Project-13 : Toxic Comment Classification
CSL2050 : Pattern Recognition and Machine Learning
Major Project Report

Manish
B21CS044
manish.2@iitj.ac.in

Samaksh Verma
B21AI037
verma.41@iitj.ac.in

Navneet Kumar
B21CS050
kumar.312@iitj.ac.in

Github Link
Website Link
API Link

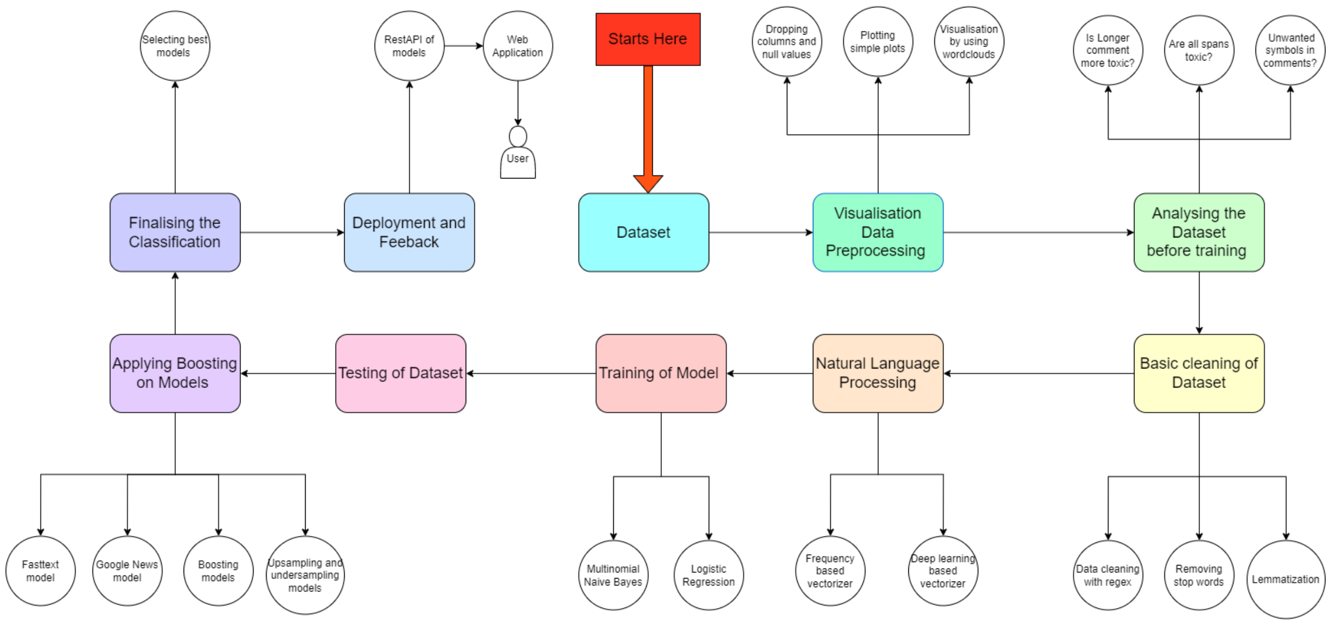
Indian Institute of Technology Jodhpur
NH 62, Surpura Bypass Rd, Karwar, Rajasthan 342030
March 2023

Abstract

The project focused on classifying toxic comments using Natural Language Processing techniques. The data was preprocessed using Scikit-learn library functions, along with regular expressions, NLTK, and SpaCy for data cleaning. The data was then vectorized using TF-IDF and Word2Vec vectorizers. The project used Multinomial Naive Bayes and Logistic Regression models along with other built-in models to classify the comments. The experiments were conducted to show the results of different models and vectorizers, and the best performing model was identified. Overall, the project demonstrated the effectiveness of using NLP techniques to classify toxic comments.

We would like to express my sincere gratitude to Professor Richa Singh for her guidance and supervision throughout the course of Pattern Recognition and Machine Learning. Her valuable insights and feedback were instrumental in shaping this project on toxic comment classification. Her dedication to teaching and commitment to providing a rich learning experience has been a source of inspiration for all of us. We are thankful for her support and encouragement throughout this project, which has helped us enhance our knowledge and skills in the field of Machine Learning.

Machine Learning Pipeline:


Selecting best models
RestAPI of models
Web Application
User
Starts Here
Dropping columns and null values
Plotting simple plots
Visualisation by using wordclouds
Is Longer comment more toxic?
Are all spans toxic?
Unwanted symbols in comments?
Finalising the Classification
Deployment and Feeback
Dataset
Visualisation Data Preprocessing
Analysing the Dataset before training
Applying Boosting on Models
Testing of Dataset
Training of Model
Natural Language Processing
Basic cleaning of Dataset
Fasttext model
Google News model
Boosting models
Upsampling and undersampling models
Multinomial Naive Bayes
Logistic Regression
Frequency based vectorizer
Deep learning based vectorizer
Data cleaning with regex
Removing stop words
Lemmatization

1. Introduction

We were given a large dataset of Wikipedia comments, which had been manually labeled by human raters for different types of toxicity such as toxic, severe-toxic, obscene, threat, insult, and identity-hate. Our task was to develop a model that could predict the probability of each type of toxicity for a given comment. It was a challenging task that required us to use various machine learning techniques and tools to build an accurate predictive model. Throughout this project, we worked collaboratively to preprocess the data, develop appropriate features, and train and evaluate different models to achieve the best possible results. The project enabled us to enhance our skills in machine learning and natural language processing, and we are grateful for the opportunity to work on such a challenging and rewarding project.

2. Data Preprocessing and Viusalization

2.1. Data Preprocessing

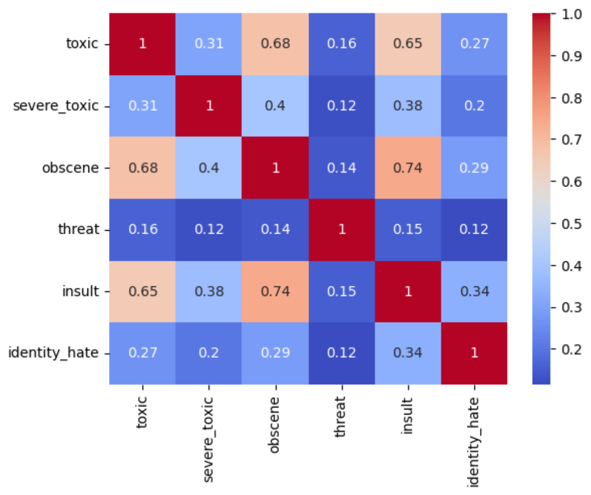
Following are the steps of preprocessing of the dataset:

- In data preprocessing, we dropped unnecessary columns such as ’Unnamed’ and ’Id’.
- We handled the null values in the dataset.
- Rows with labels as -1 were dropped from both the training and testing datasets since they couldn’t be used for testing.

2.2. Data Visualization

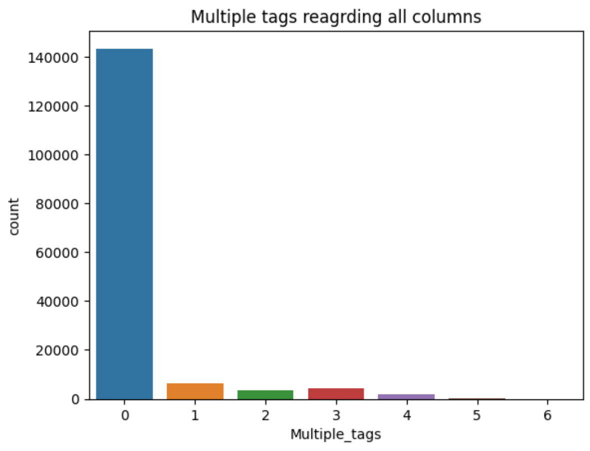
- We plotted the correlation matrix for the labels to identify any patterns or relationships among them.
- The distribution of each label was also plotted to get an understanding of the proportion of toxic comments in the dataset.
- We also plotted the distribution of multiple labeled data points to understand the frequency of comments that had multiple types of toxicity.
- In addition, we generated a word cloud of the density of abusive words to gain further insights into the language used in the comments. However, we have decided not to include these plots due to their sensitive nature.
- All of these plots and visualizations are included in our Jupyter notebook for reference and further analysis.

2.3. Distribution Plots

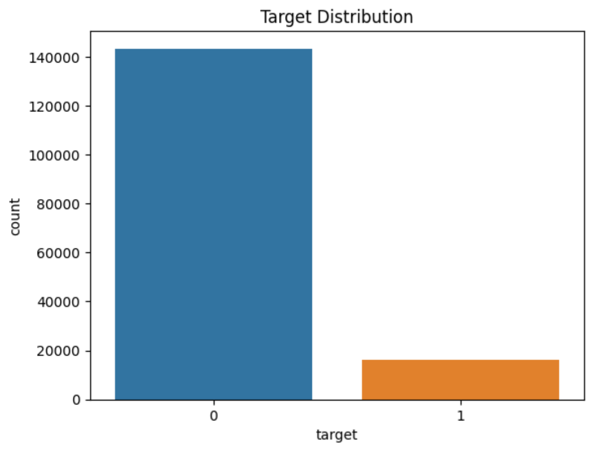


	toxic	severe_toxic	obscene	threat	insult	identity_hate
toxic	1	0.31	0.68	0.16	0.65	0.27
severe_toxic	0.31	1	0.4	0.12	0.38	0.2
obscene	0.68	0.4	1	0.14	0.74	0.29
threat	0.16	0.12	0.14	1	0.15	0.12
insult	0.65	0.38	0.74	0.15	1	0.34
identity_hate	0.27	0.2	0.29	0.12	0.34	1

(a) Correlation Plot between Labels



(b) Multiple Labeled Comments Distribution



(c) Overall toxic comments

3. Analyzing the Dataset

- In analyzing factors that contribute to toxicity, we computed various features such as word count, frequency of repeated words, comment length, and punctuation count.
- We also calculated the number of unique words in each comment to identify any distinct vocabulary used in toxic comments.
- We plotted these features to identify any trends or patterns that may help improve the performance of our model in predicting toxic comments.

After performing these steps two type of questions came to our mind.

3.1. Are longer comments more toxic?

- During our analysis, we explored the question of whether longer comments are more toxic than shorter ones.
- To investigate this, we plotted a violin plot of the number of sentences in each comment, as well as a plot of the number of words in each comment.
- These visualizations helped us understand the distribution of comment lengths and any potential correlation between comment length and toxicity.

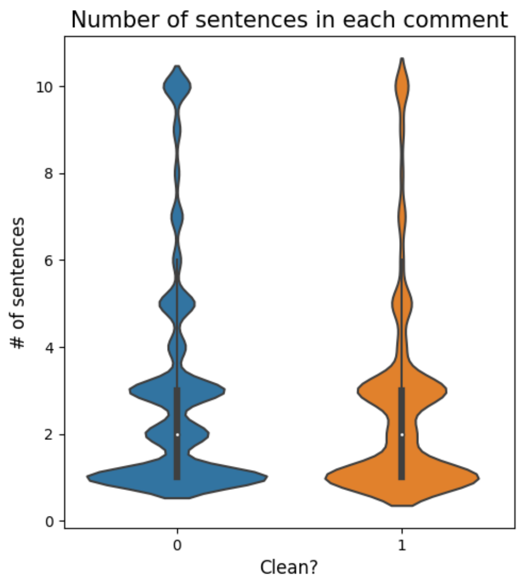
Result :

As we can see from the plots also in general the longer comments are not toxic. The density of the no of sentences and no of words in toxic comments is less than the normal comments.

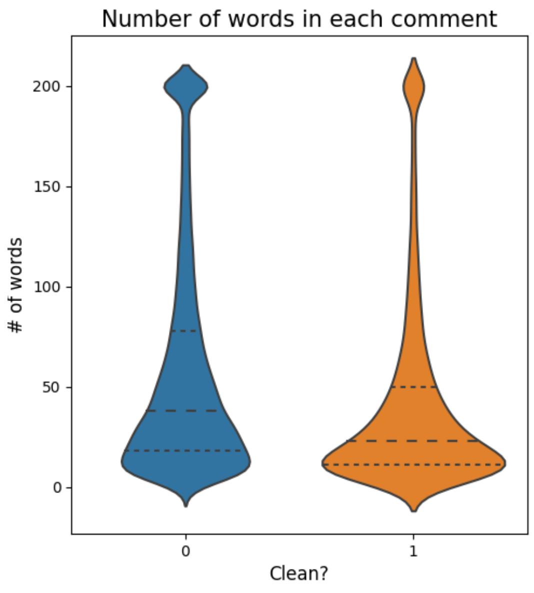
So, we can't generalise this that longer comments are more toxic, number of comments that are long are more clean than toxic.

Chart description:

Violin plot is an alternative to the traditional box plot. The inner markings show the percentiles while the width of the "violin" shows the volume of comments at that level/instance.



(a) Number of sentences in comments distribution



(b) Number of words in comments distribution

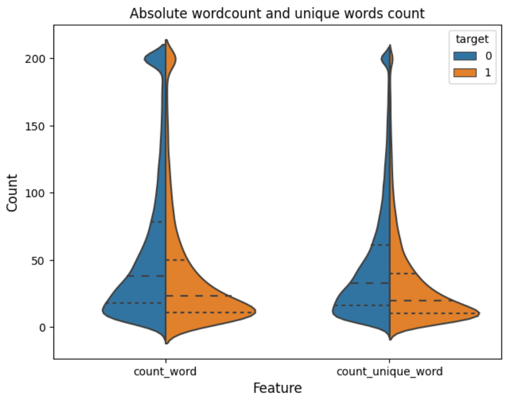
3.2. Are Spam comments more toxic?

- To analyze this, we plotted the absolute number of words and absolute word counts in comments.
- We then created a violin plot of these features to visualize the distribution of spam comments and normal comments in terms of their word count and absolute word count.
- These visualizations helped us understand the relationship between spam comments and toxicity, and provided insights into the factors that contribute to the toxicity of comments.

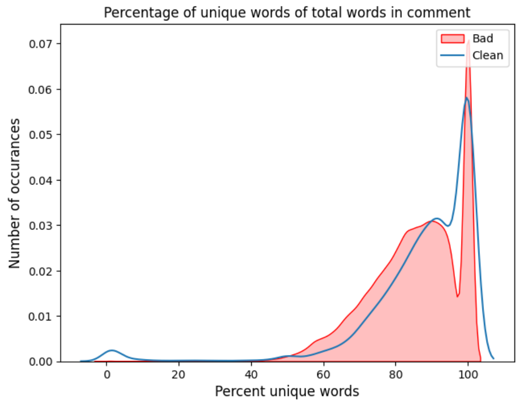
Result :

As we can see from the plots also in general the longer comments are not toxic. The density of the no of sentences and no of words in toxic comments is less than the normal comments.

So, we can't generalise this that longer comments are more toxic, number of comments that are long are more clean than toxic.



(a) Absolute and Unique words count



(b) Occurrence of unique words

From the above results we can see that there is significant impact of absolute word count in comments that's why spam comments are generally toxic. From another plot we can say that there is nearly less than 1 percentage comments that are spam and are clean.

4. Basic Cleaning of Dataset

4.1. Cleaning data with regex

- In the basic cleaning phase of our data preprocessing, we used regex to clean the comments by removing numbers, punctuations, and several other symbols that were not useful for our analysis.
- By removing these elements, we were able to reduce noise in the dataset and extract the most relevant information for our model to accurately predict the toxicity of comments.
- This step was essential in ensuring the quality and accuracy of our dataset and improving the overall performance of our model.

4.2. Removing Unnecessary words

- In the process of removing unnecessary words from our dataset, we used the NLTK library to remove stop words.
- We also replaced abbreviated words with their initial forms and converted all words to lowercase, to ensure consistency and improve the accuracy of our model.
- This step helped us to streamline our dataset and improve the efficiency of our model by removing irrelevant or redundant information from the text data.

Stop words are commonly used words in a language (such as "the", "is", "and", etc.) that do not carry significant meaning and can be safely removed without affecting the overall context of the text.

4.3. Lemmetization

- Another important step in data preprocessing was lemmatization, where we used the NLTK library to convert words to their base or dictionary form.
- This process involves reducing words to their root form, which helps to group together variations of the same word and reduces the overall dimensionality of the dataset.
- For example, the word "running" would be converted to "run" through lemmatization.
- By using lemmatization, we were able to further refine our dataset and reduce noise by removing variations of words that carried the same meaning.
- This step ultimately improved the accuracy and efficiency of our model by reducing the complexity of the text data.

5. NLP(Vectorizing the Comments)

5.1. Frequency Based Vectorizer(TFIDF)

- After data preprocessing, we used the TF-IDF vectorizer to convert the text data into numerical vectors.
- TF-IDF stands for ”Term Frequency-Inverse Document Frequency” and is a common technique used in natural language processing to quantify the relevance of words in a document.
- The TF-IDF vectorizer assigns a weight to each word in a document, based on how often it appears in the document and how frequently it appears across all documents in the corpus.
- This allows the model to assign higher weights to words that are more unique and meaningful within a given document, while downweighting common or unimportant words.
- By converting the text data into numerical vectors, the TF-IDF vectorizer enabled us to use machine learning algorithms to classify toxic comments based on their content.

5.1.1 Unigram TFIDF

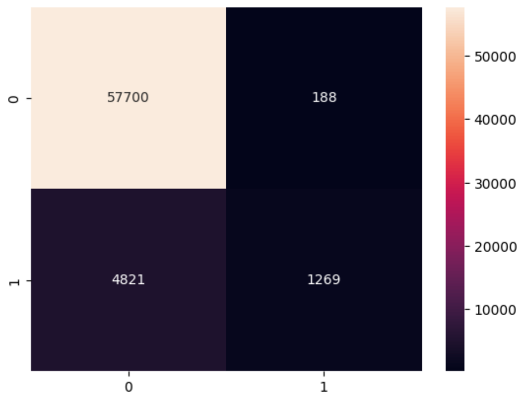
Unigram TF-IDF vectorization involves considering each individual word in the text as a separate feature, and computing its TF-IDF weight based on its frequency in the document and across the corpus.

5.1.2 Bigram TFIDF

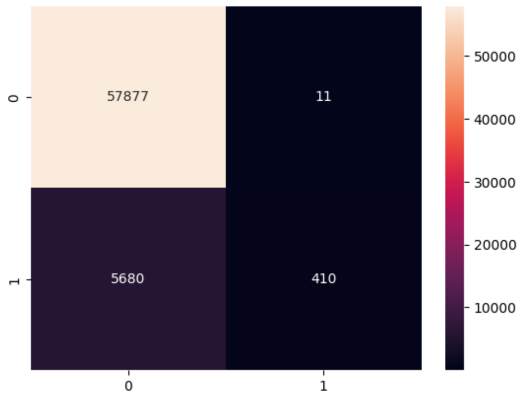
Bigram TF-IDF vectorization, on the other hand, involves considering pairs of adjacent words in the text as features, and computing their combined TF-IDF weight.

5.1.3 Difference in the results of Unigram and Bigram

From the plots also we can see that Bigram is not performing better than Unigram, it is because there is not significant use of clubbing words together and then vectorizing it.
So, we haven’t performed all operations for both type of vectorizing.



(a) Unigram results, Accuracy-Toxic : 0.9217074619400419



(b) Bigrams results, Accuracy-Toxic : 0.9110475475944856

<u>Assumption</u> :

Since a significant proportion of toxic comments are spam, TF-IDF vectorization would be effective in classifying them. This is because TF-IDF assigns higher weights to words that are rare in the corpus, but frequent in a given document. Since spam comments often contain words or phrases that are uncommon in the rest of the corpus, they are likely to receive high TF-IDF weights and be more easily distinguishable from normal comments.

<u>Result</u> :

As we have assumed TF-IDF vectorizer is performing significantly. Results are shown below in Training of Models section.

5.2. Deep Learning Based Vectorizer(Word2vec)

- Word2vec is a popular technique for generating word embeddings, which are dense vector representations of words that capture their meaning in a mathematical form.
- The word2vec algorithm is based on predicting a target word given a context of surrounding words, or vice versa, by training a neural network on a large corpus of text.
- During training, word embeddings are learned by adjusting the weights of the neural network so that the predicted target word is as close as possible to the actual target word in the training data.
- Once trained, each word in the vocabulary is represented as a dense vector of fixed dimensionality, where similar words have similar vector representations in terms of cosine similarity.
- Word2vec comes in two flavors: CBOW and Skip-gram, which respectively predict the target word given a context or the context given a target word.
- CBOW is faster and works better for smaller datasets, while Skip-gram is generally considered to perform better on larger datasets. Here we have used CBOW.

$$\vec{king} + \vec{boy} - \vec{girl} = \vec{queen}$$

Note that this assumes that $\vec{king}$, $\vec{boy}$, $\vec{girl}$, and $\vec{queen}$ are pre-trained word2vec vectors. This is one of the popular example shared by Google News Model(Word2vec Model).

For both type of vectorizing techniques the results are comparable ,we have showed the results below in Training Models section.

6. Training and Testing of Models

6.1. Training on TF-IDF Vectorized data

- Multinomial Naive Bayes is suitable for binary classification problems by treating one class as the "positive" class and the other class as the "negative" class.
- Logistic Regression is a natural choice for binary classification problems as it models the probability of the positive class given the input features.
- Both Multinomial Naive Bayes and Logistic Regression are simple and computationally efficient models that can handle large datasets.
- These models are also interpretable, which can help in understanding how they make predictions and identifying which features are most important for classification.
- The dataset being a binary labeling dataset, these models are well-suited for such classification tasks.

6.1.1 Results of TF-IDF Trained Models

Here are the accuracies for the same:
Toxic:
Multinomial NB: 0.922, Logistic Regression: 0.937

Severe Toxic:
Multinomial NB: 0.994, Logistic Regression: 0.993

Insult:
Multinomial NB: 0.949, Logistic Regression: 0.964

Obscene:
Multinomial NB: 0.948, Logistic Regression: 0.967

Threat:
Multinomial NB: 0.989, Logistic Regression: 0.990

Identity Hate:
Multinomial NB: 0.997, Logistic Regression: 0.997

6.1.2 Probable reasons of better performance of Logistic regression

- Logistic regression is a discriminative model while Multinomial Naive Bayes is a generative model.
- Discriminative models tend to perform better when the data is high-dimensional and sparse, as is the case with TF-IDF vectorization.
- Logistic regression can capture complex non-linear relationships between features and the target variable.
- Multinomial Naive Bayes assumes that the features are conditionally independent given the target variable.

6.2. Training on Word2vec Vectorized data

- After vectorizing data with Word2vec, we have used logistic regression to classify the data.
- Logistic regression is a discriminative model that can capture complex non-linear relationships between features and the target variable.

- Word2vec data is not categorical, which makes it more suitable for training discriminative models like logistic regression.

- Logistic regression has been shown to perform well on high-dimensional and sparse data, which is often the case with Word2vec vectorization.

6.2.1 Results of Word2vec Trained Models

Here are the accuracies for the same:
Toxic:
Logistic Regression: 0.932

Severe Toxic:
Logistic Regression: 0.992

Insult:
Logistic Regression: 0.959

Obscene:
Logistic Regression: 0.960

Threat:
Logistic Regression: 0.996

Identity Hate:
Logistic Regression: 0.988

After, performing both vectorizer we can say that both Word2vec and TF-IDF are performing similar. Since, we have preprocessed intially well ,So it is able to vectorize the data perfectly in both case. But from small margin TF-IDF is performing better.

7. Applying Boosting Models

7.1. Fastext Model

- FastText is an open-source, free library from Facebook AI Research (FAIR) for learning word embeddings and word classifications.

- It allows creating unsupervised learning or supervised learning algorithm for obtaining vector representations for words.

- One of the advantages of FastText over Word2Vec is that it can handle rare words better. Since the model generates embeddings for each n-gram, it can still produce a meaningful vector representation even for words that are not frequently observed in the training data. This is particularly useful in applications where rare or out-of-vocabulary words are common.

- The models built through deep neural networks can be slow to train and test. These methods use a linear classifier to train the model. FastText supports both CBOW and Skip-gram models

- In this text and labels are represented as vectors. We find vector representations such that text and it's associated labels have similar vectors. In simple words, the vector corresponding to the text is closer to its corresponding label. We have used fastext supervised model to predict the toxicity and have got the overall accuracy of **0.9211916596329989**.

Since these models were not performing as well as other models and experiments that we have performed hence we are not continuing with this model.

7.2. Adaboost

- AdaBoost (Adaptive Boosting) is a machine learning technique used as an ensemble method. It is mainly used for classification. The most common estimator used with AdaBoost is decision trees with one level which means Decision trees with only split.

- In AdaBoost, these decision trees are known as Decision Stumps. It makes use of weighted errors to build a strong classifier from a series of weak classifiers.

- The algorithm works by iteratively training a sequence of weak classifiers on reweighted versions of the data. In each iteration, the weights of misclassified samples are increased so that subsequent classifiers focus more on difficult cases.

7.2.1 Adaboost on Logistic Regression

We performed Adaboost on Logistic regression but it failed to give good accuracies and gave only about **90%** accuracy so we discontinued the use of Adaboost model.

7.3. Using Google News model

- The Google News Vectorizer model is a natural language processing (NLP) model that is used to transform text data into a numerical format that can be easily processed by machine learning algorithms. This model is specifically designed to generate word embeddings, which are dense vector representations of words that capture their semantic and syntactic relationships.

- This is similar to Word2Vec model that we have applied above

- The word vectors that we obtain from this model are typically have 300 dimensions and these word vectors are more dense

7.3.1 Applying Logistic Regression on Vectorised Data

- We have applied logistic regression on the vectorised data and then have obtained the following accuracies:

Toxic 0.904811	Insult 0.9464

Since these did not perform well we did not continue with their results.

7.4. Upsampling and Undersampling of the Dataset

We have analyzed the performance of the classifiers on various sampling on the dataset.

7.4.1 Assumption :

As we know that the data is mostly fulled of the clean comments. So, it is assumed that on increasing toxic comments the recall for the class of toxic comments is going to increase It will be classifying better toxic comments but precision will decrease it will be labelling the clean comments also as TOXIC comments.

7.4.2 Results of different types sampling

Here is the table corresponding whose ratio is considered to selecting the datapoints in the sampled dataset.

Samples	**1000**	**5000**	**10000**	**20000**	**40000**	**70000**

Here are the accuracy and recall for same type of sampling.

Accuracy	**0.922**	**0.939**	**0.941**	**0.934**	**0.919**	**0.904**

Recall toxic-1	**0.21**	**0.48**	**0.61**	**0.73**	**0.81**	**0.86**

Result : As we have noticed that as we increase the sampling of the toxic comments the recall for the class Toxic increases significantly.

But , the accuracies increase upto a middle value then it starts overfitting the model on Toxic Comments and overall acuuracies starts to decrease. And also precision also decreases for the class Toxic.

8. Best Results

Best results that we obtained were after performing **TFIDF vectorization** method and then further on performing **LogisticRegression** on it. Below are the accuracies of each class.
These results demonstrate the effectiveness of the TFIDF vectorization method and LogisticRegression in accurately predicting the classes of the data. We are confident that these results can be further improved by fine-tuning the model and exploring other techniques. Overall, these results are a promising step towards developing a robust and accurate machine learning model for the given task.
Please turn to the next page to view the best results in detail.

Toxic : 0.936	Severe Toxic : 0.993	Insult : 0.964
Obscene 0.966	Identity Hate : 0.990	Threat : 0.996

9. Deployment of the model

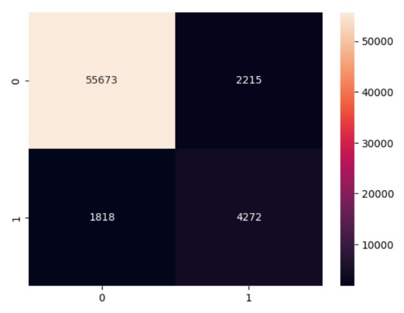
- Developed an API for classification models using REST API in Python
- Developed a website for the backend using HTML, JavaScript, and Node.js
- Used Tailwind CSS for website design
- Deployed the website on PythonAnywhere platform
- Website fetches output from API and displays result
- To use our api you can put your text after https://verma41.pythonanywhere.com/prediction/ ,for example comment =”we are awesome” we have to use https://verma41.pythonanywhere.com/prediction/we are awesome/ that will fetch you a json object with the corresponding predicted classes ,its easy to use

10. References

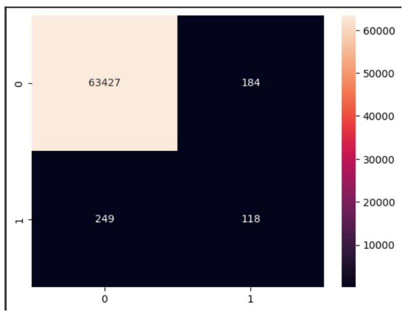
- Jigsaw Toxic Comment Classification Challenge: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
- GloVe: Global Vectors for Word Representation: https://nlp.stanford.edu/projects/glove/
- UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/index.php
- Convolutional Neural Networks for Sentence Classification: https://arxiv.org/abs/1408.5882

- Natural Language Processing with Python:
  https://www.nltk.org/book/

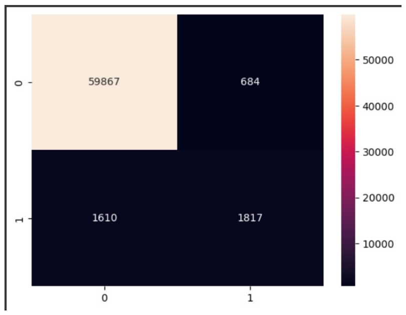
Please find the best results on the next page.



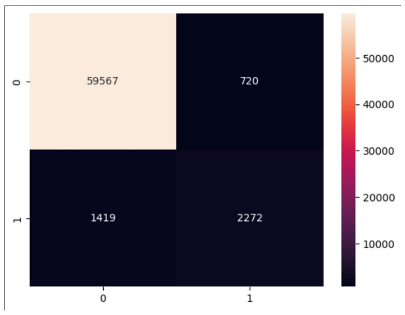
(a) Toxic



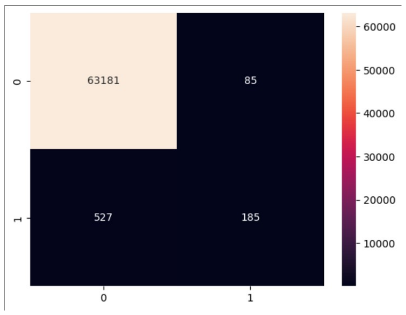
(b) Severe Toxic



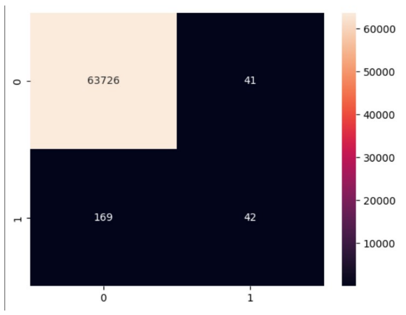
(c) Insult



(d) Obscene

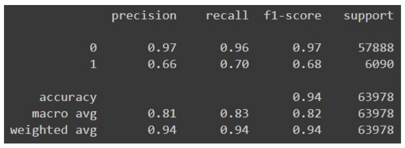


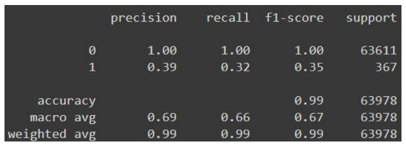
(e) Identity Hate

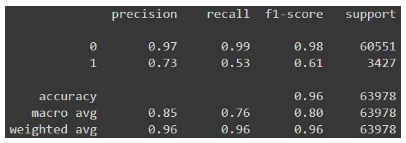


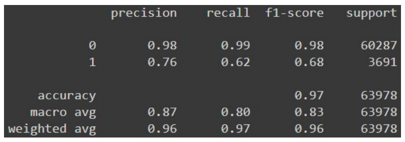
(f) Threat

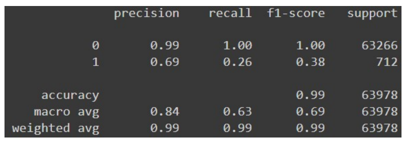
Figure 5. Confusion matrix of Respective classes


```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97     57888
           1       0.66      0.70      0.68      6090

    accuracy                           0.94     63978
   macro avg       0.81      0.83      0.82     63978
weighted avg       0.94      0.94      0.94     63978
```
(a) Toxic


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     63611
           1       0.39      0.32      0.35       367

    accuracy                           0.99     63978
   macro avg       0.69      0.66      0.67     63978
weighted avg       0.99      0.99      0.99     63978
```
(b) Severe Toxic


```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     60551
           1       0.73      0.53      0.61      3427

    accuracy                           0.96     63978
   macro avg       0.85      0.76      0.80     63978
weighted avg       0.96      0.96      0.96     63978
```
(c) Insult


```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     60287
           1       0.76      0.62      0.68      3691

    accuracy                           0.97     63978
   macro avg       0.87      0.80      0.83     63978
weighted avg       0.96      0.97      0.96     63978
```
(d) Obscene


```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     63266
           1       0.69      0.26      0.38       712

    accuracy                           0.99     63978
   macro avg       0.84      0.63      0.69     63978
weighted avg       0.99      0.99      0.99     63978
```
(e) Identity Hate

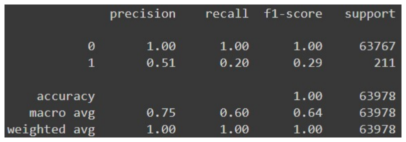

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     63767
           1       0.51      0.20      0.29       211

    accuracy                           1.00     63978
   macro avg       0.75      0.60      0.64     63978
weighted avg       1.00      1.00      1.00     63978
```
(f) Threat

Figure 6. Classification Report of Respective classes