Toxic Comment Detection

Hsuan-Yu Yeh, Qianhong Zhou, Johannes Zhou
University of Michigan
{hsuanyu, [redacted]}@umich.edu

April 17, 2018

Abstract

A textual data set of comments taken from Wikipedia is analyzed to explore possibilities in quantifying and visualizing toxicity in online comments. The original data set is supervised, with human determined labels on each comments toxicity level and category (insult, threat, etc). We apply classification techniques, using Nave Bayes, SVM, and Random Forest as a baseline to identifying toxic comments. We try to define the boundaries between levels of toxicity and different labels by quantifying the text and finding key words for each comment label. By using clustering and topic modeling we look for different ways to examine the differences between toxic comments outside of the original labels. Our results can be used to improve automated detection of toxicity in online communities.

Group Member Contributions

Hsuan-Yu Yeh: feature extraction, dimension reduction(PCA), topic modeling(LDA), clustering(K-means) and classification(NB, LDA, KNN, Random Forest). Discussion and Conclusion.

[redacted] feature extraction, data resampling, dimension reduction(FA), clustering(Hierarchical clustering) and classification(SVM). Discussion and Conclusion.

[redacted] data description, data preprocessing, feature extraction, cosine similarity, topic modeling. Introduction, abstract, future steps part in discussion.

1 Introduction

The internet has now permeated throughout all facets of human society, allowing people all over the world to connect and share information. A large part of the online experience is user interactions; many of the most active websites are communities that facilitate content sharing and discussion among its users. The importance of monitoring and studying these online interactions is already apparent and has been visited in many fields of study such as anthropology and sociology. While the applications for understanding human behavior and sentiment are numerous, online comments and interaction can also be studied to help improve the experience for users. Internet communities have always been plagued with messages directly attacking other users, spam, and other forms of toxic and unproductive behavior. These messages not only obstruct constructive discussion, but can actively facilitate hate among large groups of people and hurt the targeted individuals.

To combat this kind of content appearing on sites, discussions within communities are often moderated by appointed individual users. Analyzing and identifying toxic comments works toward a more consistent and automated form of moderation. This automation may lighten the load for human moderators or hypothetically remove the need for them. This study aims to analyze how human moderators would classify toxic comments in hopes of being able to better emulate their methods. We consider whether the human labels in the data can be replicated by models, and if the differences between labels can be identified.

1.1 Data Set

The data set is composed of 600k comments taken from Wikipedia. Each comment has been rated by a human for toxicity on a 0-1 scale. Furthermore, the comment is labeled based on their judgment over whether it qualifies as severely toxic, obscene, a threat, an insult, or identity hate. We found that about 10% of the dataset was labeled toxic, meaning this dataset is very imbalanced.

1.2 Descriptive Statistics

We conducted some preliminary exploration of the data set by looking at the labels and the words involved with each. Label frequency Fig.1 indicates the strong imbalance present in this set, with the vast majority of comments nontoxic. The four alternate labels $insult, obscene, severe\ toxic, threat$ are clearly subsets of the toxic comments. Looking at the number of comments belonging to each subset, it is also clear that many comments can fall under multiple subcategories.

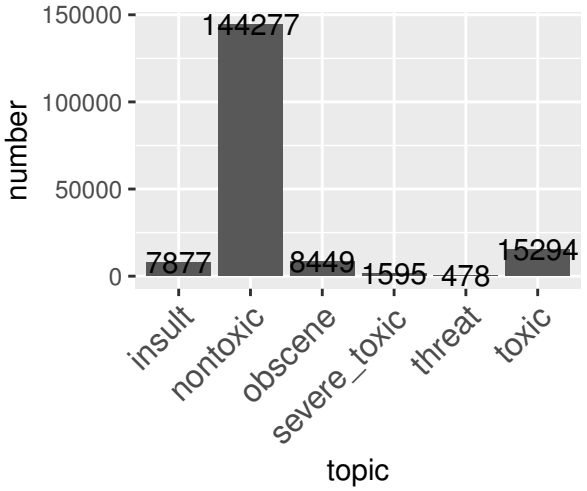


Figure 1: The message number of different topics

Table 1: The top 10 high frequency words in different toxic comments

non-toxic	toxic	obscene	threat	insult	identity_hate
article	f*ck	f*ck	die	f*ck	n*gger
wikipedia	moron	f*cking	ass	n*gger	die
page	ass	n*gger	kill	suck	faggot
talk	wikipedia	ass	block	f*cking	f*ck
like	hate	sh*t	jim	die	huge
just	like	bullsh*t	wales	hate	n*Vggas
best	hi	cunt	supertroll	sh*t	like
don	die	suck	f*ck	bitch	page
think	wanker	die	ban	moron	fucking
edit	f*cking	fat	don	faggot	wikipedia

Looking at each label individually, a survey of the most frequent words for each label allows us to get a better idea of the comments in each category. There is a clear distinction between toxic and nontoxic comments, but the difference between toxic labels is less clear. Common swear words and insults are prominent in all sub-labels. As previously mentioned, since many comments fall under more than one label, it isn't surprising there is overlap in words between labels. The overlap provides a challenge to determining the differences between categories.

2 Feature Extraction

2.1 Data Selection

Our original dataset contains 159571 comments, in which 15294 are toxic comments and 144277 are nontoxic comments. Toxic comments account for 9.58% of the entire dataset, which means our dataset is very imbalanced. This may affect the sensitivity and performance of classification models. Furthermore, extremely imbalanced datasets make any sort of classification accuracy not so meaninngful, since we can always get a pretty high accuracy if we simply classify all the data points as the majority class.

Apart from imbalance, another issue is the size of our feature matrix, which involves sample size and number of features. A total of 159671 observations is considerably large for the current resource we have to fit certain classification models, particularly SVM and Neural Network. Thus shrinking the size of the dataset would be a reasonable option. Also, the number of features we get from feature extraction may be reduced as well.

We thought of three options to fix the imbalance problem. First, we could run a bootstrap from the underrepresented toxic class with replacement to obtain a class sample size as large as the larger non-toxic group. But this method would further enlarge our data size. Another approach is to sample randomly from the nontoxic comments to obtain a sample subset of the same size as the toxic comment class. This would result in loss of information from the original dataset, since we would be removing about 80% of the data. Third, if the data size still seems large after the second way, we can maintain the class ratio of 1:1 and sample from both classes.

To determine the optimal approach, wee implement a multinomial Naive Bayes model for variant sample sizes and number of features (1000, 500, 100). The text in red shows that using 100 features, sampling 5000 observations from both classes to form a smaller dataset would only result in a small amount of loss in accuracy (Tab.2). So we decide to proceed with the smallest subset (5000 points sampled from each class).

On the other hand, we can see reducing number of features would indeed result in a considerable amount of loss in accuracy. Compared to the 5000:5000 sampling strategy, if we use 1000 features, the test accuracy is 0.834, while using 100 features only gives an accuracy of 0.745, losing 8.9 percentage points. However, considering the limited resources we have, and also our expectation that other classification models may achieve better results with this method, we decided to choose 100 features.

Table 2: Test accuracies of Naive Bayes model using variant sample sizes and variant number of features.

	class ratio	class size	count vector			tfidf		
	toxic:nontoxic	toxic:nontoxic	1000	500	100	1000	500	100
original		15294:144277	0.944	0.937	0.912	0.944	0.938	0.918
bootstrap from toxic class	1:9	16031:144277	0.940	0.937	0.915	0.941	0.937	0.917
	2:8	36069:144277	0.905	0.896	0.864	0.907	0.897	0.864
	3:7	61833:144277	0.875	0.864	0.819	0.883	0.870	0.823
	4:6	96185:144277	0.861	0.849	0.799	0.874	0.862	0.807
	5:5	144277:144277	0.837	0.818	0.764	0.861	0.846	0.784
sample from nontoxic class	1:9	15294:137646	0.944	0.937	0.912	0.945	0.939	0.917
	2:8	15294:61176	0.910	0.902	0.870	0.911	0.903	0.871
	3:7	15294:35686	0.884	0.873	0.835	0.887	0.878	0.836
	4:6	15294:22941	0.851	0.841	0.790	0.861	0.852	0.803
	5:5	15294:15294	0.833	0.817	0.766	0.855	0.839	0.786
sample from both classes	5:5	10000:10000	0.835	0.811	0.757	0.859	0.841	0.781
	5:5	8000:8000	0.832	0.813	0.754	0.856	0.841	0.781
	5:5	5000:5000	0.834	0.810	0.745	0.858	0.847	0.770

2.2 Vectorization

Since the data set is composed of a large number of comments, data cleaning and transformation are important. Our goal is to analyze the text with machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

In order create a data set that can be analyzed we extract a feature matrix with vectorization methods, which entail the following steps:

-Tokenization assigns integer values to predetermined 'tokens' of text, perhaps using whitespace or punctuation as seperators. The integer ids are then counted to tally the occurrences of tokens in each document.

-Judging by occurrence frequency in the text, normalizing and weighting the tokens with diminishing importance.

After vectorization, features and samples are defined as follows: Each individual token occurrence frequency (normalized or not) is treated as a feature. the vector of all the token frequencies for a given document is considered a multivariate sample. Features are combined into a matrix with one row per document and one column per token occurring in the dataset.

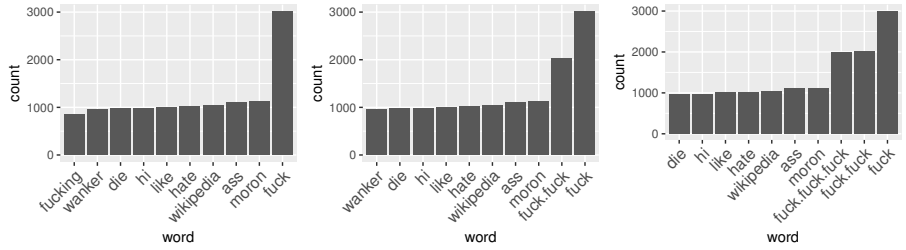
This specific strategy (Bag of Words or Bag of n-grams) describes text data by quantitative word occurrences, and does not take into account the position of the words in the document.

As mentioned above, a second possible step is normalizing and weighting the tokens by frequency. In this study we apply both a basic word count vectorization and a vectorization that takes frequency into account. The tf-idf vectors multiply term frequency and inverse document frequency of tokens for a weighted, normalized feature matrix of the text data.

Sample Size:5000 toxic+5000 non-toxic (8000 train, 2000 test)

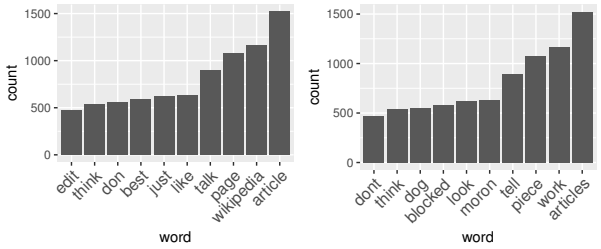
Table 3: Prediction accuracy for different feature-extracting settings with Naive Bayes model

Feature Num	Char	Word1	Word2	Word3	Char_tfidf	Word1_tfidf	Word2_tfidf	Word3_tfidf
5000/1000	0.791	0.823	0.767	0.739	0.838	0.86	0.848	0.839
1000/500	0.779	0.801	0.749	0.732	0.807	0.839	0.829	0.826
500/100	0.758	0.748	0.711	0.694	0.781	0.775	0.751	0.744



(a) Maximum 1 word (b) Maximum 2 words (c) Maximum 3 words

Figure 2: Top 10 toxic words in feature matrix

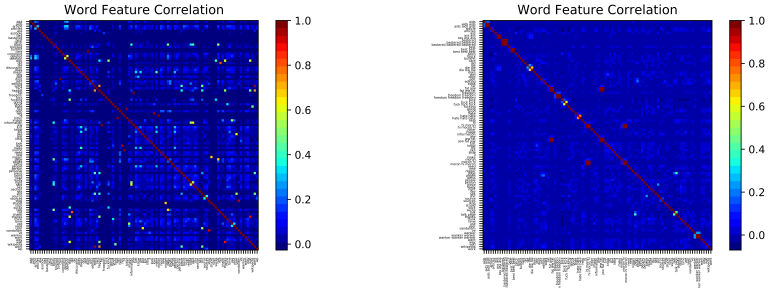


(a) Count feature matrix (b) Tf-Idf feature matrix

Figure 3: Top 10 non-toxic words in feature matrix

3 Dimension Reduction

The document term matrix is extremely large and sparse, a common phenomenon for vector representations in text analysis. We try to conduct dimension reduction to alleviate this issue. Singular Value Decomposition is a dimensionality reduction technique and Principle Component Analysis/Factor Analysis are two special cases of SVD based on the feature's covariance and correlation matrix. Since only some words show in a certain comment and the other words are contained in other comments separately, the feature matrix(document-term matrix) is very sparse. Thus, the covariance and correlation between most of the words are low as Fig.4.

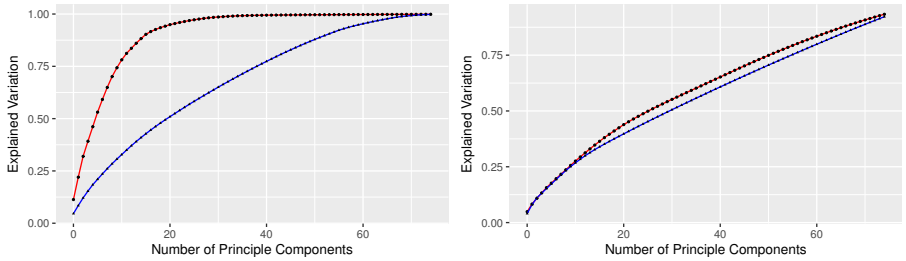


(a) Count Matrix (b) Tf-idf Matrix

Figure 4: The correlation matrix for feature matrix

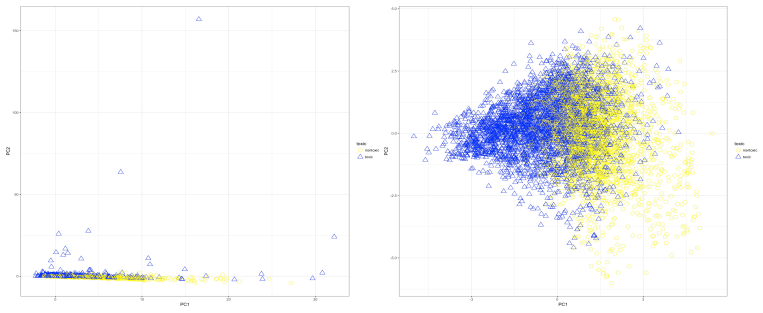
3.1 Principle Component Analysis

The result of PCA is shown in Fig.6. As we can see, for the count vector feature matrix, most of the observations locate around the origin. For the Tf-idf feature matrix, even though the observations can be approximately seperated, the oberservations in two groups overlap with each other severely. Also, as we only take few principle components, the explained variance ratio is low as shown in Fig.5. Generally, PCA does not perform well.



(a) Covarice Matrix (b) Correlation Matrix

Figure 5: The proportion of variance explained by PCs, red: Count Vector feature matrix, blue: Tf-idf feature matrix



(a) Count Matrix (b) Tfidf Matrix

Figure 6: The 2-dim PC scores plot.

3.2 Factor Analysis

Factor Analysis is a model-based dimension reduction technique and aims to relate a given set of variables to a set of common underlying factors. Perform factor analysis on both count vector/ tfidf feature matrices, and conclude the result in Tab. 4 and Tab. 5. As shown in Tab. 4, using the count vector features, the p-value for FA using varimax rotation with only one factor is 0, which means one common factor is not sufficient to explain the data. However, two common factors are still not enough to explain the data, since the p-values are 0 too. Also, the cumulative proportion of variance explained by the two factors is poor, again confirming that our FA model is not good enough to explain the data. Our data may not lie near a 2-dimensional linear submanifold, and also may not follow the probabilistic assumptions that FA model assumes. For the tfidf features, FA performs poorly as well.

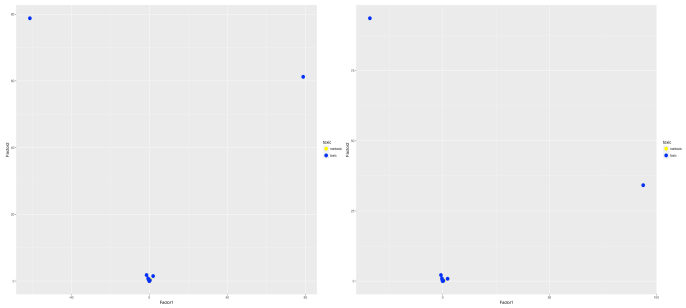
Draw the scores plot, and label the points by the class. As shown in Figure 7(a) and 7(b), for FA on count vector features, most points are aggregated around the origin. This is because in both cases, only variable 14, 26, 39 and 43 have non-zero loadings on both Factor 1 and Factor 2. Also, few outliers are distinguished from the majority. As shown in Figure 7(c) and 7(d), FA can not distinguish two classes very well, since two classes are overlapping considerably. In these two cases, a number of variables possess non-negative loading on the two factors.

Table 4: Factor Analysis Using Count Vector Features

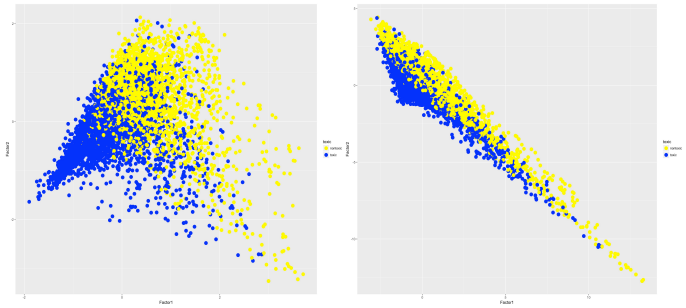
	K=1, varimax	K=2, varimax		K=2, promax	
factor	Factor 1	Factor 1	Factor 2	Factor 1	Factor 2
proportion var	0.020	0.020	0.020	0.020	0.020
cumulative var	0.020	0.020	0.040	0.020	0.040
p-value	0	0		0	

Table 5: Factor Analysis Using Tfidf Features

	K=1, varimax	K=2, varimax		K=2, promax	
factor	Factor 1	Factor 1	Factor 2	Factor 1	Factor 2
proportion var	0.012	0.011	0.009	0.060	0.056
cumulative var	0.012	0.011	0.019	0.060	0.116
p-value	0	3.97e-272		3.97e-272	



(a) varimax, count vector features (b) promax, count vector features



(c) varimax, tfidf features (d) promax, tfidf features

Figure 7: Scores Plot of Factor Analysis using Varimax/Promax Rotation.

3.3 Summary

Overall, FA's performance is similar to PCA when describing the data in a much lower-dimensional space, and performs poorly for distinguishing two classes of data. The reason for this poor performance is the sparsity of the document-term matrix. To deal with this problem, we could try Latent Sementic Analysis(LSA), which performs SVD on the document-term matrix instead of the covariance matrix. LSA is a technique for computing the semantic similarity of words in document-term matrix, and would most likely be more applicable to our dataset.

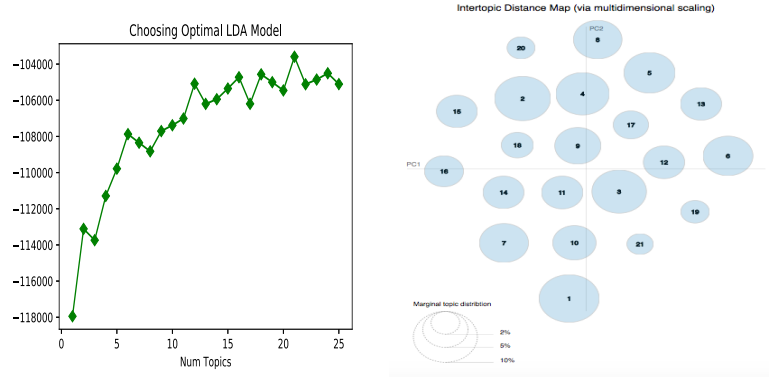
4 Topic Modeling and Clustering

There are total 100 feature words for the comments. We try to organize these comments with the document through some methods. In topic modeling, a topic is decided by a cluster of words. Different

topics may share the same words and each comment is considered a mixture of topics. We try to discover the hidden topical patterns that are present across the data. Also, try to annotate the comments according to these topics. In clustering, the basic idea is to group comments into different groups based on some similarity measure. This will yield only one topic with each comment.

4.1 Latent Dirichlet Allocation and Non-Negative Matrix Factorization

In text categorization, Latent Dirichlet Allocation (LDA)[**?**] is a popular method for topic modeling. Given the word count vectors, we can find the conditional distribution of different topics and thus, build up the joint distribution of the hidden topics. To decide the number of topics, we look at the models' log likelihood scores and set the topic number to 21 from the results. After projecting the data onto 2-PCs space, we can visualize the distribution of groups and thus, measure similarity between each group (Fig 8). The keywords that are representative for each group are shown in Table 6. LDA for the 10000 wikipedia comments produces two topics with noisy data (Topic 1 and Topic 12) and also some topics that are clearly nontoxic (i.e. Topic 2-5, and Topic 10). Looking over the remaining topics, we find some topics are more obviously toxic (Topic 1, Topic 6, Topic 9 and Topic 17) while others are more ambiguous and a mix of aggressive words and words that are hard to interpret without more context.



(a) LDA mse plot (b) Intertopic Distance Map

Figure 8: LDA results

Table 6: Top-5 most relevant terms for each topic

cluster1	f*ck	ass	sh*t	asshole	dick
cluster2	don	time	did	care	want
cluster3	wikipedia	deletion	articles	article	deleted
cluster4	page	talk	image	discussion	really
cluster5	like	people	dog	little	just
cluster6	fat	fucking	dont	stupid	penis
cluster7	article	think	life	comment	piece
cluster8	good	edits	section	help	editing
cluster9	n*gger	going	use	source	hitler
cluster10	make	say	said	does	new
cluster11	edit	stop	pages	vandalism	link
cluster12	hate	sex	wp	-	-
cluster13	die	aids	wiki	sucks	-
cluster14	wanker	freedom	best	suck	-
cluster15	know	just	ve	person	ll
cluster16	right	block	hey	personal	articles
cluster17	jew	b*tch	bad	faggot	-
cluster18	user	blocked	did	-	-
cluster19	moron	hi	-	-	-
cluster20	hi	moron	-	-	-
cluster21	bastered	bullshit	-	-	-

We dig further into topic modeling to try to find the differences in the dataset between comments that were labeled as toxic and those that were not. Since imbalance should no longer be an issue for unsupervised methods like topic modeling, we decided to return to the original data set and divided it into toxic and nontoxic sets. Furthermore, we explored applying non-negative matrix factorization(nmf) as a alternative topic modeling approach. By modeling the toxic set we hoped to gain some insight into the different toxic labels and what convinced the human raters to give a comment a particular label. Looking at the toxic topic results, we note that each topic does have a fairly unique set of words, which implies sublabels should be distinguishable. The topics themselves are not immediately obvious, but upon scrutiny a rough pattern can be seen in each group. The associations of the words in each topic could prove useful in identifying toxicity when seen in close proximity. The most compelling information from the nontoxic results is some insight into what moderators would not rate as toxic. Some words that appear in these groups are not innocent, but can be colloquially used in a less aggressive manner in banter. This would prove very useful for an automated process which would normally have difficulty understanding context, and begin recognizing some words as non-toxic when seen in close proximity.

Table 7: Topic Models for Toxic and Nontoxic Comments

Toxic Comments											
NMS	Topic 0	f*ck	asshole	shut	cunt	faggot	hey	want	mother	f*ckin	n*gger
	Topic 1	like	don	wikipedia	just	page	know	people	stop	talk	article
	Topic 2	f*cking	cunt	faggot	asshole	mother	retard	moron	life	hope	idiot
	Topic 3	suck	dick	cock	asshole	balls	hey	big	cunt	ur	penis
	Topic 4	bitch	sh*t	ass	piece	little	son	eat	faggot	f*ckin	hell
	Topic 5	gay	faggot	fag	sex	ur	ass	like	im	homosexual	likes
LDA	Topic 0	moron	die	hi	wikipedia	aids	wiki	idiot	pig	love	stupid
	Topic 1	like	don	just	gay	wikipedia	f*cking	page	people	know	stop
	Topic 2	sh*t	fat	bitch	nipple	wanker	piece	dont	buttsecks	super	care
	Topic 3	suck	sucks	dick	fag	f*cking	dog	bark	loser	balls	mother
	Topic 4	f*ck	n*gger	cunt	dickhead	yourselfgo	like	homo	user	bitches	cuntbag
	Topic 5	hate	ass	faggot	jew	cock	penis	huge	f*cker	shut	bastered
Nontoxic Comments											
NMS	Topic 0	ip	douche	issues	thought	kind	nonsense	time	view	gg	salt
	Topic 1	talk	read	nigga	community	user_talk	asking	ck	claim	does	knowledge
	Topic 2	blocks	edited	communist	ve	didnt	edie	niggas	steal	women	ignorant
	Topic 3	nigga	women	whore	thing	useful	hey	half	user_talk	dicked	said
	Topic 4	attack	dicked	attacking	speak	di	stupidity	sure	huh	scum	somebody
	Topic 5	things	happen	half	language	view	names	wouldn	hanibal911you	cause	gg
LDA	Topic 0	attack	nigga	talk	women	edie	attacking	dicked	wrote	di	comments
	Topic 1	women	nigga	hear	talk	valid	whore	half	niggas	chula	stupid
	Topic 2	nonsense	writing	useless	harassment	enigmaman	minorities	yo	useful	standards	kind
	Topic 3	women	hey	useful	knew	thing	niggas	nigga	cool	fat	communist
	Topic 4	attack	somebody	scum	sock	thought	huh	don	old	administrator	knowledge
	Topic 5	ip	douche	issues	kind	thought	things	nigga	view	talk	time

4.2 K-means++

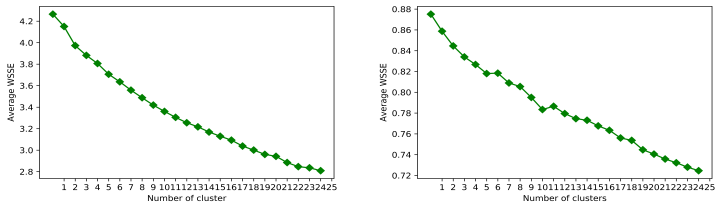
K-means clustering groups similar observations in clusters in order to extract insights from a vast amount of unstructured data. K-mean++ is an algorithm for choosing the initial values for the K-means clustering algorithm. Through the elbow method, we decide the optimal number of clusters as 21 and 10 for Count and Tf-idf feature matrices. For the Count feature matrix, the result seems to be not fair, most observations are grouped into one cluster Tab.8. Since many groups contain only one toxic comment, we only list out the first 6 clusters in the table. For the Tf-idf feature matrix, the clusters perform better. The comments are equally assigned to the 10 clusters on Tab.9. Cluster 1, Cluster 8 and Cluster 9 are mainly composed of toxic comments while the ratio of toxic and non-toxic comments in other clusters is about 0.5. For visualization, we transform the Tf-idf feature data onto 2-PCs space. However, it's hard to explain the clustering result in a low dimension space, as can be seen in Fig.10.

Table 8: Count Feature Matrix, 21 clusters, including the number of comments in each cluster, the ratio of toxic comments and the top-5 high frequency words in each cluster.

cluster0	cluster1	cluster2	cluster3	cluster4	cluster5
7	9967	2	5	1	1
1	0.498	1	1	1	1
user	wikipedia	hi	f*ck	fat	wanker
sh*t	article	moron	page	jew	wp
like	page	wp	fucking	wp	good
asshole	like	going	penis	good	edit
person	talk	edit	hate	edit	editing

Table 9: Tf-idf Feature Matrix, 10 clusters, including the number of comments in each cluster and the ratio of toxic comments and the top-5 high frequency words in each cluster.

cluster0	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	cluster7	cluster8	cluster9
1088	496	4472	265	675	605	576	403	577	843
0.481	1	0.492	0.532	0.788	0.504	0.554	0.598	0.929	0.839
don	f*ck	wikipedia	going	talk	just	people	did	fucking	article
like	b*tch	page	like	page	like	like	edit	stop	wikipedia
know	sh*t	thanks	just	wikipedia	know	wikipedia	bloacked	f*ck	page
wikipedia	asshole	good	wikipedia	pages	wikipedia	don	editing	sh*t	articles
think	ass	user	think	user	make	just	pages	wikipedia	think



(a) Count vector features (b) Tf-idf vector features

Figure 9: Deciding the number of clusters.

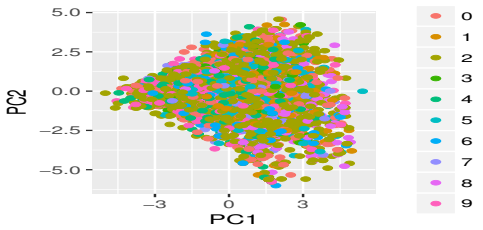


Figure 10: K-means++ clustering.

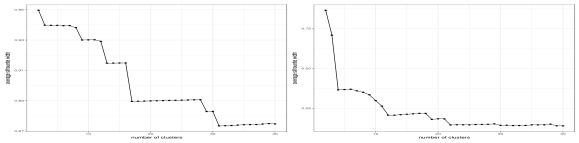
4.3 Hierarchical Clustering

Hierarchical clustering [**?**][**?**]is a bottom-up or agglomerative clustering approach. Unlike K-means, which requires us to pre-specify the number of clusters K, it does not require that we commit to a particular choice of K. For moderate size datasets, the solution can be visualized by a dendrogram. It start with n clusters, where n equals the number of objects in the data. Then at each step, merge the closest pair of clusters until all objects form a single cluster. The dissimilarity between two clusters is defined by the notion of linkage. After the dendrogram is developed, we can identify clusters by making a horizontal cut across the dendrogram. The distinct sets of observations beneath the cut can be interpreted as clusters.

We intend to implement single linkage, complete linkage, average linkage, and Ward's clustering on both count vector and tfidf feature matrices. First, to determine the optimal number of clusters, we use the average silhouette method to measure the quality of a clustering, which computes the average silhouette coefficient of observations for different values of K using complete linkage. The closer the average silhouette to 1, the better the clustering. The result is shown in Fig.12. In both cases, clustering with 2 clusters achieves the largest average silhouette. Thus we will proceed with K=2.

Summarize the clustering result in Tab.10. Average silhouette Width and Agglomerative Coefficient (AC) both measure the overall clustering quality, where AC is always between 0 and 1 and the closer it is to 1, the better the clustering. All the methods performs similarly, with one disagreement between average silhouette width and AC appearing in Ward's clustering using tfidf features. This is not common, since the two coefficients should agree with each other. We shall explore further below.

Only Ward's clustering using tfidf features is able to divide the data into the clusters of similar size. All the other cases all result in 9999 data points in one cluster and only 1 point in the other. Although Ward's clustering possess a low value of average silhouette, using tfidf features, it has a value of AC higher than all the other methods. Also, Fig.12(f) looks similar to the corresponding PCA and FA plots. Thus, although Ward's clustering using tfidf features only has an average silhouette of 0.028, it probably agrees with those pre-defined categories best among all these hierarchical clustering methods.

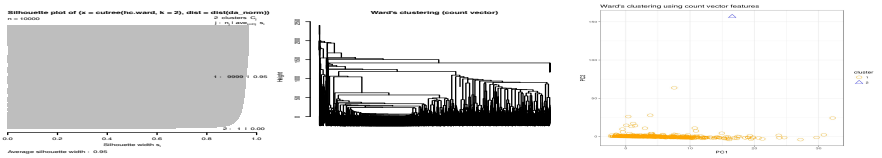


(a) count vector (b) tfidf

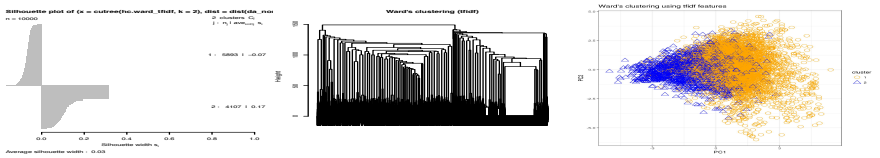
Figure 11: Average Silhouette Plot of complete linkage clustering on both count vector and tfidf features.

Table 10: Hierarchical Clustering

		Single	Complete	Average	Ward
Average Silhouette Width	count vector	0.950	0.950	0.950	0.950
	tfidf	0.865	0.865	0.865	0.028
Agglomerative Coefficient	count vector	0.984	0.985	0.982	0.987
	tfidf	0.961	0.962	0.958	0.971
Cluster Size	count vector	9999,1	9999,1	9999,1	9999,1
	tfidf	9999,1	9999,1	9999,1	5893, 4107



(a) Silhouette Plot, count vector (b) Dendrogram, count vector (c) Ward's clustering, count vector



(d) Silhouette Plot, tfidf (e) Dendrogram, tfidf (f) Ward's clustering, tfidf

Figure 12: Silhouette Plot and Dendrogram of Ward's clustering using count vector and tfidf features. (c) and (f) show data projected onto the first two PC directions labeled by the corresponding cluster.

5 Cosine Similarity

Cosine Similarity is a measure of distance between text data. Cosine similarity is a classical approach that quantifies the content overlap between documents. In order to explore the differences between the toxic sub-categories we created a weighted tf-idf document term matrix for each sub-label and calculated the cosine similarities between each label. This method defines the overlap between documents as the the angle between vectors, which make up our vector based representation of the text. The angle is calculated through the dot product of the two document vectors. Our results indicate that Severe and Obscene are most alike with a similarity of 0.036, while Insult and Threat are most different with a similarity of 0.021. Overall all similarities are low, which may simply be due to the high variance of what individual comments look like.

Table 11: Cosine Similarity

	Insult	Obscene	Severe	Threat
Insult	1			
Obscene	0.025	1		
Severe	0.032	0.036	1	
Threat	0.021	0.022	0.029	1

6 Classification

To classify comments into toxic and nontoxic classes, we implemented a number of classification models, on both Count Vector and Tf-idf feature matrices. The size of the data inspired us to reduce the dimensions so we also performed classification on PCA-preprocessed data, with the number of principal components retained such that 80% of the total variance is explained by the PCs.

Overall, we applied multinomial Naive Bayes(NB), Linear Discriminant Analysis(LDA), k-nearest neighbors(kNN), Support Vector Machine(SVM) and Random Forest on the feature matrices. We first fitted these models on original feature matrices (without dimension reduction). For NB and LDA, we used the default parameters provided by the Python machine learning package *scikit-learn*. For kNN, by the elbow method, we set $k = 27$ for count vector feature matrix, and $k = 17$ for Tf-idf feature matrix. For Random Forest, we chose the number of trees as 23 for both feature matrices. For SVM, using cross validation to tune the cost parameters, we used three different kernels, linear (cost=10 for count vector, $10^{2.5}$ for Tf-idf), polynomial (degree=2, cost=10^3, 10^3), and Gaussian (gamma=0.01, cost=$10^{1.5}$, 1). Test accuracies of different classification methods are shown in Table 12 and Table 13.

We first compare the performance of different methods applied on the original count vector data matrix. Generally speaking, SVM with linear kernel performs best (test accuracy=0.766) and Random Forest performs the second (test accuracy=0.759), while LDA and SVM with polynomial kernel have poor performance.

Comparing multinomial NB with LDA, both classifiers assume Gaussian class conditional densities, but multinomial NB also assumes independent variables, while LDA assumes a common covariance matrix. Multinomial NB is typically used for document classification, with events representing the occurrence of a word in a single document , and is suitable for classification with discrete features (e.g., word counts for text classification). We can see it achieves a pretty good test accuracy, 0.747. However, LDA performs worst among all the methods, which could be due to that it makes too many assumptions, and is very sensitive to outliers.

SVM with linear kernel performs the best. Our feature matrix has 100 features, and our document vectors are sparse, making our data matrix possibly linearly separable [**?**]. So SVM with linear kernel could be well-suited for text classification. In fact, SVM with linear kernel performs quite well in other choices of feature matrix (Tfidf-transformed, and PCA-preprocessed features). SVM with Gaussian kernel also performed quite well.

Random forest has a good performance as well in all choices of feature matrices. It is considered one of the most competitive classifiers.

For feature matrix with the term frequency-inverse document frequency (tf-idf) feature extraction method, we can see a general improvement in the classification performance. This could be due to that tf-idf has a mechanism of evaluating how important a word is to a document in a collection. While term frequency (tf) can reflect how frequently a word occurs in a document, there are words in a document, however, that occur many times but may not be important, for example, "the", "is", "of", and so forth. Thus looking at a term?s inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents can help solve this problem. So tf-idf may be a better feature extraction method than count vector, illustrating an important point in machine learning: better features tend to beat a cleverer algorithm. In fact, we can see in Tab.13 within the scope of PCA-preprocessed data, all classification performs better using tf-idf features than count vector features.

Also, we have to notice that LDA, previously performed worst using count vector feature matrix, improves a lot when we change to tf-idf features. So this means that perhaps after all, linear classifier should perform well on our text dataset, which is possibly linearly separable.

For the PCA-preprocessed data, since some information is lost by dimension reduction, we can expect that the classification result generally got worse. Multinomial NB is impacted most, while SVM seems more robust, especially SVM with Gaussian kernel.

Table 12: Prediction accuracy of different classification methods.

Feature Matrix	NB	LDA	KNN	SVM(linear)	SVM(polynomial)	SVM(Gaussian)	Forest
Count Vector	0.747	0.695	0.737	0.766	0.664	0.729	0.759
Tf-idf Vector	0.744	0.749	0.745	0.755	0.726	0.768	0.760

Table 13: Prediction accuracy of different classification methods using PCA-preoprocessed data

Feature Matrix	NB	LDA	KNN	SVM(linear)	SVM(polynomial)	SVM(Gaussian)	Forest
Count	0.499	0.656	0.644	0.73	0.661	0.723	0.701
Tf-idf	0.665	0.749	0.723	0.750	0.748	0.765	0.755

7 Discussion and Conclusion

The project is set out to identify toxic comments, and examine subgroups and define toxicity within the comments dataset we have. To begin with, dimension reduction methods, Principal Component Analysis and Factor Analysis did not perform well on our text dataset, possibly due to high dimensions and sparse feature matrices.

For text mining, topic modeling provides us with methods to organize, understand and summarize large collections of textual information. Through backtracking, we tried to figure out what topics would cause comments to be labeled in some way. The results confirm that there is a discernable difference between different types of toxic comments, and also opens up the possibility of studying non-toxic comments to recognize when bad words are used in a nontoxic manner.

In order to find meaningful groups of the data, such that objects in a group are similar to one another and different from the objects in other groups, we performed clustering, including K-means++ and hierarchical clustering. For K-means++, it is hard to find an optimal number of clusters since the within sum of squares error keeps going down when we increase the number of clusters, and it is also unclear how to explain the how data are similar to each other in a certain cluster. For hierarchical clustering, average silhouette width suggested us to set the optimal number of clusters as 2. All methods except for Ward's clustering applied on the Tf-idf feature matrix produce similar results: the data points are

subdivided into two clusters, 9999 objects versus 1 object. It remains to be shown why this one comment stands out among other comments. Meanwhile, this clustering result does not give us more information of the data since the majority of comments are in the same group. The Ward's clustering applied on Tf-idf feature matrix seems to give us the closest result to the pre-defined classes, however, this method seems to have the worst clustering quality since it has the lowest average silhouette width.

For classification, the performance of models applied on Tf-idf feature matrix is generally better than that of count vector feature. As mentioned before, this could be due to that Tf-idf feature extraction methods can produce better features than count vector method. Possibly due to high dimensionality of features and sparsity of document vectors, support vector machine with linear kernel performs generally the best, and is very robust after we performed PCA on the data, while other methods performed worse to different degrees with respect to test accuracy. This suggests that SVM with linear kernel could be a very suitable classification method for text categorization.

This study is only limited by its data set, with its collection of toxic comments and labels. For this data set, we could try different dimension reduction techniques like Latent Semantic analysis and Sparse PCA to tackle the challenges of sparse matrices. Since in text data has a discrete distribution, this sort of dataset does not meet the continuity assumption of PCA. In addition, we could expand the analysis by considering the entire dataset. Since the document-term matrix will be different, there may be some interesting hidden patterns revealed through topic topic modeling or clustering that we did not find with our sampled dataset. Finally, text classification using Neural Networks is a powerful option that we did not explore. Using methods like Convolutional Neural Nets (CNN) would let us take more complex approaches such as training on the character level rather than the word level. Word2vec is an NLP tool we came across that offers a variety of methods to vectorize and analyze textual data, and would also be worth applying to this dataset. More NLP methods in general would be applicable to this study. The CYK Model could allow us to see if the sentence structure of toxic comments are significantly different from nontoxic comments. In future work, toxic comment analysis can be extended many ways as Natural Language Processing and other text mining processes continue to improve and develop.

References

[1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*, 2017.

[2] Tibshirani Hastie and R Tibshirani. & friedman, j.(2008). the elements of statistical learning; data mining, inference and prediction.

[3] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[4] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.