toxic (367). For using unbalanced test data we doubled the toxic instances to 1,144. From the confusion the analysis results we see that the Complement Naive Bayes model improved performance on this test data. This may have occurred by giving increased instances to toxic so the model could differentiate them better. The toxic class is where all of our models struggled to predict.



Model Performance with Ensemble

In an attempt to improve our models we created a simple ensemble model which combined these three best models by a “hard” vote, or a simple majority vote between the models.



Model Performance in Relation to Training/Test Data

It is important to note that we trained on a subset of data available to us. Our training set had 4,398 observations and our test data set had 1,516 observations. We could have used up to 63,978 observations; however, we decided not to for the following reasons:

1. We wanted to make sure to train on a balanced data set. 90% of the original training set had comments which were classified as one of the toxic categories. Therefore, that meant we had to reduce the amount of non-toxic comments by a lot to have a balanced set.
2. Certain operations took a very long time depending on how big the data set was. For example, generating a Doc2Vec took at least 1 hour when being run on the original training set.

Training and testing with more observations would have surely increased the accuracy score of our models. However, given the time and resources available to us, we made the decision to use the subset of data to ensure we were able to complete all the tasks at hand.

Application Development

We developed a very simple command-line application that trains the best 3 models we evaluated and puts them into an ensemble model for the application to use. The best 3 models used is the following:

- Stochastic Gradient Descent Logistic Regression using the Tfidf matrix and normalized truncated Singular Value Decomposition.
- Rocchio (Nearest Centroid) using the Tfidf matrix.
- Complement Naive Bayes using the Tfidf matrix.

The application will prompt the user for a comment and then classify it using the chosen model. This application resides in the Toxic_App/ directory and is called *toxic_app.py*. It can simply be run by running *python toxicApp.py*.

Application Performance

When we tested our application we found that it tended to classify some comments that are non-toxic as toxic or severe-toxic. The difference between the 2 primarily has to do with whether there are swear words in the comment itself. If you were to put a comment that contains a top 10 word from toxic or severe toxic categories, our application usually classified the comment correctly. We are not surprised by these results considering we only got an accuracy score of around 72% for the ensemble mode which we are using in the application. We also focused more on reducing the false negatives.. The reason is because we would rather be wrong with classifying a non-toxic comment as toxic or severe- toxic than vice-versa.

One interesting observation was with the comment “I like sunshine.”. Our application was classifying this comment as toxic. We found out it was because of the word “like”. When looking at the exploration we did at the beginning of the project, one interesting thing we saw was that the word “like” was in the top 10 words for comments which were classified as non-toxic and mild-toxic. Even though we ended up dropping the mild-toxic category; somehow, our model classifies this word in the toxic category. I believe that if we were to have increased our training and test data for our models, the application would have achieved a better accuracy for classifying such comments.

Appendix

Data Set

We used a data set from a Kaggle competition which can be accessed here: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data?select=test.csv.zip.

Sample Test Run of toxic_app.py

```
/Users/lisasaurus01/opt/anaconda3/bin/python
/Users/lisasaurus01/git/ToxicApp/Toxic_App/toxic_app.py

[nltk_data] Downloading package stopwords to

[nltk_data]     /Users/lisasaurus01/nltk_data...

[nltk_data]   Package stopwords is already up-to-date!

[nltk_data] Downloading package wordnet to

[nltk_data]     /Users/lisasaurus01/nltk_data...

[nltk_data]   Package wordnet is already up-to-date!

[nltk_data] Downloading package punkt to

[nltk_data]     /Users/lisasaurus01/nltk_data...

[nltk_data]   Package punkt is already up-to-date!

Welcome to the Toxic Comment Identifier App!

Please enter a comment to classify or else 'q' to quit:my name is lisa

This comment was classified as: non-toxic

Please enter a comment to classify or else 'q' to quit:Before adding a new product to the
list, make sure it's relevant

This comment was classified as: non-toxic

Please enter a comment to classify or else 'q' to quit:u suck

This comment was classified as: severe-toxic

Please enter a comment to classify or else 'q' to quit:i don't like you
```

```
This comment was classified as: toxic

Please enter a comment to classify or else 'q' to quit:i like sunshine

This comment was classified as: toxic

Please enter a comment to classify or else 'q' to quit:The word bitch is actually only
offencive in American and Canadian English.

This comment was classified as: severe-toxic

Please enter a comment to classify or else 'q' to quit:q

Good Bye!


Process finished with exit code 0
```

README.md

A README.md file is located in the home directory of this project.

GitHub

Our entire project is located in Github in this address:

https://github.com/jjbocek/ToxicApp

Presentation

A presentation of our project can be accessed here https://vimeo.com/809245926#t=15