Detecting and Classifying Toxic Comments

Nikita Bawane
UIN: 661069000

Ritu Gangwal
UIN: 670646774

Utkarsh Ujwal
UIN: 659563524

Abstract:

Social Media platforms have made it extremely easy for people to share their opinion and make a difference in the society through their words. While many of the online comments/opinions these days are constructive and healthy, there is a part of these comments which tend to be highly derogatory. These days people often make obscene and hurtful comments both offline and more frequently in the online world. Hence, it is essential to have an identification system which can evaluate the toxicity of the comments online.

With this motivation in mind, our project aims to perform multilevel classification of comments into six different levels of toxicity – toxic, severe-toxic, obscene, threat, insult, identity-hate. The dataset is a compilation of the labelled Wikipedia user comments by Jigsaw. Along with sentiment analysis and various feature extraction methodology, such as Word2Vec, Counter-Vector and TF-IDF, we build Machine Learning models to evaluate the probability of a comment belonging to one of the six labels. The metrics used to evaluate the model performance is Confusion Matrix and AUC.

Keywords: Toxicity, Feature Extraction, TF-IDF, Counter-vector, Machine Learning, Sentiment Analysis, TextBlob, Vader, Accuracy, ROC, AUC, Multilevel classification, Correlation, Support Vector Machine, Naïve Bayes, Logistic Regression, Confusion Matrix

I. INTRODUCTION

These days on internet, users operate behind the shields of anonymity and behave ways in which they would ideally not behave in person. Nowadays derogatory comments are made by one another, not only in offline environment but also in online environments like social networking websites and online communities. It is not uncommon to watch celebrities have "twitter fights", users troll each other and openly threaten one another when they are not in agreement. For organizations which host these platforms, it is important for them to analyze these comments and address them correctly to curb their future occurrences.

II. PROBLEM STATEMENT

A research initiative, Conversation AI team, by Jigsaw and Google works on identifying toxic comments and improving online conversation. Their focus is on understanding the online behavior and have already built various models to achieve this, but their models have errors as they do not categorize users based on the level of toxicity.
The goal of this research paper is to create a multi-head classification system which identifies the probability of the toxicity and assigns any of the six labels to a single comment. Each comment can belong to more than one label.

III. RELATED WORK

Sentiment classification regarding toxicity has been intensively researched in past few years, largely in the context of social media data where researchers have applied various machine learning methods to try and tackle the problems of toxicity as well as of sentiment analysis. Comment abuse classification research initially began with Yin et al's application of combining TF-IDF with sentiment/contextual features. They compared the performance of this model with a simple TF-IDF model and reported 6% increase in FI score of the classifier on chat style databases. Different approaches for detecting types of toxicity in comments have been attempted –
- Georgakopoulos et al. obtained an accuracy of 91.2% by implementing Deep Learning models involving Convolutional Neural Networks (CNNs).

- Khieu et al. obtained a label accuracy of 92.7% by applying various Deep Learning approaches like LSTM for classifying online comments based on toxicity.

- Chu et al. obtained mean accuracy of 94% by implementing methods like Convolutional Neural Network (CNN)with character level embedding for detecting toxicity for comments.

– Kohli et al. obtained a Mean Accuracy of 97.78%. by implementing a Deep Learning method i.e, Recurrent Neural Network (RNN) LSTM with Custom Embeddings for classification of comments based on toxicity.

The main unexplored issue is to identify algorithms that can implement high sensitivity in detection of toxic comments.
Users can get frustrated if a comment ,which is not toxic, is identified as toxic. Hence it is essential that a model is made which has the highest degree of sensitivity. In this paper, we have used both sentiment analysis and machine learning models to classify toxicity, making this paper a novel and practical work in toxic comments classification.

IV. DATASET DESCRIPTION

After an extensive data search, we decided to go with Toxic Comment Classification Challenge dataset which contains comments which are profane. This dataset, found on Kaggle, was generated by Conversion AI Team by labelling Wikipedia Talk Page comments. The data contains comments and six toxic labels, namely - toxic, severe-toxic, obscene, threat, insult, identity-hate. There are four files available on Kaggle and we will be using three:

a. train.csv – contains comments with their true labels

b. test.csv – contains comments whose toxicity probabilities must be predicted, and labels must be assigned.

c. test_labels.csv – contains true labels of the test data. Some of the comments are marked -1 as they are not included during prediction. This file was made available after the Kaggle competition was over.

V. METHODS AND TECHNIQUES

V.I Data exploration

Before even contemplating the relationship between dependent and independent variables, it is first crucial to become familiar with the contents of the modeling data by exploring the distributional properties of each variable.

One of the primary challenges in classification cases is having appropriately labeled data, in which a representative training set could be extracted for modeling. For Natural Language Processing [NLP] problems, this dilemma is even more prominent. Although many large unlabeled text corpora are readily available, human labeled data is much more infrequent.

Visualizations are done in the form of Histograms showing the distribution of comment lengths over the whole corpus of the dataset. We have used Python (NLTK library) for data preprocessing, exploration, and modeling.

V.II Text pre-processing

Text pre-processing, also identified as the data cleaning process, implicates that the outliers or noisy data are removed from the target dataset. This stage also incorporates the development of any strategies required to deal with the inconsistencies in the target data.

The text preprocessing techniques followed before processing the text data are:

a. **Tokenization:** Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols, and other elements called tokens. Hence here all the sentences are broken down into words/tokens by using nltk.tokenize.

b. **Remove IP address** - IP addresses in comments are unnecessary and would take up a lot of memory space when converted to vectors.

c. **Non- Alphabetic characters** – After tokenizing, we have converted all our sentences in lower case and then removed all the non-alphabetic characters like numbers, punctuations, etc. Since we have defined Regex tokenizer as *RegexpTokenizer(r'([a-zA-Z]+)')*, we remove all characters (numbers and special characters) and only retain alphabets.

d. **POS Tagging:** After the initial clean up, we have given POS to each word by using nltk.pos_tag to get the correct sense of each word in a particular sentence.

e. **Lemmatization:** It usually refers to giving root word properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. For our datasets, we have used nltk.stem.WordNetLemmatizer() to lemmatize each word based on its POS.

f. **Removing stop words** - Words such as articles and some verbs are usually considered stop words because they do not help us to find the context. Hence these were removed from all our datasets.

V.III Sentiment Analysis

Sentiment Analysis techniques help us analyze and determine the sentiment behind a text (or sentence). Using some basic sentiment analysis techniques, we can know if the sentiment behind a text is positive, negative, or neutral.

Why do we need Sentiment Analysis on our dataset? During the data exploration, we discovered that around 10% of the comments in the training dataset are toxic. We have 159571 comments and creating feature vectors from these many comments would require high computational space. Hence, we decided to perform sentiment analysis to retrieve only those rows which have negative sentiment. In the training set, we have the true labels but while handling the test set, this step is crucial as it will help reduce the number of comments used for creating feature vectors on this huge dataset (Note that test data too, has around 150k comments).

As per our design decision, we will perform sentiment analysis on any dataset before feature selection, so that the positive comments can be filtered out and only negative comments are retained.

Types of Sentiment Analysis Performed: We have used two common libraries to perform this task –

a. TextBlob – This is a Python (2 or 3) library which is used to process text data. It contains simple APIs to perform NLP tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification etc. We will be using this library to perform sentiment analysis. This contains two major metrics –

i) Polarity – The polarity value varies from -1 and 1, where -1 denotes negative sentiment and 1 shows positive sentiment.

- if polarity score < 0 -> 'Negative'
- else if polarity score == 0 -> 'Neutral'
- else -> 'Positive'

ii) Subjectivity – This score tells us if the text is more of an opinion or a fact. The value for subjectivity score varies from 0 to 1 where 0 means objective (fact) and 1 means subjective (opinions).

b. VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon and rule-based tool that is attuned to sentiments expressed on social media. VADER has been found to work well with social media texts, product and movie reviews etc. It tells us how positive and negative the sentiment is. It is fully open sourced under the MIT License. To get the sentiment score from VADER, a function called "polarity_scores()" is used to get different polarity scores.

- Positive
- Neutral
- Negative
- Compound

Positive, Neutral and Negative scores tell us what proportion of text falls in which category. The sum of all the scores should add up to 1.

Compound score calculates the sum of all ratings and normalized between -1 to 1. Most negative score is denoted by -1 and most positive score is denoted by 1.

The compound score is compared with the threshold value to categorize it Positive, Negative or Neutral sentiment. We used the value 0.05 and below is the algorithm –

- Compound value >= 0.05 = Positive
- Compound value <= -0.05 = Negative
- (Compound value > -0.05) and (Compound value < 0.05) = Neutral

V.IV Topic Modelling

The idea behind LDA is in using Dirichlet priors for the document-topic as well as term-topic distributions It therefore allows for Bayesian inference over a 3-level hierarchical model with the third layer being the distinguishing factor in comparison to a normal Dirichlet multinomial clustering model.

$$P(\theta_{1:M}, \mathbf{z}_{1:M}, \beta_{1:k} | \mathcal{D}; \alpha_{1:M}, \eta_{1:k})$$

If we have a set of M documents and each document have N words, where each word is generated by a single topic from K set of topics. Posterior probability of:

θ — Randomly initialized matrix where θ(i,j) is the probability of the ith document which contains the jth topic. It is the distribution of topic for each document,

z — N Topics for each document,
β — Another randomly initialized matrix where β(i,j) represents the probability of ith topic which contains the jth word. It is distribution of word for each topic
given,
D — Entire data (i.e. the corpus),
and using parameters,
α — Distribution related parameter that governs distribution of topics over all the documents in the corpus (document — Topic distribution)
η — It is a distribution related parameter governing the distribution of words in each topic. It is parameter vector for each topic. (topic — word distribution)

V.V Feature Extraction/Vectorization

After performing the sentiment analysis and segregating negative comments from positive comments, we must build models on comments to predict the toxicity of each comment. The problem with machine learning models is that it cannot process raw text data as is it. To bring the text data to a format (matrix or vector) understandable by the machine learning models, we use various feature extraction methods.

There are three feature extraction methods –

- Bag-of-Words (Count Vectorizer)
- TF-IDF
- Word Embedding – word2vec

1. Bag of Words/ Count Vectorizer – This is the most conventional and simple to perform feature vectorization method. Every word is used as a feature for training the classifier. It records the occurrences of each word appearing in every document. Basic steps involved to create this Bag-Of-Words model are:

- Tokenize every sentence to get tokens (words)
- Create dictionary so that there is no duplication of tokens and
- Calculate the count of each word.

Advantages –

- Simple implementation and easy to understand

Disadvantages –

- Order of occurrence is lost, and vector has randomized order of words. To overcome this, we can consider N-gram (say, Bigram words)
- Rareness of the term is not considered.

These drawbacks can be overcome by using TF-IDF.

2. TF-IDF – The main idea behind this method is if a word appears several times in the same document then it is awarded a higher score whereas if it appears several times across different document then its overall score decreases. There are two aspects to this –

- TF – Term Frequency – Specifies how frequently a word occurs in the whole document
  - Tf(wi, dj) = No. of times word (i) occurs / Total number of words in document dj

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

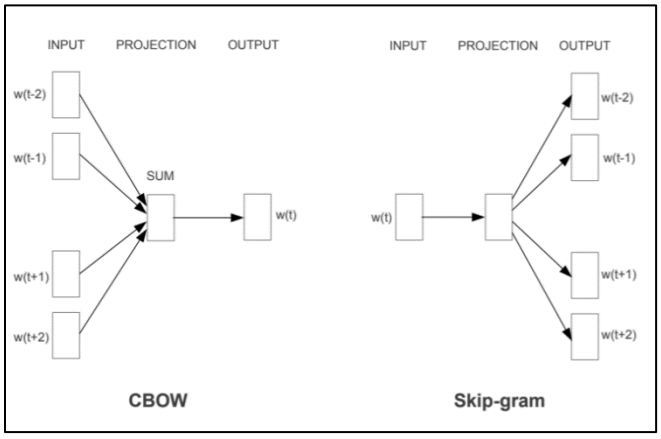
- IDF – Inverse Document Frequency – Specifies how frequently or rarely a word occurs in each document.
  - Idf = log {(total number of documents) / (total number of documents that contain the word)}

$$IDF = log_{10} \frac{Number\ of\ Document}{Number\ of\ document\ in\ which\ word\ appear}$$

3. Word Embeddings – Unlike the previously mentioned methods (BoW and TF-IDF), this method captures the semantic and contexts. It also detects similar words by preserving contexts and relationships between the words. In this project, we have used the Word2Vec implementation. It vectorizes words by using 2-layer shallow neural networks.
Word2Vec uses Cosine similarity to find out the similarity between words. Along with checking the similarity, Cosine Similarity also tests the orthogonality.
Word2Vec fundamentally used two types of algorithm – CBoW (Continuous Bag-of-Words) and skip gram model.

- In CBoW,
  - the target word is predicted from the context
  - to make the prediction, sum of context vectors is used.

- In Skip Gram,
  - predicts the context from target word.
  - It is a reverse algorithm of CBoW.
  - It can work well with raw data as it is unsupervised learning.
  - In comparison to other word to vector methods, this requires less memory.
  - Requires high training time.



V.VI Multinomial Classification Supervised Machine Learning Models

This section elaborates on the different algorithms implemented on the data set to build the predictive models. The techniques namely, Naïve Bayes, Logistic Regression and Support vector machine are implemented.

a. Naïve Bayes – Baseline Model
Naïve Bayes classification is based on the Bayes Theorem and is a probabilistic approach to machine learning.

$$P(A|B) = (P(B|A)P(A)) / P(B)$$

The above equation represents the probability of A given B has already occurred. "A" is the hypothesis while "B" is the evidence. The most important assumption of this theorem is that the features are all independent from each other. We use the naïve bayes model to classify the data into one of the six categories of toxicity. Consider the following example, we must calculate the probability of a comment being toxic given the comment:
P (comment is toxic| comment content: "I love eating pizza!")
Calculating the probability of comment being toxic based on Bayes Theorem, we can evaluate the following:
*P (comment is toxic | comment content) = P (comment content | toxic) *P(Toxic) / P (comment content)*

The probability of P(Toxic) is proportion of toxic comments that appeared in the training data. Due to naive simplification, we do not need to calculate the probability of finding the content of the comment in the pool of words. Likelihood of a new comment being toxic is calculated by taking product of the word appearing or not appearing in the toxic words pool. Similarly, we can calculate the likelihood of the comment not being toxic. This calculation can be simplified by taking the ratios as series of summations through log scale: *Log(P(message is toxic | comment content) P(message is non-toxic | comment content))*
Smoothing can be done to avoid division by zero.

b. Random Forest Model
Random Forest are one of the most popular ensemble methods which is a non-parametric model which can perform both regression and classification. It works on the idea that a group of weak learners work together and create a stronger learning model. Random forest is a collection many decision trees (called Stumps). Random forest is created in three major steps –

- It creates bootstrapped dataset for each tree. So, every tree is trained on a random sample of data. This provides a good bias-variance compromise.
- Features which are uncorrelated are randomly selected for each tree.
- We repeat the two steps for all the trees.

To construct Random Forest tree, we make three main decisions -

- Method to split the leaf
- Type of predictors used in each leaf
- Methods used to inject randomness

There two ways to split the leafs – axis aligned splits and linear splits. The data is sent to the sub-trees depending on whether it exceeds a specific threshold. The threshold value is selected randomly or by optimizing a function of the data in the leafs.

Randomness can be injected into the tree construction by choice of coefficients for random selection of features or selecting the dimensions to split candidates. Once the model is trained, each tree's prediction for specific data point is given by -

$$f_n^j(x) = \frac{1}{N^e(A_n(x))} \sum_{\substack{Y_i \in A_n(x) \\ I_i = e}} Y_i$$

And average prediction for each tree is –

$$f_n^{(M)}(x) = \frac{1}{M}\sum_{j=1}^{M} f_n^j(x)$$

c. Logistic Regression Model

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

$$f(x) = \frac{1}{1+e^{-(x)}}$$

The cost function represents optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error.

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1-y^{(i)}\right)\log(1-h\theta(x(i)))\right]$$

Ridge regression uses L2 regularization which adds the following penalty term to the OLS equation.

$$+\lambda\sum_{j=0}^{p} w_j^0$$

Lasso regression uses the L1 penalty term and stands for **Least Absolute Shrinkage and Selection Operator**.

$$+\lambda\sum_{j=0}^{p} |w_j|$$

d. Support Vector Machine (SVM) Model

The main objective of support vector machines is to determine an optimal separating hyperplane, which correctly classifies the data points of different classes. The dimensionality of the hyperplane is equal to the (number of input features -1). One can chose many possible separating hyperplanes, but the objective is to find a plane that has the maximum margin. The data points closest to the separating hyperplanes are the support vectors.

The input to SVM is a set of (input, output) training pair samples. The features are denoted as x1, x2…xn, and the output result y. There can be high number of input features xi. The output is given as a set of weights wi for each feature, whose combination predicts the value of y.

The optimization function for SVM is as given below –

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

The given optimization problem has concave quadratic objective function and only linear constraints. We can solve this problem by Lagrange duality.

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum_i^n \alpha_i [y_i(\vec{w}\cdot\vec{x}+b) - 1] \qquad \frac{\partial L}{\partial b} = -\sum_i^n \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial w} = \vec{w} - \sum_i^n \alpha_i y_i x_i = 0$$

$$\vec{w} = \sum_i^n \alpha_i y_i x_i \qquad \sum_i^n \alpha_i y_i = 0$$

$$L = \sum_i^n \alpha_i - \frac{1}{2}\sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

The optimization of maximizing the margin is done to reduce the weights to a few, which correspond to the important features. These features are important in deciding the hyperplane and they correspond to the support vectors because they "support" the hyperplane.

V.VII Performance Metrics

To be able to evaluate the performance of each algorithm, several metrics are defined accordingly, and are discussed briefly in this section.

– Confusion Matrix: It is a very informative performance measures for classification tasks. Ideally, we look for no miss classification in the diagonal items

– Accuracy: This metric measures how many of the comments are labeled correctly. However, in our data set, where most of comments are not toxic, regardless of performance of model, a high accuracy was achieved.

– Precision and Recall: Precision and recall in were designed to measure the model performance in its ability to correctly classify the toxic comments. Precision explains what fraction of toxic classified comments are truly toxic, and Recall measures what fraction of toxic comments are labeled correctly.

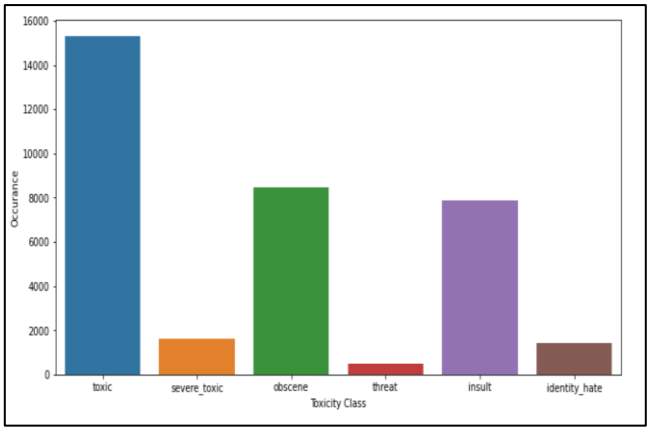
– F Score: This combines both Precisions and Recall together to give a very informative and advanced metric.

- ROC (Receiver Operating Characteristics Curve) and AUC-ROC is a probability curve and AUC represent measure of separability. It tells how well a model is capable of distinguishing between classes.

VI. MODELLING PROCESS

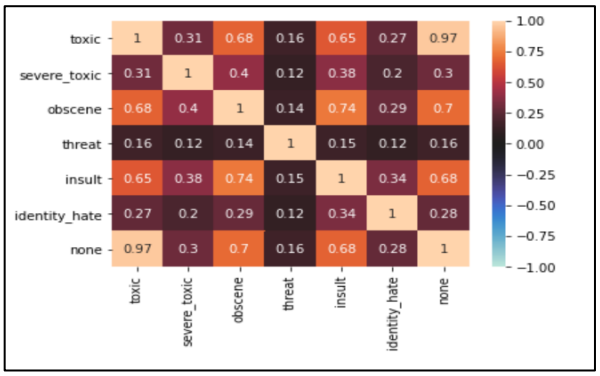
VI.I Data Preprocessing and Exploration

The train dataset consists of 159,571 user comments from Wikipedia and the test data contains 153164 comments. As mentioned before, the file has one input string (comments) and six labels. The distribution and breakdown of the comments across all labels in shown in the graph below



A very important point to note here is that a comment need not belong to just one class label. There is an overlap of labels.

There are 39% of the comments which are only toxic, while the rest have some overlap with each other. 16225 out of 159571 comments, around 10%, are classified as some category of toxic.



For example, there are 15294 toxic comments. (9.58% of all data), out of which,

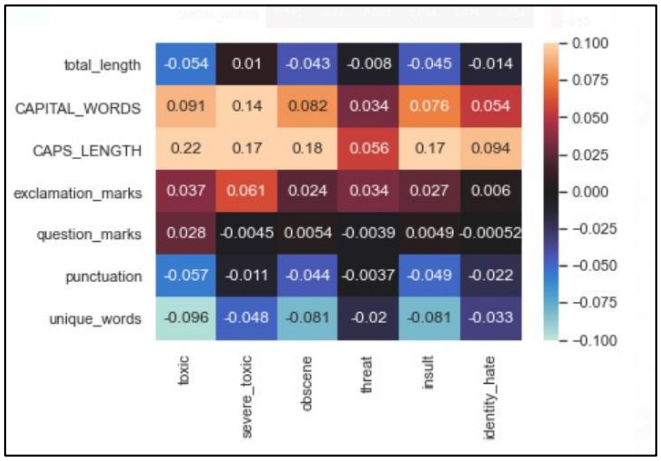
- 1595 are also severe toxic.
- 7926 are also obscene.
- 449 are also threat.
- 7344 are also insult.
- 1302 are also identity hate.

The correlation matrix gives us an insight about the overlapping categories. Some of the key observations that cab made through the correlation matrix are –

1. There is a **positive correlation between insult and obscene** (0.74). This means that comments which are categorized as insults and likely to be obscene as well.
2. Toxic is positively correlated with "obscene" and "insult" (0.68 and 0.65 respectively)
3. Identity hate does not have many overlapping comments with any other category. Since this category has the least overlap, it will be easier to predict this.
4. The "toxic" category completely contains "severe-toxic" which is semantically correct considering the two category names.
5. The 0.31 correlation factor is explained by the fact that 'severe_toxic' represents a small percentage (11.67%) of 'toxic'.

Furthermore, we can consider additional attributes which can help us distinguish between the different toxic labels –

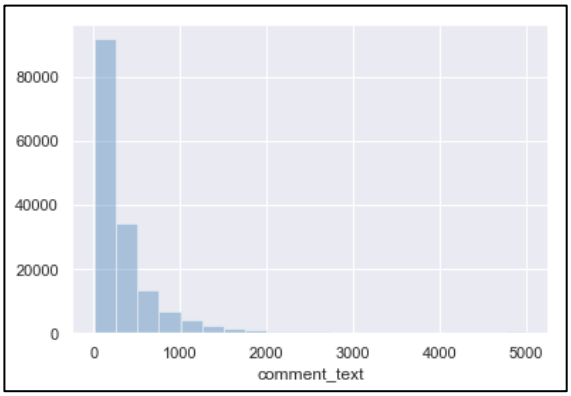
1. Capital Letter Words
2. Exclamation Marks
3. Question Marks
4. Punctuations
5. Repeated Words



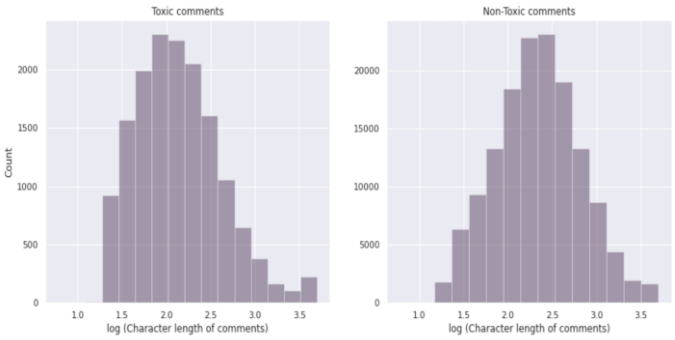
From the above correlation, it is evident that people tend to use capital letters while they write toxic comments. Also, there is a significant use of "!!!" when people try to be toxic.

Checking if Toxic Comments are longer than Non-Toxic Comments – In order to assess the frequency of words size being repeated in two different categories, the overall frequency over number of words was plotted.

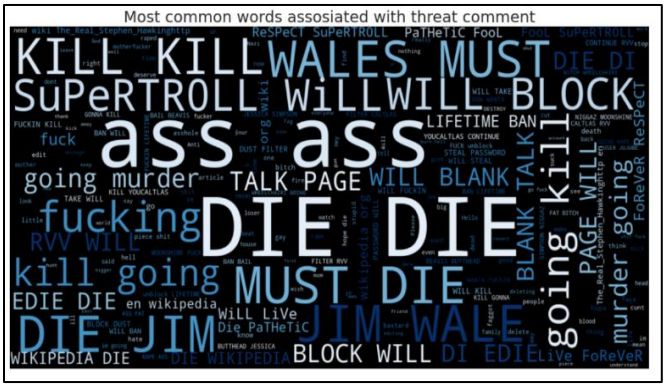
Below is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words.



Interestingly, the toxic group were usually shorter than non-toxic and therefore it helped us to focus more on shorter sentences to improve algorithm performance.
The mean character length of the toxic comments (295) is less than the mean character length of the non-toxic comments (404.54).



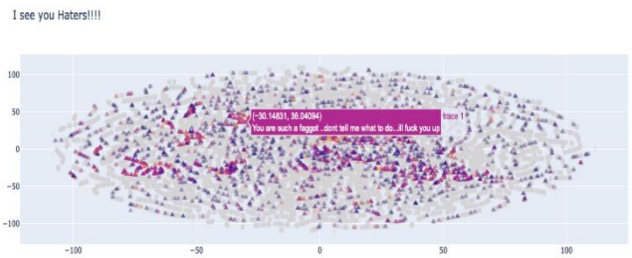
Below is a visualization of most frequently used words under categories – threat by using Word Cloud.



VI.II. Topic Modeling with LDA

Topic modelling is an unsupervised clustering technique that analyses the data to form a cluster of words pertaining to a set of documents. It does not require explicit training or predetermined tags to categorize the text. The different topics created, gives us a sense of what that particular topic is in regard to. In our problem the different topics does not necessarily mean the cluster of similar abusive words but the context in which they are being used and is superficially depicts what people are hating for. These topics may or may not be overlapping. This totally depends on number of topics we choose our text to be classified into.

This is the distribution of comments from the dataset via Plotly and t-SNE. The darker the color, the more labels a comment checks. t-SNE is basically a unsupervised, non-linear technique used for visualizing high dimensional data



VI.III Sentiment Analysis:

Previously, we performed preprocessing on the c omment_text column and got "cleaned_comment_text". This column will be used to perform sentiment analysis.

For TextBlob sentiment analysis, we import the "textblob" library. Two functions – getPolarity and getSentiment (change the name) are defined to return polarity score and sentiment based on the polarity score respectively. The sentiment is decided on the algorithm mentioned above.
Similarly, for Vader we import "vaderSentiment " library. Two functions - getVaderScore and getVaderSentiment are defined to return polarity score and sentiment based on the polarity score respectively. The sentiment is decided on the algorithm mentioned earlier.

Results
Accuracy of TextBlob: = 77.37%
Accuracy of Vader: = 73.58%

Since the accuracy of Vader (73.58%) is less that TextBlob (77.37%), we will continue with TextBlob and make a new training set with rows where Textblob sentiment = 1.

After applying TextBlob sentiment analysis, we get a new training set with 37072 rows. This is will be considered as a new dataset on which we will perform feature vectorization.

cleaned_comment_text	TextBlob_Score	TextBlob_Sentiment	Vader_Score	Vader_Sentiment	TextBlob_Toxic	Vader_Toxic
explanation edits make username hardcore metal...	0.136364	Positive	0.5574	Positive	0	0
aww match background colour seemingly stuck th...	0.25	Positive	0.2263	Positive	0	0
hey man really try edit war guy constantly rem...	0.15	Positive	-0.2415	Negative	0	1
make real suggestion improvement wonder sectio...	0.3	Positive	0.4767	Positive	0	0
sir hero chance remember page	0	Neutral	0.6808	Positive	0	0
congratulation well use tool well talk	0	Neutral	0.7964	Positive	0	0
cocksucker piss around work	0	Neutral	-0.7783	Negative	0	1
vandalism matt shirvington article revert plea...	0	Neutral	-0.3182	Negative	0	1
sorry word nonsense offensive anyway intend wr...	-0.22	Negative	-0.5719	Negative	1	1

VII. EXPERIMENTS AND RESULTS:

In our analysis, after reducing the training set to 37,072 lines, we implemented bag of words (count vector), TFIDF and word2vec methods to convert our string data to numeriacal form. Here, every sentence was transformed into vectors of the same size.

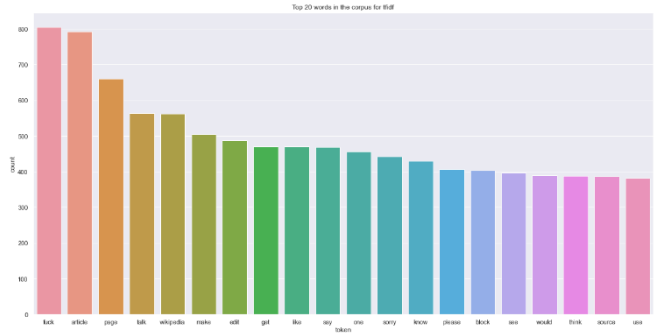
After doing vectorisation and embedding, we have applied 4 models namely Naïve Bayes, Random forest, Logistic regression and Linear SVM to calculate the F1, recall and accuracy on both training and test data. On training data, we have used a method of cross validation, hence we did not divid our training set further into train and validation.

Besides TFIDF, we have also used Count-Vectorizer. However, its performance is not as good as TFIDF. The TFIDF Vectorizer is actually the Count Vectorizer followed by TFIDF Transformer. TFIDF Transformer converts a count matrix to a normalized TF or TF-IDF representation.

We have determined to use the method of TFIDF instead of the raw frequencies of occurrence of a word in each document to scale down the influence of words/tokens that appear very frequently in the given corpus and that are hence empirically less useful than features that occur in small portion of the training dataset. Hence, we can improve accuracy here with TF-IDF

For example: As this corpus consists of data from the Wikipedia's talk page edits, the tokens such as wiki, Wikipedia, edit are very common in use. But for our classification purpose, they do not provide us valuable information and that should probably be the reason why TFIDF worked better with this Wikipedia corpus than Count Vectorizer.

Below is the graph showing top 20 words in the corpus for TF-IDF for exploration purposes.



We have also implemented both of these count vectorizer and TFIDF for unigram and bigram. Below are the results we obtained for both training and test data.

On training dataset – Count Vectorizer:

	Model	Label	Recall	F1	Accuracy
0	MultinomialNB	toxic	0.100658	0.181688	0.800173
1	MultinomialNB	severe_toxic	0.066140	0.117014	0.969195
2	MultinomialNB	obscene	0.133580	0.233339	0.880449
3	MultinomialNB	threat	0.030484	0.054865	0.992879
4	MultinomialNB	insult	0.082602	0.150235	0.879262
5	MultinomialNB	identity_hate	0.122158	0.200675	0.980686
6	LogisticRegression	toxic	0.132459	0.230757	0.805298
7	LogisticRegression	severe_toxic	0.056552	0.102765	0.969600
8	LogisticRegression	obscene	0.137537	0.238770	0.880557
9	LogisticRegression	threat	0.011396	0.020460	0.992744
10	LogisticRegression	insult	0.090527	0.162939	0.879882
11	LogisticRegression	identity_hate	0.086246	0.152442	0.980713
12	LinearSVC	toxic	0.126710	0.221934	0.804138
13	LinearSVC	severe_toxic	0.053936	0.098298	0.969573
14	LinearSVC	obscene	0.135756	0.236216	0.880422
15	LinearSVC	threat	0.015242	0.027356	0.992717
16	LinearSVC	insult	0.084893	0.153756	0.879370
17	LinearSVC	identity_hate	0.083596	0.148447	0.980713
18	RandomForestClassifier	toxic	0.147260	0.248649	0.805675
19	RandomForestClassifier	severe_toxic	0.097460	0.166222	0.968952
20	RandomForestClassifier	obscene	0.154953	0.263602	0.881609
21	RandomForestClassifier	threat	0.015242	0.027619	0.992879
22	RandomForestClassifier	insult	0.172708	0.281366	0.885952
23	RandomForestClassifier	identity_hate	0.164632	0.267887	0.981765

Since random forest does not perform computationally well on our train data, we discarde and tried running the other three models. Below is the result on test data.

	Model	Label	Recall	F1	Accuracy
0	MultinomialNB	toxic	0.910579	0.880687	0.910579
1	MultinomialNB	severe_toxic	0.993654	0.992193	0.993654
2	MultinomialNB	obscene	0.946091	0.927220	0.946091
3	MultinomialNB	threat	0.996624	0.995106	0.996624
4	MultinomialNB	insult	0.948404	0.929168	0.948404
5	MultinomialNB	identity_hate	0.989371	0.986919	0.989371
6	LogisticRegression	toxic	0.909891	0.883026	0.909891
7	LogisticRegression	severe_toxic	0.994060	0.992256	0.994060
8	LogisticRegression	obscene	0.945825	0.927317	0.945825
9	LogisticRegression	threat	0.996624	0.995047	0.996624
10	LogisticRegression	insult	0.948029	0.929160	0.948029
11	LogisticRegression	identity_hate	0.989606	0.986089	0.989606
12	LinearSVC	toxic	0.909531	0.881400	0.909531
13	LinearSVC	severe_toxic	0.994186	0.992272	0.994186
14	LinearSVC	obscene	0.945809	0.927139	0.945809
15	LinearSVC	threat	0.996655	0.995063	0.996655
16	LinearSVC	insult	0.948013	0.929082	0.948013
17	LinearSVC	identity_hate	0.989606	0.986089	0.989606

Similarly, below are the tables for training and test data based on TFIDF vectorization:

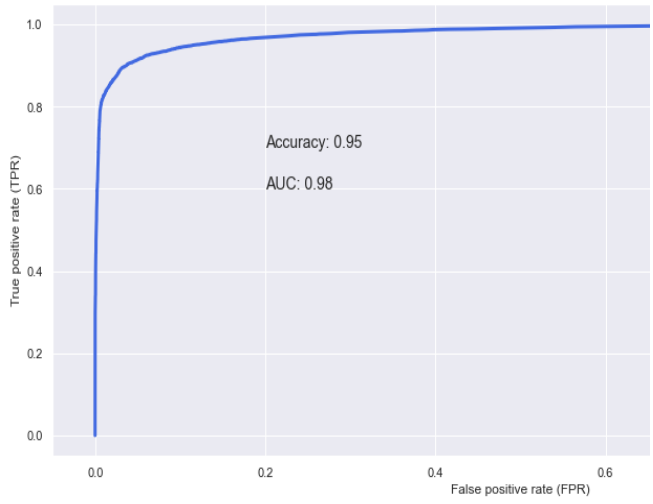
Training dataset:

	Model	Label	Recall	F1	Accuracy
0	MultinomialNB	toxic	0.624631	0.749172	0.907801
1	MultinomialNB	severe_toxic	0.095721	0.165632	0.970247
2	MultinomialNB	obscene	0.595887	0.728201	0.939388
3	MultinomialNB	threat	0.000000	0.000000	0.992933
4	MultinomialNB	insult	0.502713	0.632002	0.924310
5	MultinomialNB	identity_hate	0.013298	0.026111	0.979931
6	LogisticRegression	toxic	0.702543	0.802059	0.923554
7	LogisticRegression	severe_toxic	0.226285	0.316498	0.969870
8	LogisticRegression	obscene	0.710070	0.808807	0.954278
9	LogisticRegression	threat	0.053561	0.095444	0.993122
10	LogisticRegression	insult	0.592203	0.694248	0.932564
11	LogisticRegression	identity_hate	0.140789	0.238739	0.981873
12	LinearSVC	toxic	0.770544	0.821548	0.926198
13	LinearSVC	severe_toxic	0.302006	0.376505	0.969141
14	LinearSVC	obscene	0.774587	0.834165	0.958028
15	LinearSVC	threat	0.206125	0.309771	0.993526
16	LinearSVC	insult	0.639973	0.707640	0.931647
17	LinearSVC	identity_hate	0.308105	0.426667	0.983276

Test dataset:

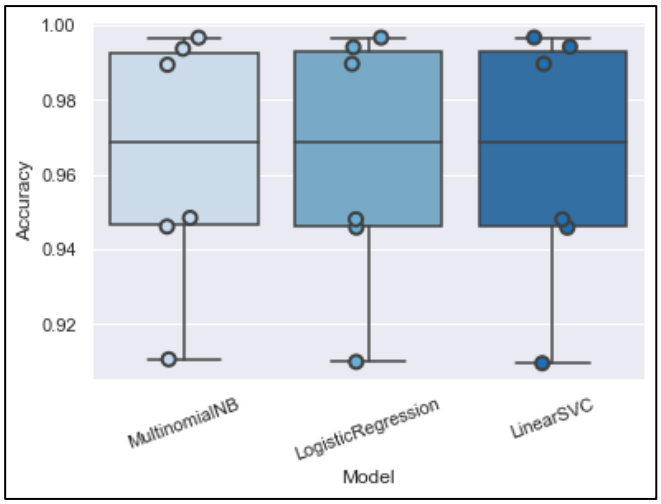
	Model	Label	Recall	F1	Accuracy
0	MultinomialNB	toxic	0.930586	0.930232	0.930586
1	MultinomialNB	severe_toxic	0.994373	0.992761	0.994373
2	MultinomialNB	obscene	0.961315	0.958271	0.961315
3	MultinomialNB	threat	0.996702	0.995056	0.996702
4	MultinomialNB	insult	0.958142	0.954237	0.958142
5	MultinomialNB	identity_hate	0.989027	0.983723	0.989027
6	LogisticRegression	toxic	0.934899	0.935125	0.934899
7	LogisticRegression	severe_toxic	0.993623	0.992914	0.993623
8	LogisticRegression	obscene	0.965973	0.963653	0.965973
9	LogisticRegression	threat	0.996311	0.995474	0.996311
10	LogisticRegression	insult	0.963144	0.960182	0.963144
11	LogisticRegression	identity_hate	0.990481	0.988348	0.990481
12	LinearSVC	toxic	0.920457	0.925927	0.920457
13	LinearSVC	severe_toxic	0.992669	0.992614	0.992669
14	LinearSVC	obscene	0.962159	0.962496	0.962159
15	LinearSVC	threat	0.996171	0.995860	0.996171
16	LinearSVC	insult	0.955172	0.955380	0.955172
17	LinearSVC	identity_hate	0.989403	0.989007	0.989403

After performing vectorization, we have also tried word embedding on our new train dataset and found the below results for our test dataset. Below is the ROC curve with accuracy for word2vec method on toxic comments:



Summary:

Based on the cross validation above, we noticed both the linear SVC model and Logistic Regression models perform good. Multinomial Naive Baye model is used as our baseline and it does not perform well for the threat label and identity_hate labels as seen in the table above. This is because these two labels have the least number of observations.



On comparing the accuracies on the test data for toxic label across all the three feature vectorizations as shown in the table below, we found that Word2Vec has the highest accuracy and performs better as compared to other two features. Also, the linear SVM model has the best performance across all models.

Feature vectorisation	Models		
	Naïve Bayes	Logistic Regression	Linear SVM
Count Vectorization	91.057%	90.989%	90.953%
TF-IDF	93.058%	93.489%	92.045%
Word2vec	93.280%	94.691%	95.010%

VIII. CONCLUSION AND FUTURE WORK

It is a well-known fact that harmful or toxic comments in the social media space have many negative impacts to our society. The capacity to identify comments as toxic and sub classification levels quickly and accurately could provide many social benefits while mitigating its harm. Also, our research has shown the potential of readily available algorithms such as SVM and Logistic Regression with TF-IDF and Word2Vec to be employed in such a way to address this challenge.

In our specific study, it is clearly demonstrated that the best model with respect to the multi-label classification for our dataset was Linear SVM – Word2Vec. It has outperformed all the other models.

Additionally, the followings are some of the suggested studies that should be considered as future work in this area:

a). We would like to implement sentiment analysis by NRC dictionary which can further quantify our sentiments and help in proper toxic comment classification.

b). We also recommend a proposal to improve the NLP classifiers by using other algorithms such as Convolutional Neural Networks (CNN) and test their performances against different LSTM models.

c). We suggest using kernel SVM for text processing and text classification. It requires a grid search for hyper-parameter tuning to get the best results.

d). We can further deep dive into feature engineering by implementing spelling corrector and various n-gram models to have proper context of the comments to avoid misclassification.

IX. REFERENCES

[1] L.-C. Yu, J. Wang, K. R. Lai, and X. Zhang, “Refining Word Embeddings for Sentiment Analysis,” Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5:135–146, 2017.

[3] Van Aken, Betty Risch, Julian Krestel, Ralf Löser, Alexander. (2018). Challenges for Toxic Comment Classification: An In-Depth Error Analysis. 10.18653/v1/W18-5105.

[4] "GloVe: Global Vectors for Word Representation". Jeffrey Pennington, Richard Socher, Christopher D. Manning. Computer Science Department, Stanford University. [Online]. Available: https://nlp.stanford.edu/pubs/glove.pdf

[5] Kevin Khieu and Neha Narwal: ”Detecting and Classifying Toxic Comments”, https://web.stanford.edu/class/cs224n/reports/6837517.pdf

[6] "Linguistic Regularities in Continuous Space Word Representations" Omas Mikolov, Wen-tau Yih, Geoffrey Zweig. Microsoft Research. [Online]. Available: https://www.aclweb.org/anthology/N13-1090. [Accessed: 15-Dec-2018]

[7] Theodora Chu, Kylie Jue and Max Wang: ”Comment Abuse Classification with Deep Learning”, https://web.stanford.edu/class/cs224n/reports/2762092.pdf

[8] ManavKohli, Emily Kuehler and John Palowitch: ”Paying attention to toxic comments online”, https://web.stanford.edu/class/cs224n/reports/6856482.pdf

[9] Leonie Rosner, Stephan Winter, and Nicole C Kramer. Dangerous minds? effects of uncivil online comments on aggressive cognitions, emotions, and behavior. Computers in Human Behavior, 58:461–470, 2016.

[10] Nadbor, “DS lore,” Text Classification With Word2Vec - DS lore. [Online]. Available: https://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/. [Accessed: 15-Dec-2018].

[11] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[12] https://scholar.smu.edu/cgi/viewcontent.cgi?article=1134&context=datasciencereview