Assignment 5: Final Project

Group: 5

By:

- **Ahmed Nasser**
- **Abdelrahman Ali**
- **Esraa Fayad**
- **Aya Metwally**

Table of contents:

Contents

I. Introduction

I.I Problem Formulation/Explanation:

The goal of this problem is to develop a machine learning model that can accurately predict the presence and types of toxicity in online comments. Given a dataset of comments labeled with binary indicators for various types of toxicity (toxic, severe_toxic, obscene, threat, insult, identity_hate), the objective is to build a classification model that can generalize to new, unseen comments.

The model needs to process the text of each comment and predict the likelihood of each toxicity type present in the comment. This is a multi-label classification problem, as a comment can exhibit multiple types of toxicity simultaneously.

II. Data Preparation

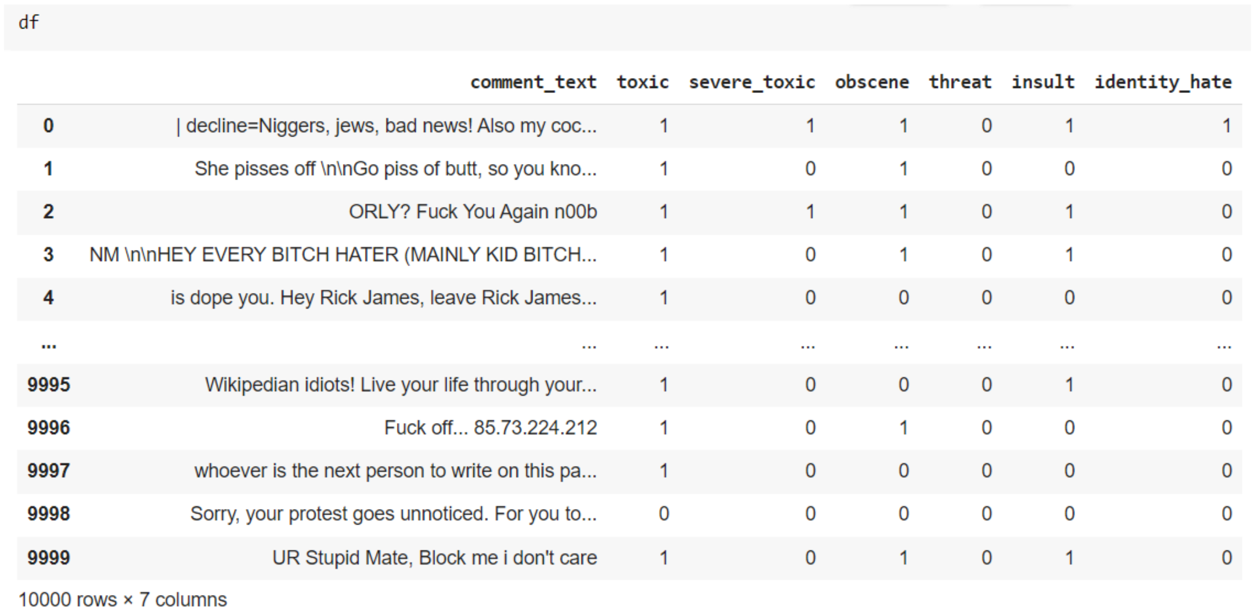
II.I Data understanding:

We want to understand our data to make the most use of all information we go to classify Toxic comments so first we:

Installing required packages

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split


import string
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objects as go
from sklearn.feature_extraction.text import CountVectorizer

import matplotlib.pyplot as plt
import seaborn as sns
import re
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.metrics import (accuracy_score, roc_auc_score, f1_score, hamming_loss,
                             classification_report, multilabel_confusion_matrix)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import ClassifierChain
from sklearn.dummy import DummyClassifier


import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

Then we visualize our dataset to get more information about columns and target



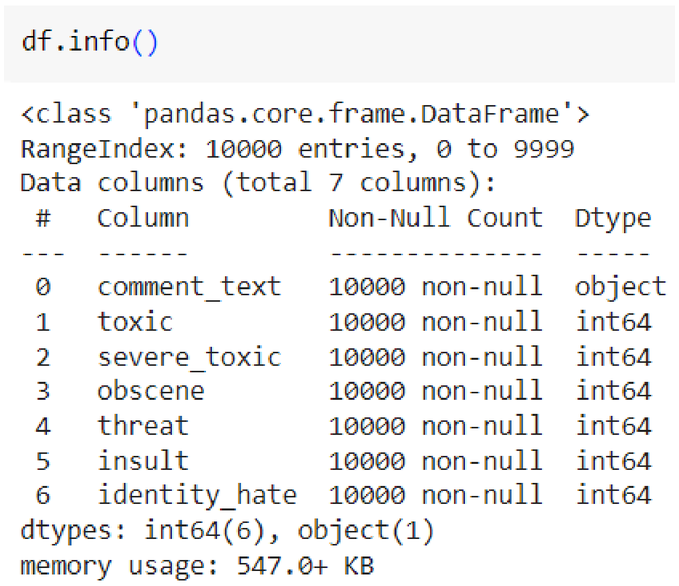
```
df
```

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	| decline=Niggers, jews, bad news! Also my coc...	1	1	1	0	1	1
1	She pisses off \n\nGo piss of butt, so you kno...	1	0	1	0	0	0
2	ORLY? Fuck You Again n00b	1	1	1	0	1	0
3	NM \n\nHEY EVERY BITCH HATER (MAINLY KID BITCH...	1	0	1	0	1	0
4	is dope you. Hey Rick James, leave Rick James...	1	0	0	0	0	0
...	...	...	...	...	...	...	...
9995	Wikipedian idiots! Live your life through your...	1	0	0	0	1	0
9996	Fuck off... 85.73.224.212	1	0	1	0	0	0
9997	whoever is the next person to write on this pa...	1	0	0	0	0	0
9998	Sorry, your protest goes unnoticed. For you to...	0	0	0	0	0	0
9999	UR Stupid Mate, Block me i don't care	1	0	1	0	1	0

10000 rows × 7 columns

As we see from figure above, this dataset related to comments on a platform, and it contains various columns including 6 **Target columns** "id," "comment_text," "toxic," "severe_toxic," "obscene," "threat," "insult," and "identity_hate." These columns associated with toxicity.

The target variables would be the labels used for training and evaluating the model.



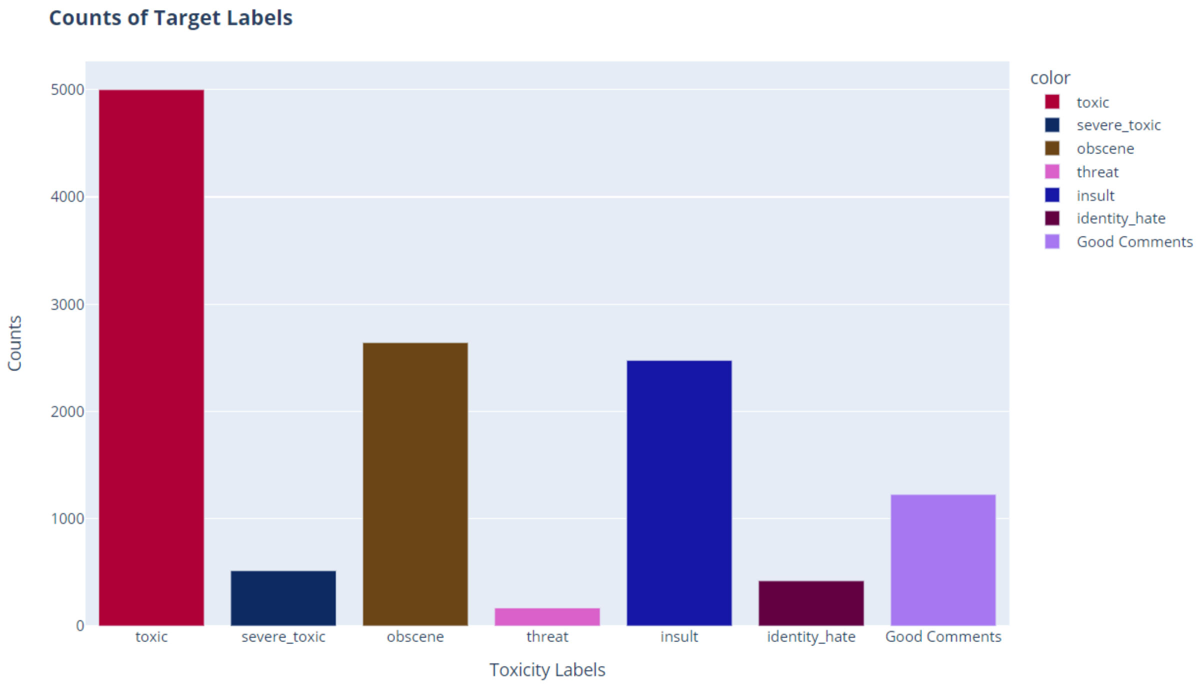
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   comment_text   10000 non-null  object
 1   toxic          10000 non-null  int64 
 2   severe_toxic   10000 non-null  int64 
 3   obscene        10000 non-null  int64 
 4   threat         10000 non-null  int64 
 5   insult         10000 non-null  int64 
 6   identity_hate  10000 non-null  int64 
dtypes: int64(6), object(1)
memory usage: 547.0+ KB
```

From Figure above we get more info about dataset to see if there exist any nulls and we found no nulls.

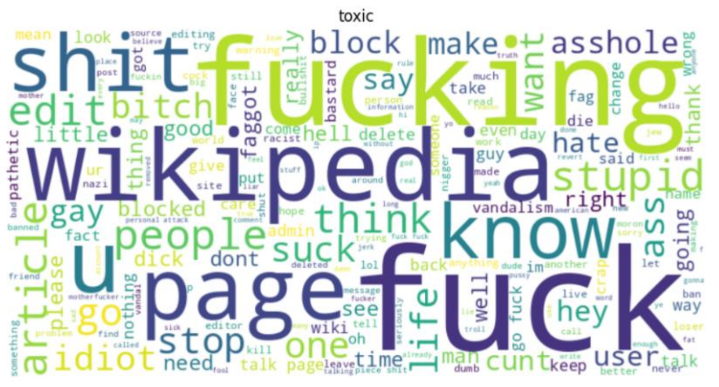
II.II Data Visualization:

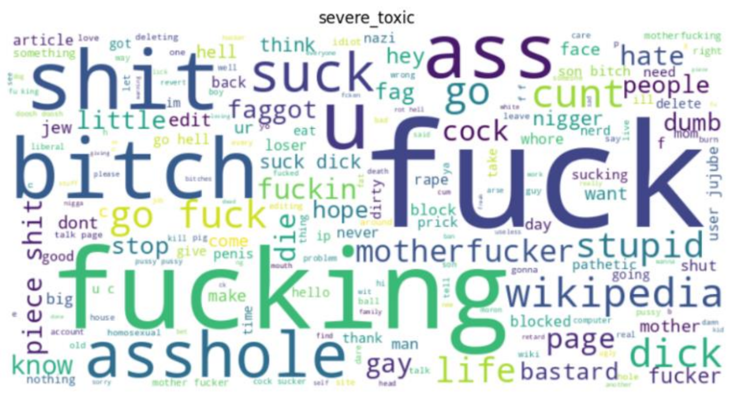
From the figure below we count the target labels to see if the data is biased and we found that the data is in good condition.

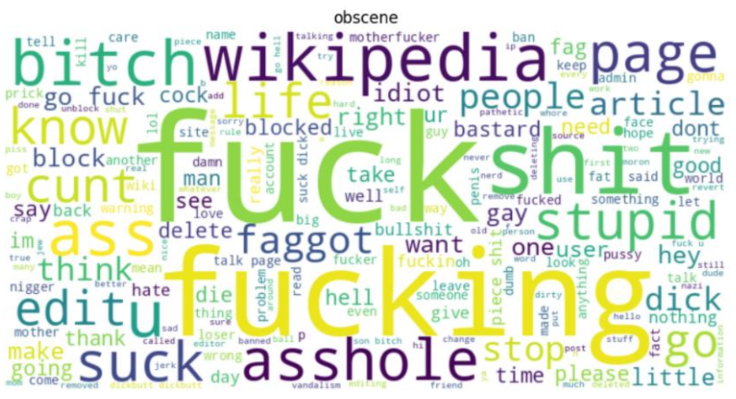


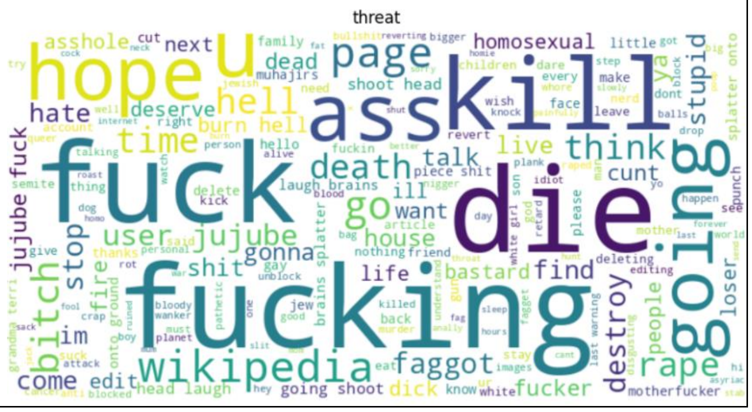
After that we used word cloud to make a graphical representation of word frequency in the given text, where the size of each word in the cloud is proportional to its frequency in the text. Word clouds are often used to visually summarize and highlight the most common words in a piece of text.

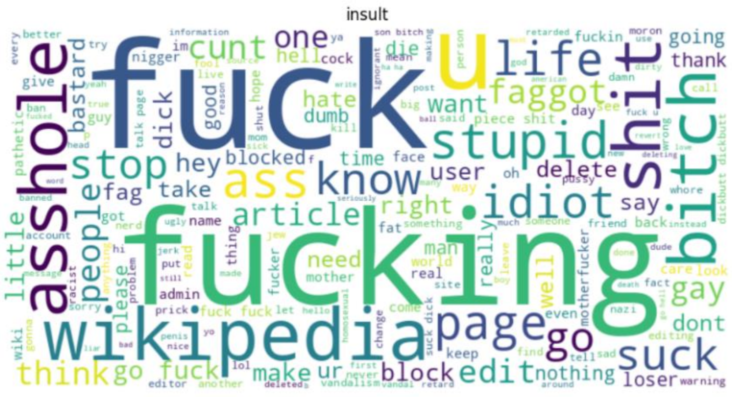
We make word cloud plot for each target column:

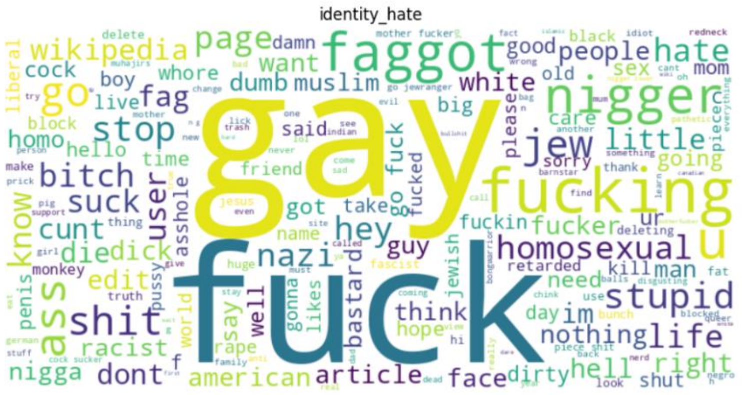






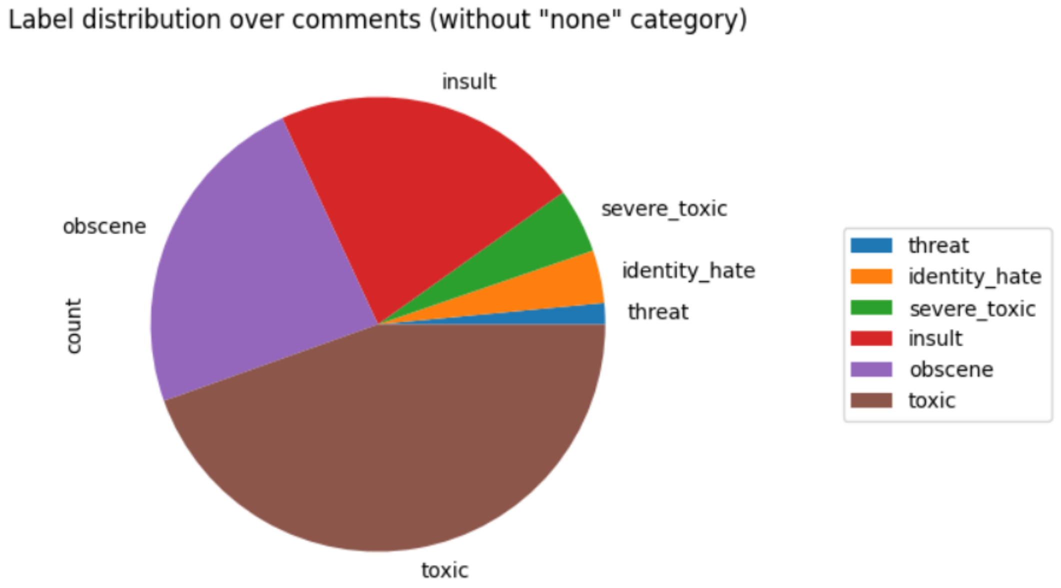




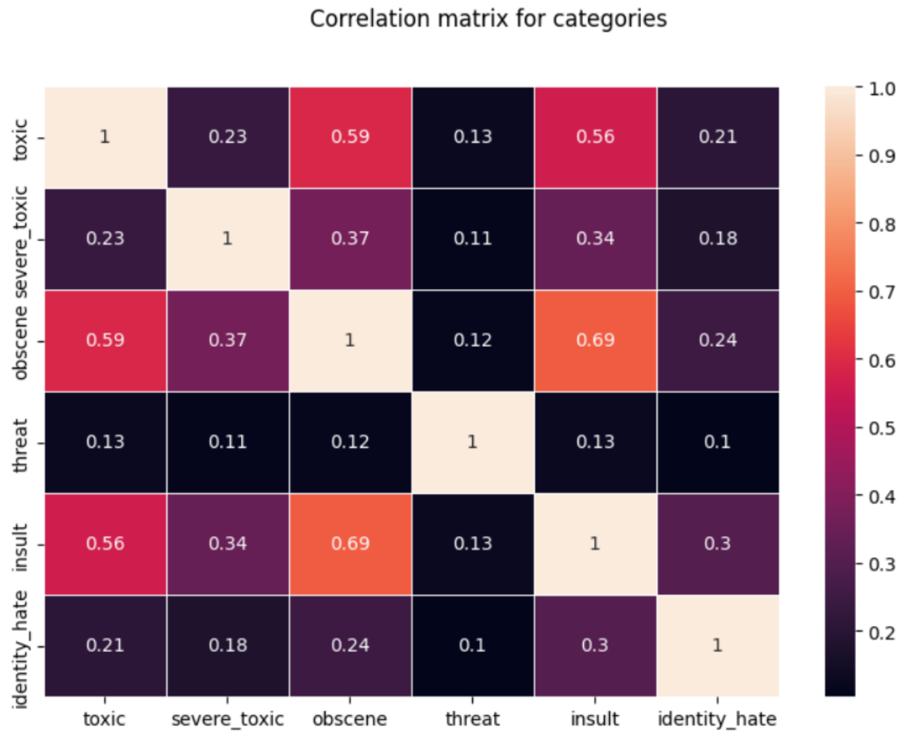


We can conclude the most frequent toxic words appeared in each target column

- Then we create a pie chart to visualize the count of different toxic label categories in your dataset.

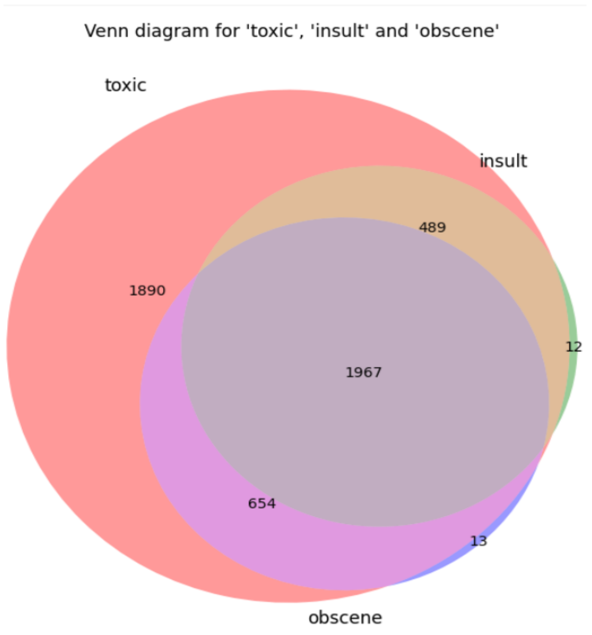


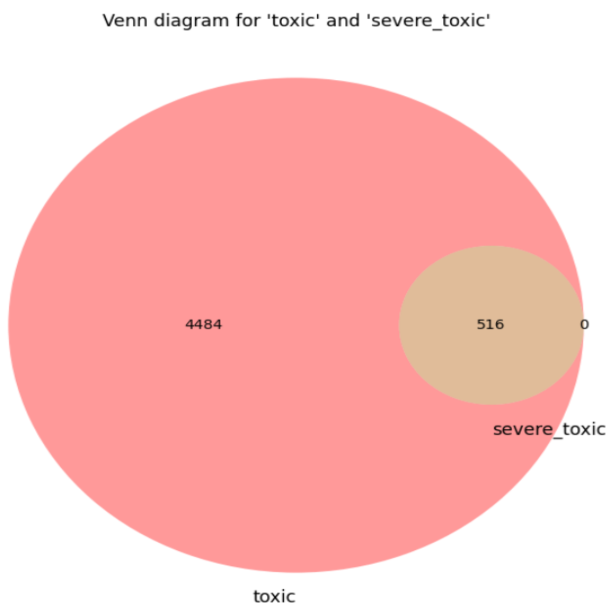
- Then create a heatmap to visualize the correlation matrix among different toxic label categories in your dataset.



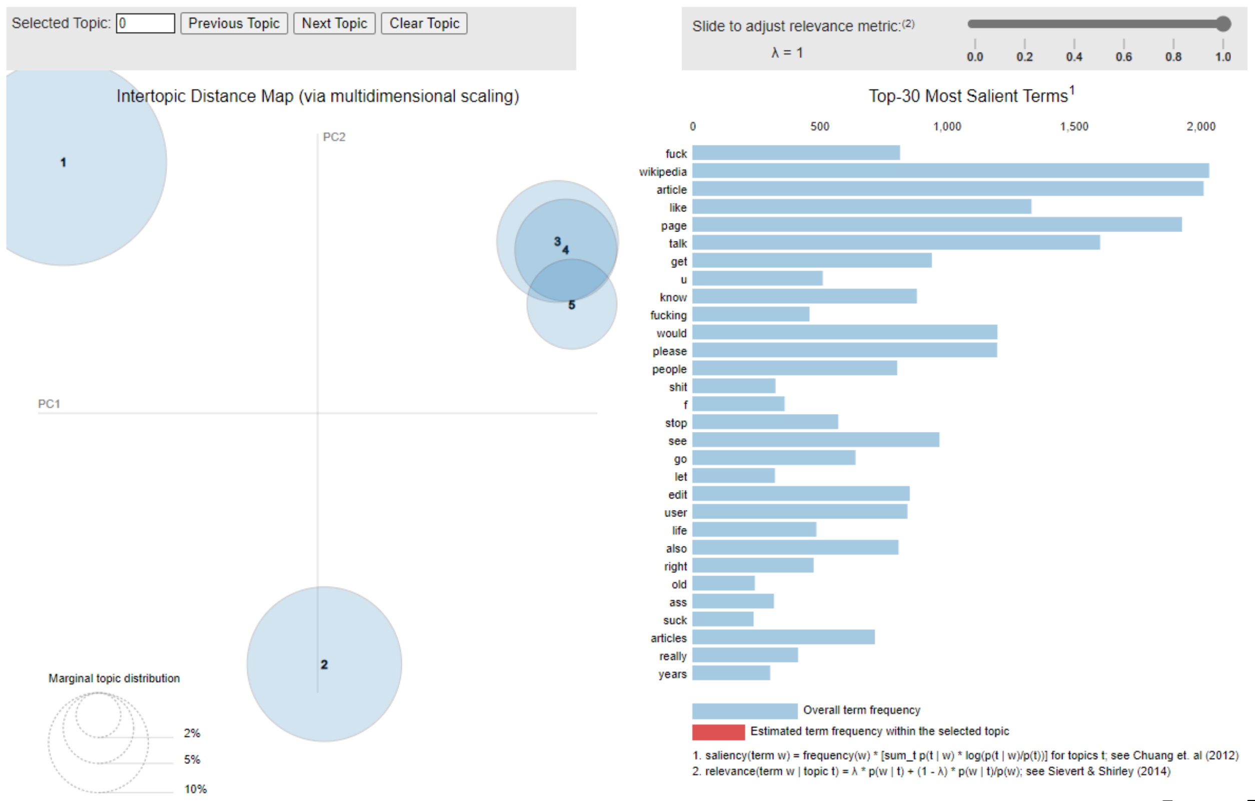
	toxic	severe_toxic	obscene	threat	insult	identity_hate
toxic	1	0.23	0.59	0.13	0.56	0.21
severe_toxic	0.23	1	0.37	0.11	0.34	0.18
obscene	0.59	0.37	1	0.12	0.69	0.24
threat	0.13	0.11	0.12	1	0.13	0.1
insult	0.56	0.34	0.69	0.13	1	0.3
identity_hate	0.21	0.18	0.24	0.1	0.3	1

- Then we used Venn diagram to visualize the relationships between three different label categories ('toxic', 'insult', and 'obscene') and for categories ('toxic' and 'severe_toxic') in the dataset. A Venn diagram is a graphical representation that shows the intersections and differences between sets of data. In this case, it helps you understand how many comments fall into each combination of label categories.



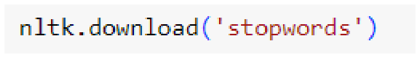


The following figure used to visualize the output of a topic modeling algorithm, typically applied to a collection of text documents. Topic modeling, such as Latent Dirichlet Allocation (LDA) provided by the gensim library, is a technique used in natural language processing and text analysis to discover the underlying topics within a corpus of text.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

II.III Data Preparation:

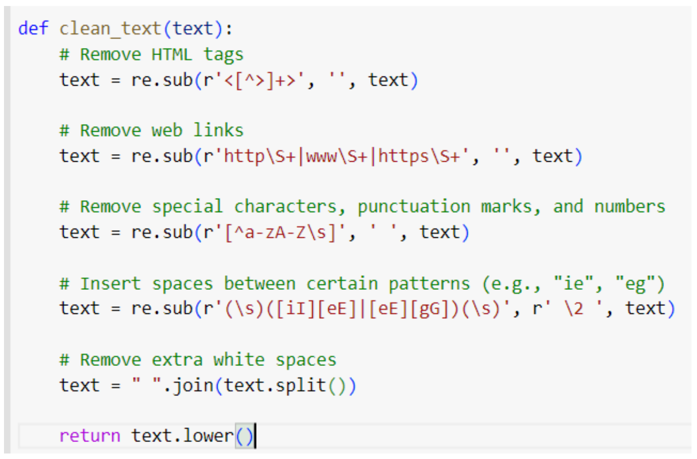


```
nltk.download('stopwords')
```

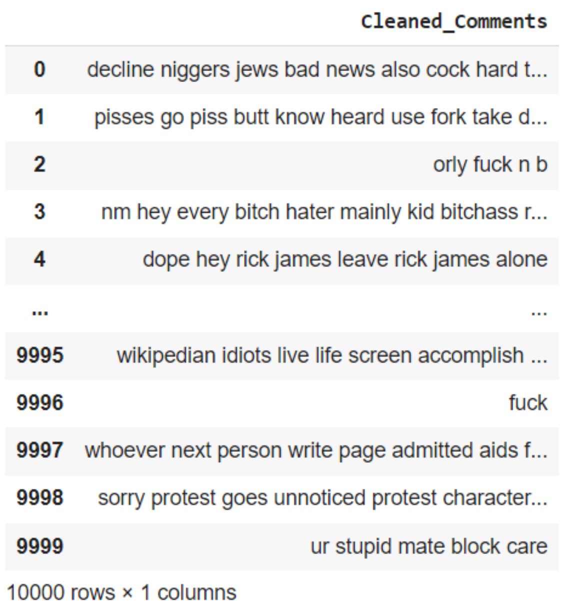
We used NLTK stopwords to download a set of stopwords from the NLTK (Natural Language Toolkit) library. Stopwords are commonly used words in a language (such as "and", "the", "is", "in", etc.) that are often removed from text data during natural language processing tasks like text analysis, text classification, and sentiment analysis. These words are generally considered to be of low informational value and are often excluded to focus on the more meaningful content words.

Then we cleaned the text and replace the html characters with " " then we removed the punctuations and considered only alphabets and numeric then we convert to lower case and we split and join the words

Then we create “clean_text” function. This function is used to preprocess and clean text data by removing HTML tags, web links, special characters, punctuation, and numbers, and then converting the text to lowercase the clean text function and final text cleaned should look like:



```
def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)

    # Remove web links
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)

    # Remove special characters, punctuation marks, and numbers
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)

    # Insert spaces between certain patterns (e.g., "ie", "eg")
    text = re.sub(r'(\s)([iI][eE]|[eE][gG])(\s)', r' \2 ', text)

    # Remove extra white spaces
    text = " ".join(text.split())

    return text.lower()
```



	Cleaned_Comments
0	decline niggers jews bad news also cock hard t...
1	pisses go piss butt know heard use fork take d...
2	orly fuck n b
3	nm hey every bitch hater mainly kid bitchass r...
4	dope hey rick james leave rick james alone
...	...
9995	wikipedian idiots live life screen accomplish ...
9996	fuck
9997	whoever next person write page admitted aids f...
9998	sorry protest goes unnoticed protest character...
9999	ur stupid mate block care

10000 rows × 1 columns

-Then we split the data into train and test set

We give 30% of data to test with it and 70% for training.

III. Feature Engineering:

BOW: We used bag of words to convert a collection of text documents into a format that can be understood by machine learning algorithms. The BOW model treats each document as a "bag" of individual words, disregarding the order and structure of the words, and represents the documents as vectors in a high-dimensional space. Each dimension corresponds to a unique word in the entire corpus.

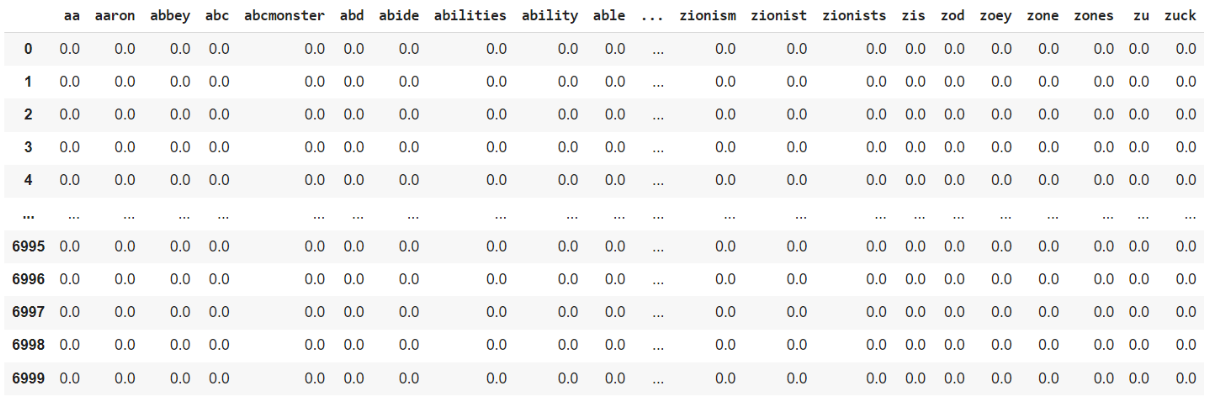
To summarize the usefulness of BOW in our problem:

1. Simplicity
2. Feature Extraction
3. Scalability
4. Language Independence
5. Versatility
6. Preprocessing

TF-IDF: Then we used it as it is text representation technique in natural language processing (NLP) and machine learning. It's used to address some of the limitations of the Bag of Words (BoW) model and provide a more sophisticated way to represent text data. TF-IDF assigns a numerical weight to each word in a document based on its importance in the context of a collection of documents. Here's why TF-IDF is used in text analysis of our problems:

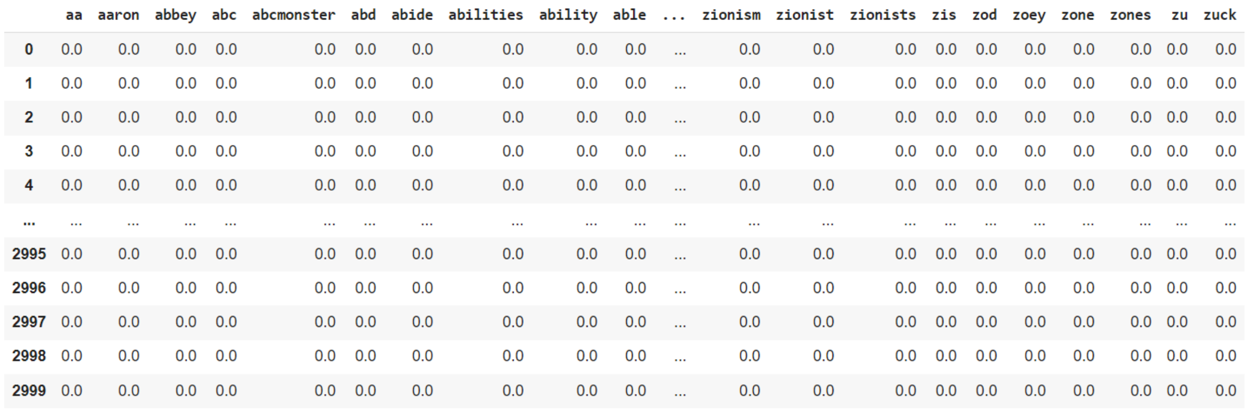
1. Term Importance
2. Addressing Common Words
3. Contextual Understanding
4. Sparse Representation
5. Variety of Applications
6. Flexibility

Here is the Result for TF-IDF for both training and test set:



	aa	aaron	abbey	abc	abcmonster	abd	abide	abilities	ability	able	...	zionism	zionist	zionists	zis	zod	zoey	zone	zones	zu	zuck
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

7000 rows × 10000 columns

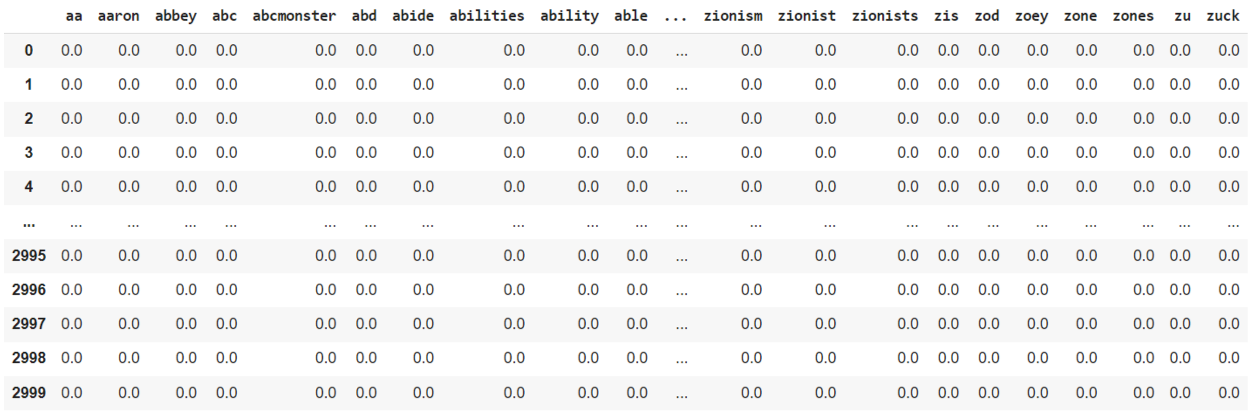


	aa	aaron	abbey	abc	abcmonster	abd	abide	abilities	ability	able	...	zionism	zionist	zionists	zis	zod	zoey	zone	zones	zu	zuck
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3000 rows × 10000 columns

Then we used TF-IDF Vectorized, and output is like TF-IDF:

Test set output for TF-IDF Vectorized



	aa	aaron	abbey	abc	abcmonster	abd	abide	abilities	ability	able	...	zionism	zionist	zionists	zis	zod	zoey	zone	zones	zu	zuck
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3000 rows × 10000 columns

TSNE of TF-IDF:

N-grams: We used it as it is common technique used in natural language processing (NLP) and text analysis to capture local patterns and contextual information within sequences of words or characters. They are important because they provide a way to represent language in a more meaningful and informative manner It is good for:

1. Capturing Context

2. Phrase Recognition

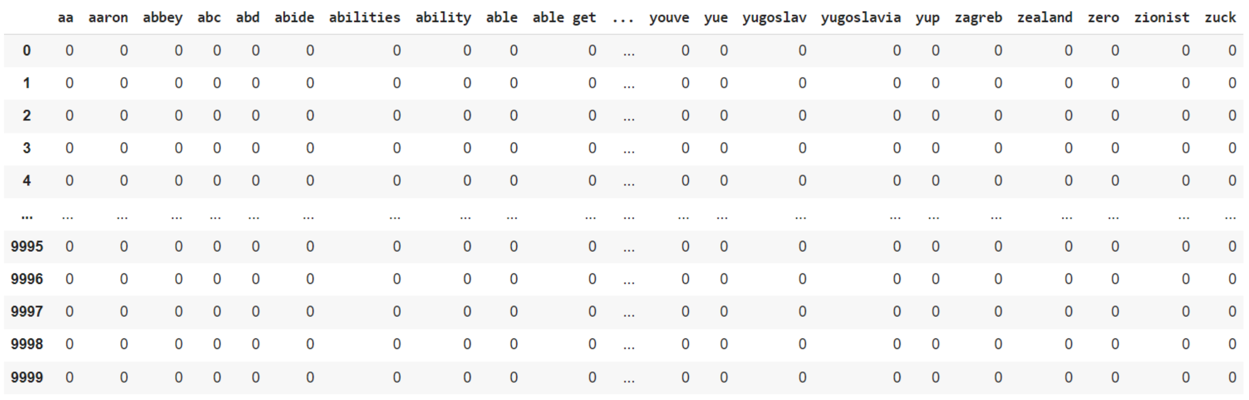
3. Local Patterns

4. Text Classification

5. Sentiment Analysis

6. Language Modeling

And here is our results for N-gram



	aa	aaron	abbey	abc	abd	abide	abilities	ability	able	able get	...	youve	yue	yugoslav	yugoslavia	yup	zagreb	zealand	zero	zionist	zuck
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

10000 rows × 10000 columns

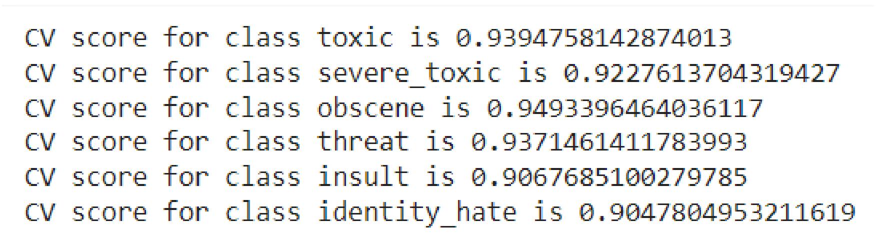
IV. Classification:

We trained our model using **Logistic Regression** as it is a commonly used classification algorithm that is well-suited for binary and multi-class classification tasks like our problem.

In this snippet, it is applied to each class separately to perform multi-label classification. Each class is treated as an independent binary classification problem.

The LR models can predict whether a given document belongs to a specific class based on the TF-IDF representation of its text by using separate LR models for each class, the code can handle the multi-label nature of the classification task, where each document can be associated with multiple classes.

Here is our results for training using Logistic Regression:



```
CV score for class toxic is 0.9394758142874013
CV score for class severe_toxic is 0.9227613704319427
CV score for class obscene is 0.9493396464036117
CV score for class threat is 0.9371461411783993
CV score for class insult is 0.9067685100279785
CV score for class identity_hate is 0.9047804953211619
```

- Then we used **SVM** as it is a popular machine learning algorithm used for both binary and multi-class classification tasks. SVMs can also be adapted for multi-label classification, where each instance can belong to multiple classes simultaneously. In the provided code snippet, SVM is used as the classifier for a multi-label classification problem what it is good for:

1. Effective for High-Dimensional Data

2 . Non-Linearity

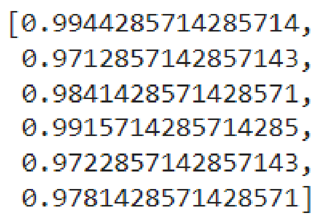
3. Good Generalization

4. Regularization

5. Multi-Label Classification

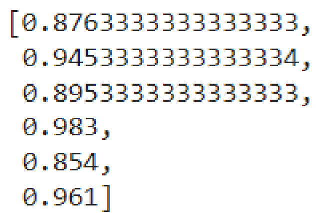
6. Predictive Power

The results of training and testing in SVM with TF-IDF:



```
[0.9944285714285714,
 0.9712857142857143,
 0.9841428571428571,
 0.9915714285714285,
 0.9722857142857143,
 0.9781428571428571]
```

Training Accuracy



```
[0.8763333333333333,
 0.9453333333333334,
 0.8953333333333333,
 0.983,
 0.854,
 0.961]
```

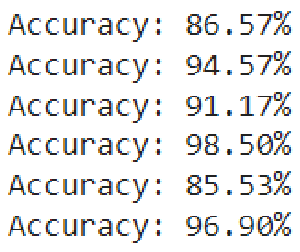
Testing Accuracy

We can see in right figure scores for the testing.

We used XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that has gained popularity and achieved state-of-the-art results in various machine learning competitions and real-world applications. XGBoost is particularly well-suited for a wide range of classification and regression tasks, including text classification what it is good for:

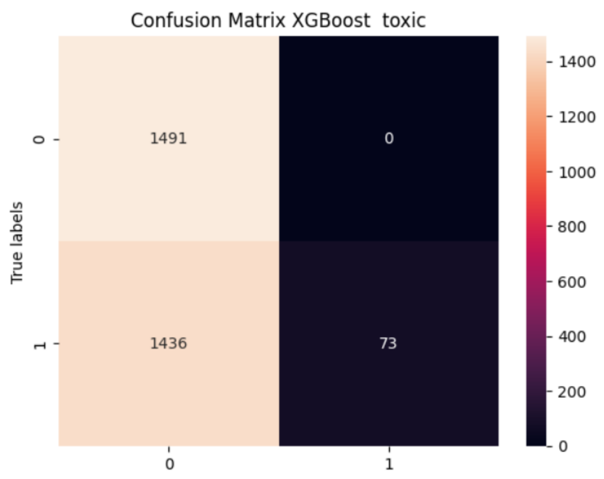
1. Performance
2. Handling Non-Linearity
3. Regularization
4. Ensemble Learning
5. Feature Importance
6. Flexibility
7. Availability
8. Efficiency

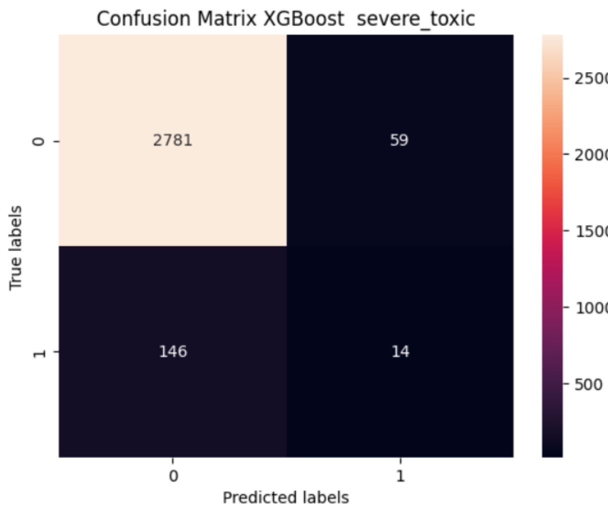
Here is the accuracies for our Model XGBoost with TF-IDF

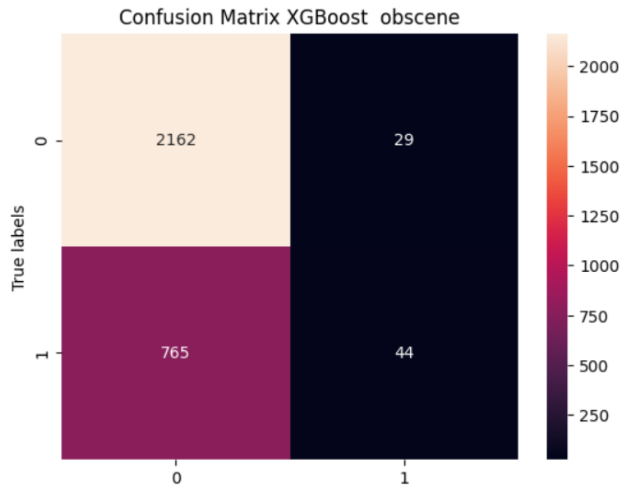


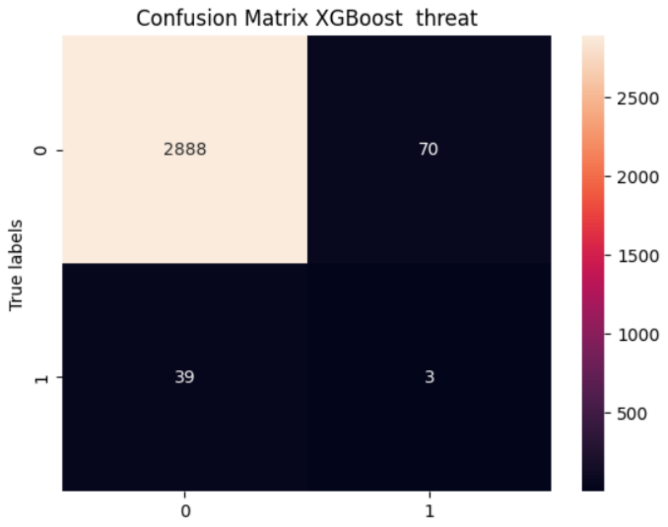
```
Accuracy: 86.57%
Accuracy: 94.57%
Accuracy: 91.17%
Accuracy: 98.50%
Accuracy: 85.53%
Accuracy: 96.90%
```

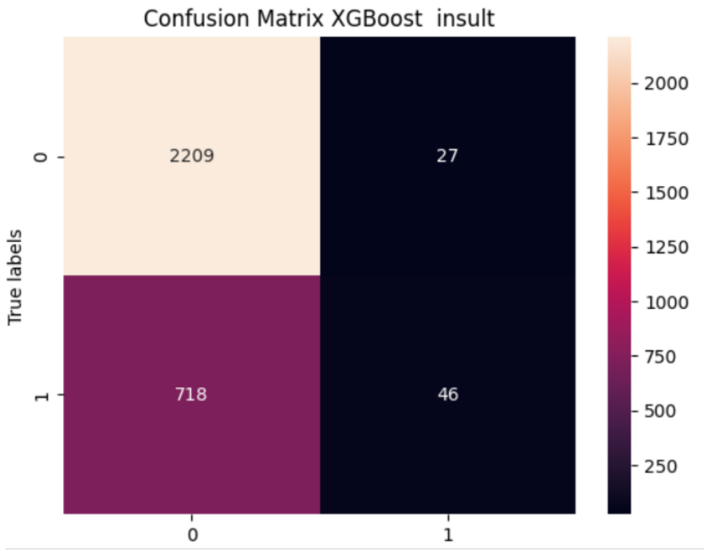
Now we will visualize the confusion matrix for XGBoost for predicted value for each target.

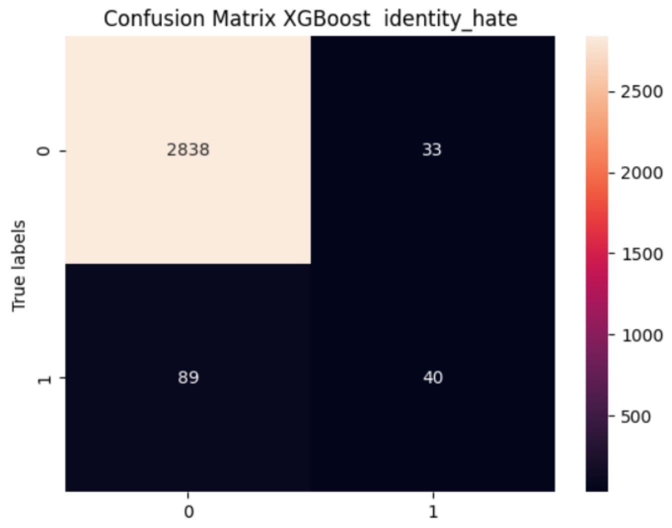












V. Evaluation of ML Results:

VI. Clustering:

For our clustering we used K-Means as it is a widely used clustering algorithm that partitions data points into distinct groups or clusters based on their similarity. It is commonly used in various fields, including data analysis, machine learning, image processing, etc and it is good for:

1. Simplicity and Speed

2. Scalability

3. Interpretability

4. Unsupervised Learning

5. Segmentation and Grouping

6. Feature Engineering

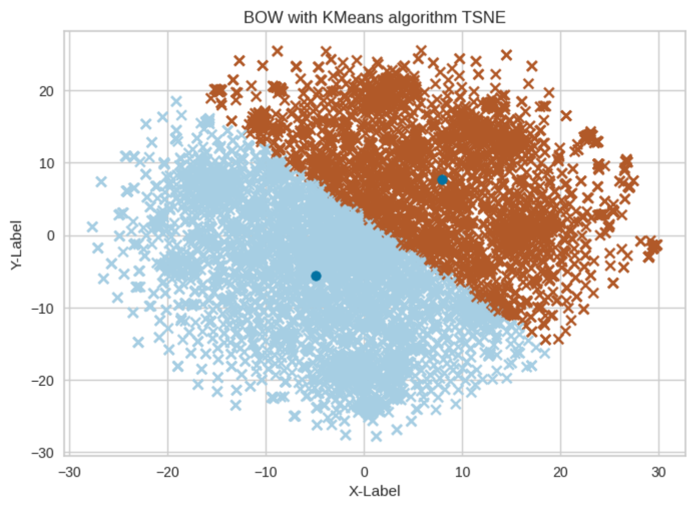
7. Clustering Validation

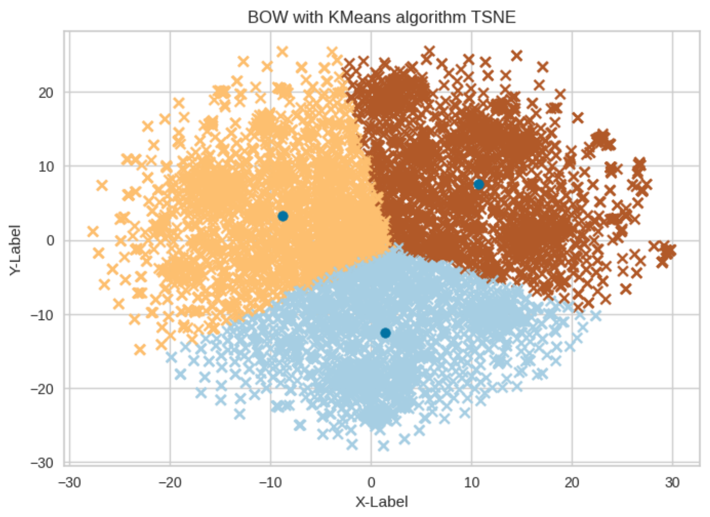
We applied K-Means in our model to cluster our targets and have high accuracy.

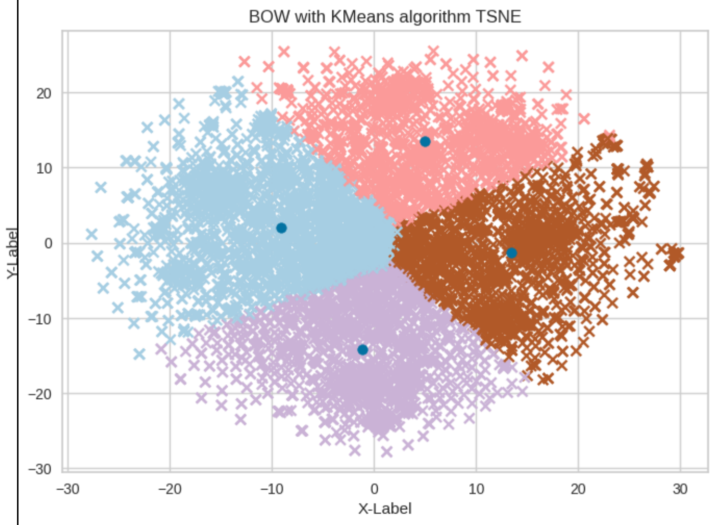
Using Bag-of-Words (BoW) representation with the K-Means clustering algorithm is a common approach in text analysis and natural language processing. BoW is a simple and effective way to represent text data numerically, and K-Means is a popular clustering algorithm that can be applied to group similar text documents together.

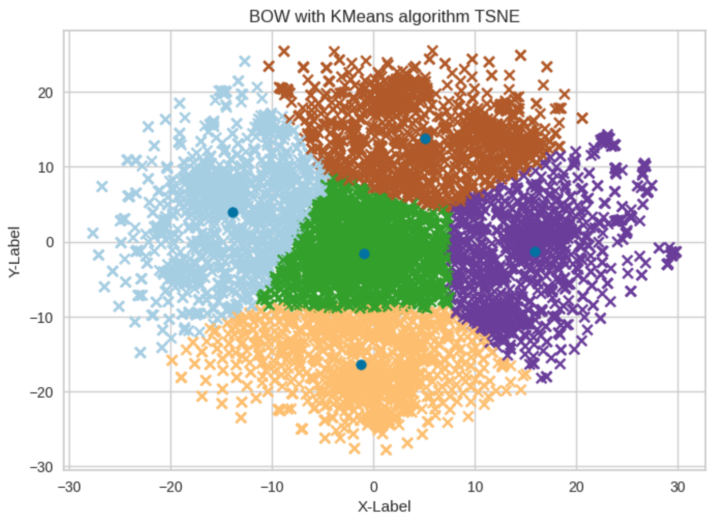
And here is the results for:

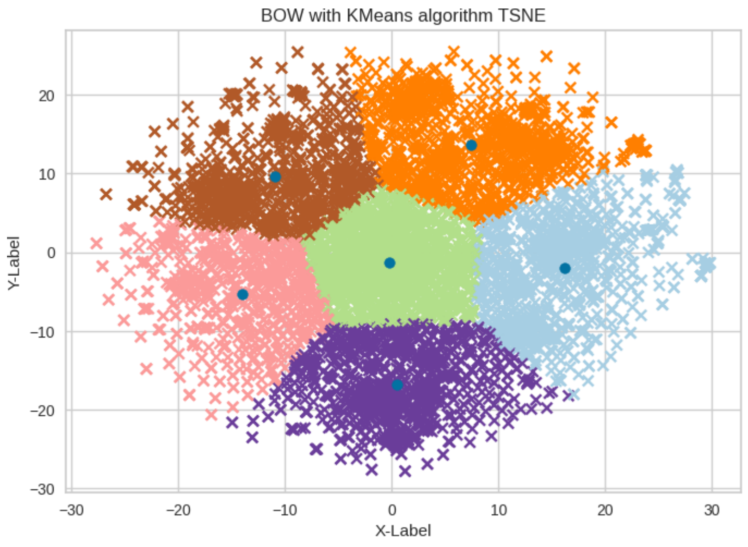
1. BOW with KMeans algorithm TSNE with K = 7

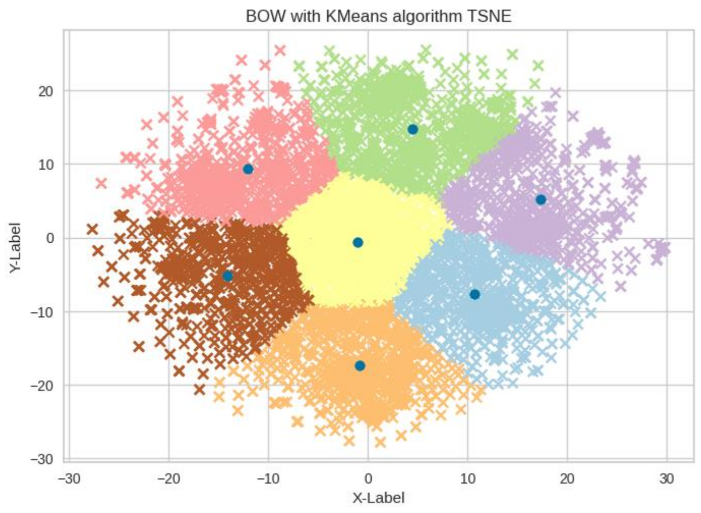


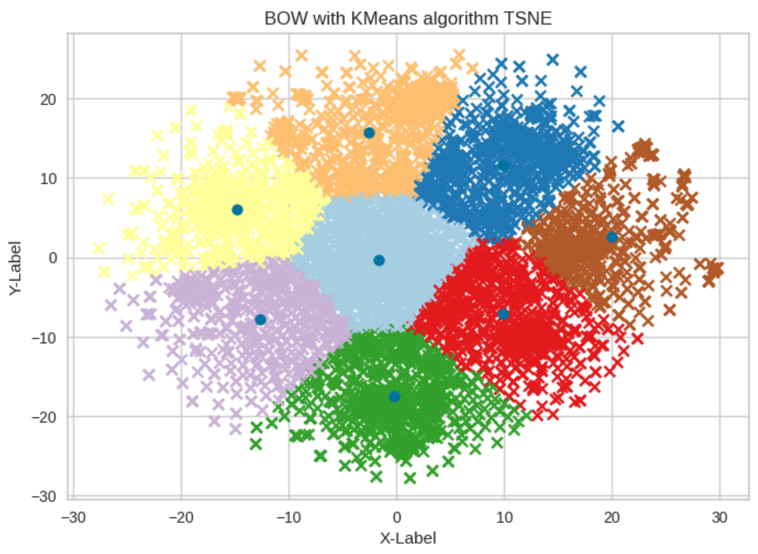



BOW with KMeans algorithm TSNE
Y-Label
X-Label


BOW with KMeans algorithm TSNE
Y-Label
X-Label

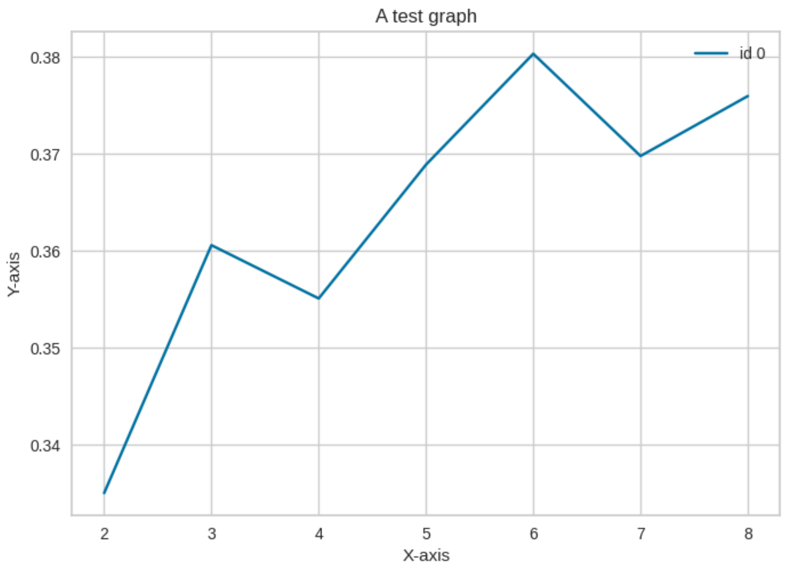

BOW with KMeans algorithm TSNE
Y-Label
X-Label


BOW with KMeans algorithm TSNE
Y-Label
X-Label

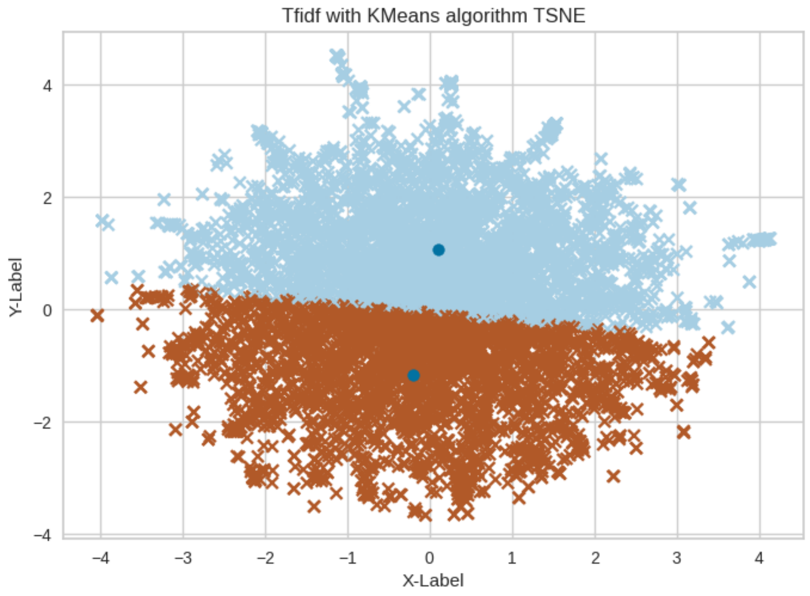

BOW with KMeans algorithm TSNE
Y-Label
X-Label

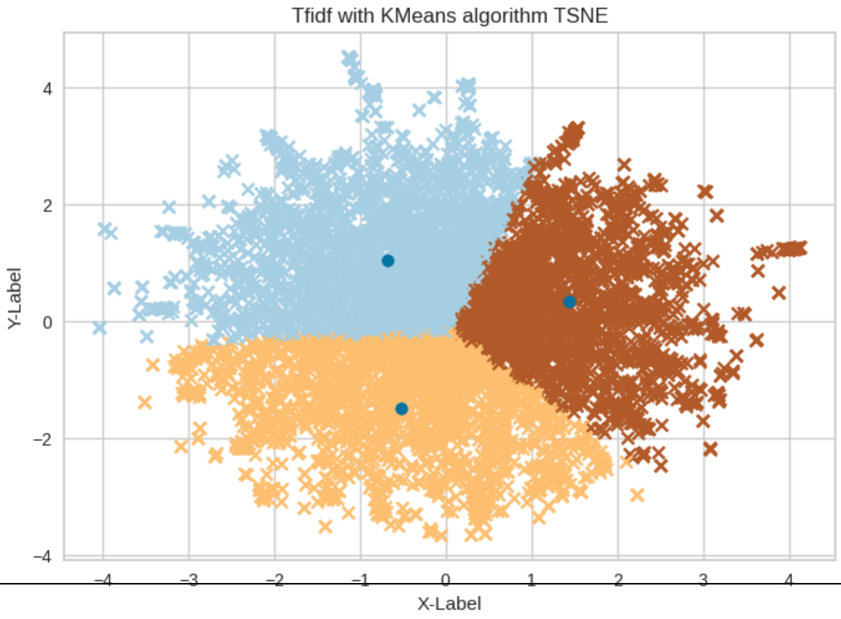
Group 5

2. Silhouette score BOW

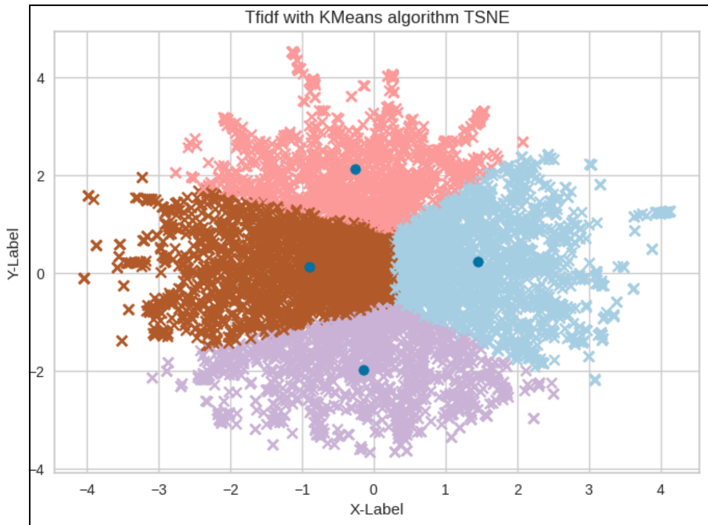


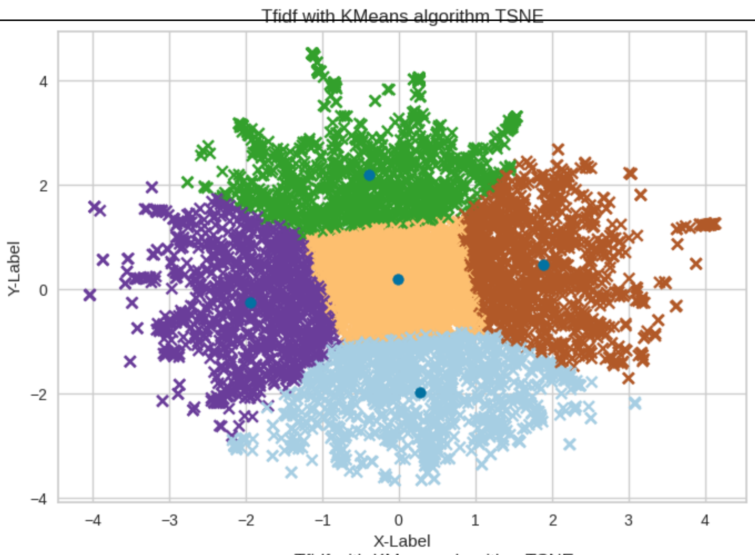
3. TF-IDF with KMeans algorithm TSNE with K = 7

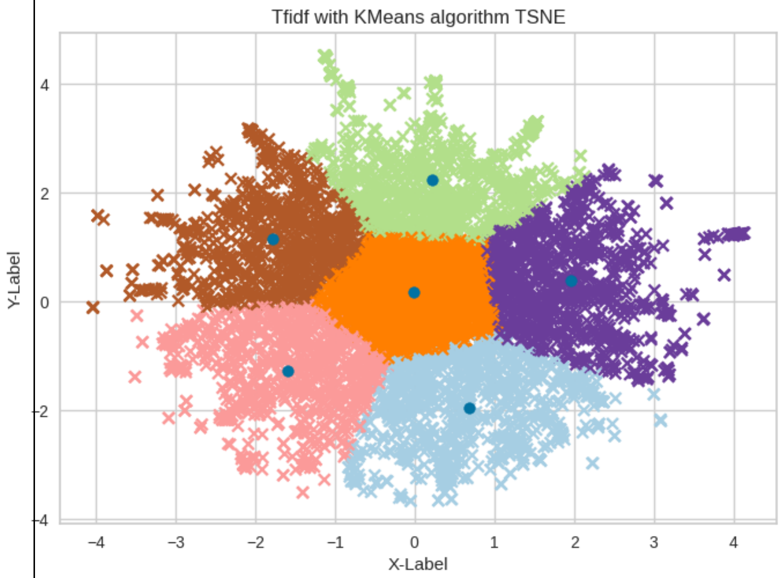


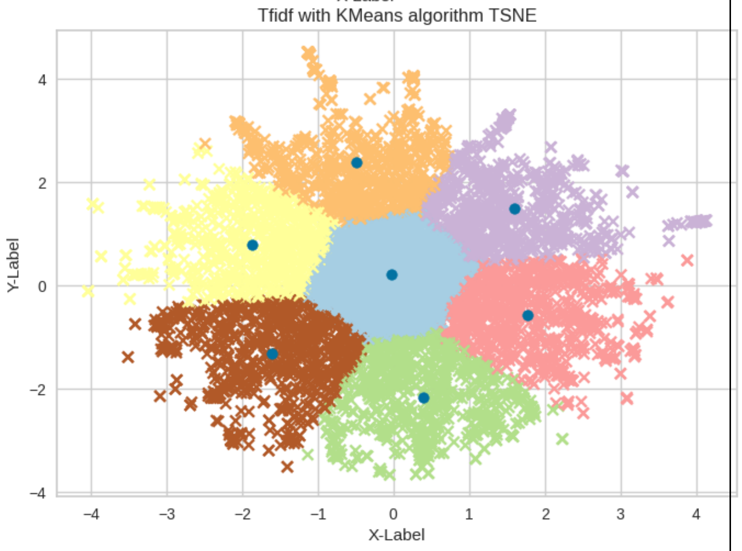


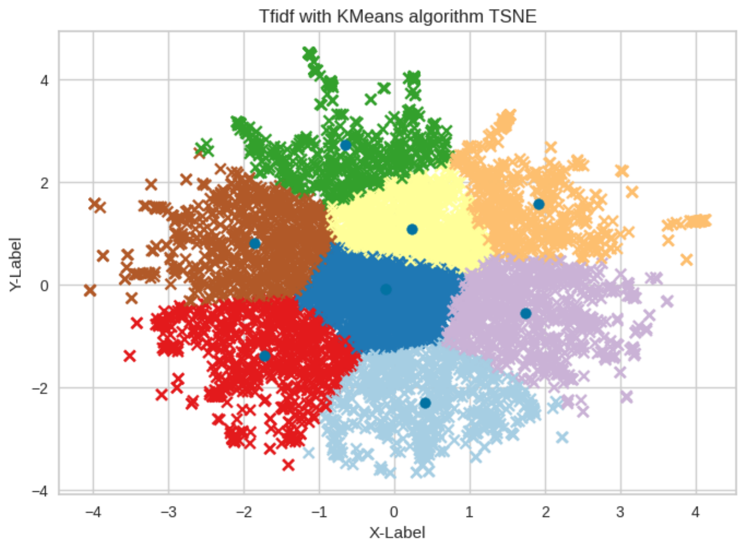
Group 5



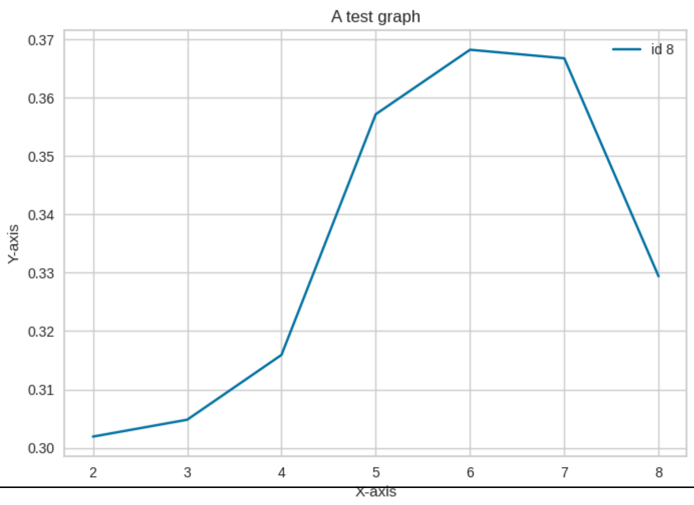








Silhouette score BOW:



VII. Error Analysis:

The confusion matrix in the image shows the results of a machine learning model that was trained to classify text as either toxic or not toxic. The model was able to correctly classify 1436 text samples as not toxic and 73 text samples as toxic. However, the model also made two errors: it classified 0 text samples as toxic that were actually not toxic, and it classified 1491 text samples as not toxic that were actually toxic.

The two errors in the confusion matrix represent two types of errors that can be made by machine learning models: false positives and false negatives. False positives occur when the model classifies a sample as positive when it is actually negative. In this case, the model classified 0 text samples as toxic that were actually not toxic. False negatives occur when the model classifies a sample as negative when it is actually positive. In this case, the model classified 1491 text samples as not toxic that were actually toxic.

The error rate of the model can be calculated by dividing the number of errors by the total number of samples. In this case, the error rate is (0 + 1491) / (1436 + 73 + 0 + 1491) = 0.098, or 9.8%.

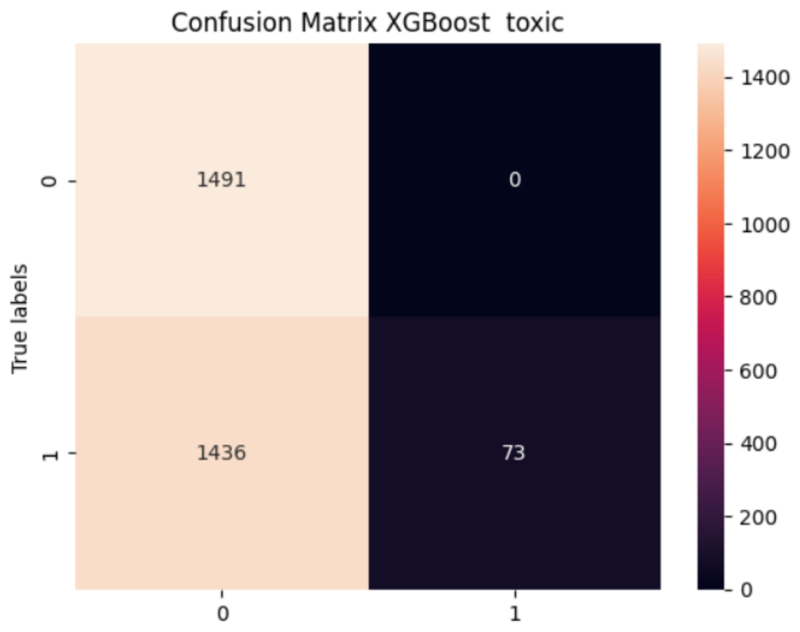
The error rate of the model can be improved by adjusting the hyperparameters of the model, or by training the model on a larger dataset. However, it is important to note that there is no such thing as a perfect machine learning model. All machine learning models will make some errors.

The following are some of the possible reasons why the model made the errors in the confusion matrix:

The dataset that the model was trained on was not balanced. This means that there were not equal numbers of toxic and non-toxic text samples in the dataset. This can lead to the model overfitting to the majority class, which in this case is the non-toxic class.

The model was not trained for a long enough time. This can lead to the model not learning the patterns in the data as well as it could.

The model was not complex enough. This can lead to the model not being able to capture the nuances in the data.



For **SVM** with **TF-IDF** we can see that the insult performs bad in test as it might be a problem in:

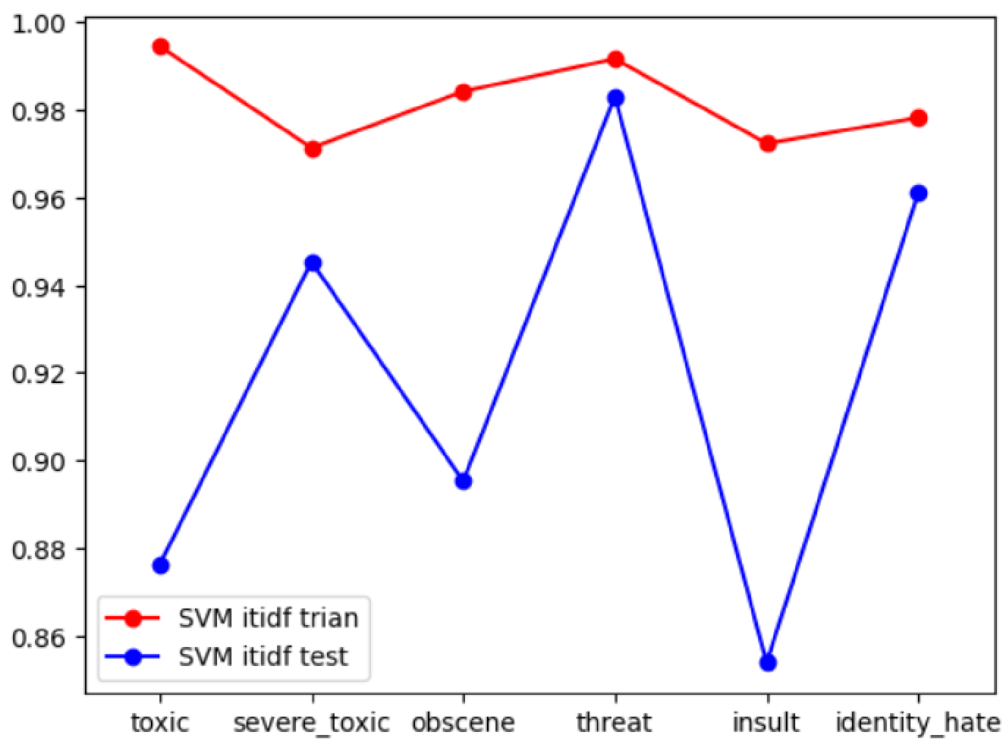
1. Data Quality and Labeling: The quality of the training data and the accuracy of the labels play a critical role in model performance. If the training data is noisy, inconsistent, or contains mislabeled examples, the model may struggle to learn meaningful patterns.

2. Class Imbalance: If the dataset is imbalanced, meaning one class (in this case, insults) has significantly fewer examples than the other class (non-insults), the model may have difficulty learning to differentiate the minority class.

3. Feature Representation: The choice of features used to represent the text data (e.g., Bag-of-Words, TF-IDF, word embeddings) can impact model performance. Inappropriate or insufficient features may not capture the relevant information needed to distinguish insults.

4. Model Complexity: The chosen machine learning algorithm and its complexity can affect performance. If the model is too simple, it may struggle to capture complex patterns. On the other hand, if it's too complex, it may overfit the training data.

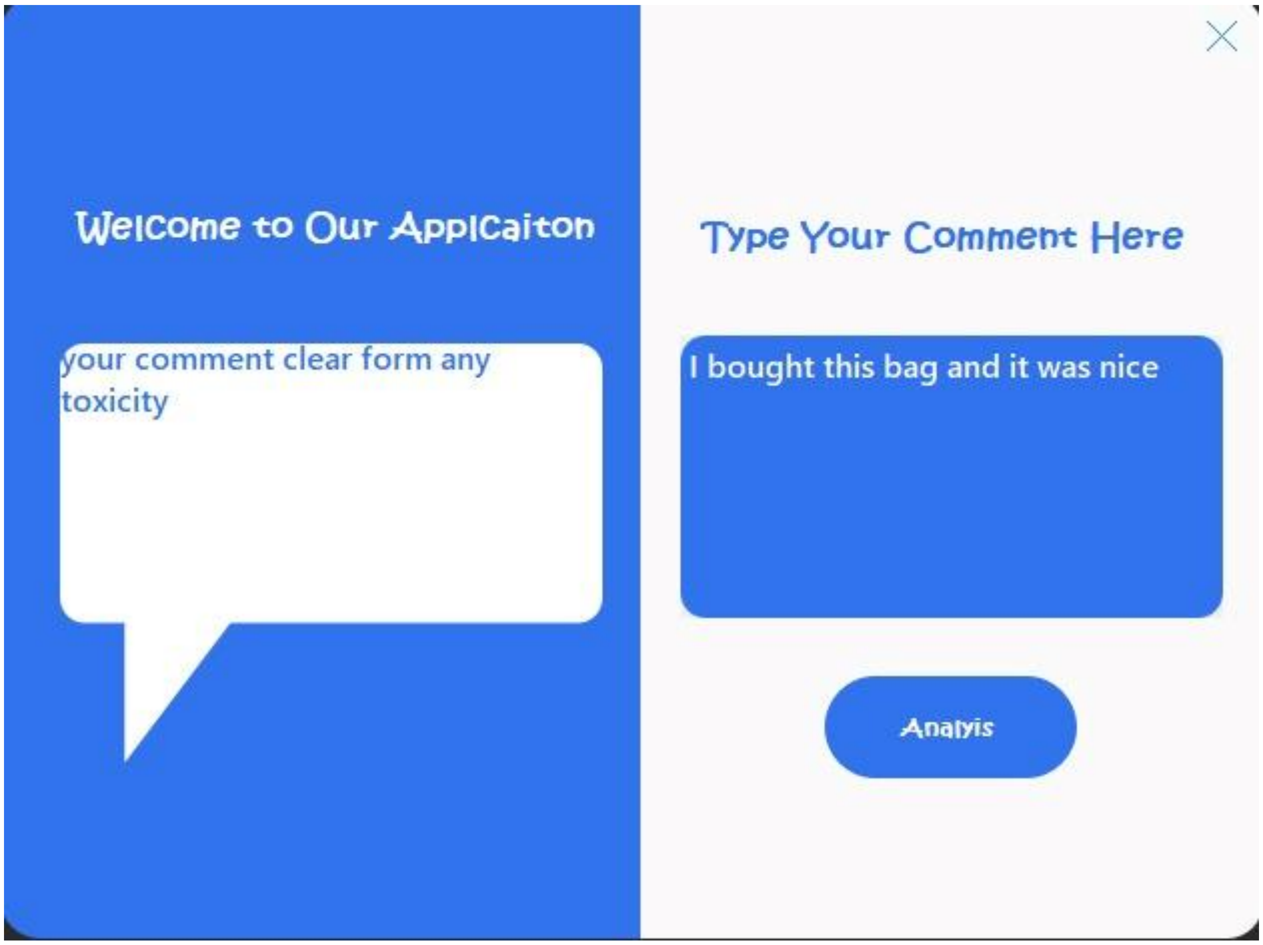
5. Hyperparameters: Model hyperparameters, such as learning rate, regularization strength, or the number of clusters in K-Means, can significantly influence performance. Poorly tuned hyperparameters may lead to suboptimal results.



VIII. Visualization of Results:

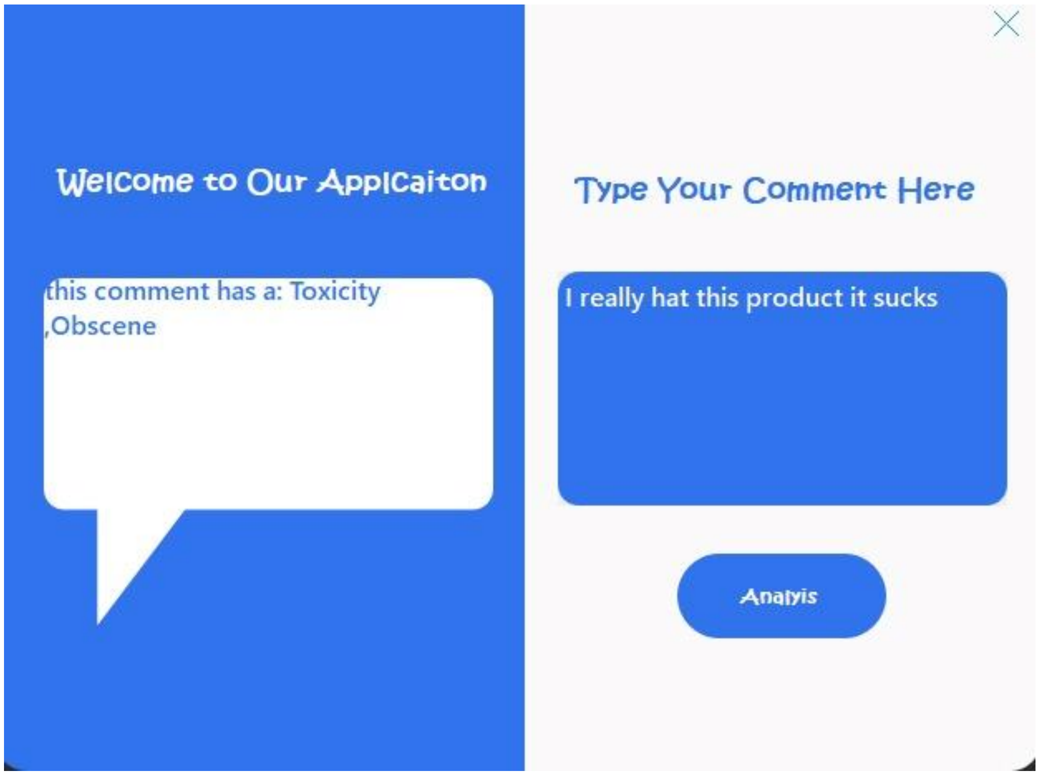
Our GUI was made by PyQt5 it is a Python library that provides bindings for the Qt application framework, allowing you to create graphical user interfaces (GUIs) for your Python applications. If you have a specific question or topic related to PyQt5 that you'd like to discuss or learn about, please provide more details, and I'd be happy to assist you! Whether it's about creating GUI applications, handling events, using widgets, or any other aspect of PyQt5.

-Here we test a comment with no Toxicity on it and the model classify it successfully.


Welcome to Our Applcaiton
your comment clear form any toxicity
Type Your Comment Here
I bought this bag and it was nice
Analyis

As we see in screen below we type a toxic comment for the product and I classify it Toxicity and Obscene

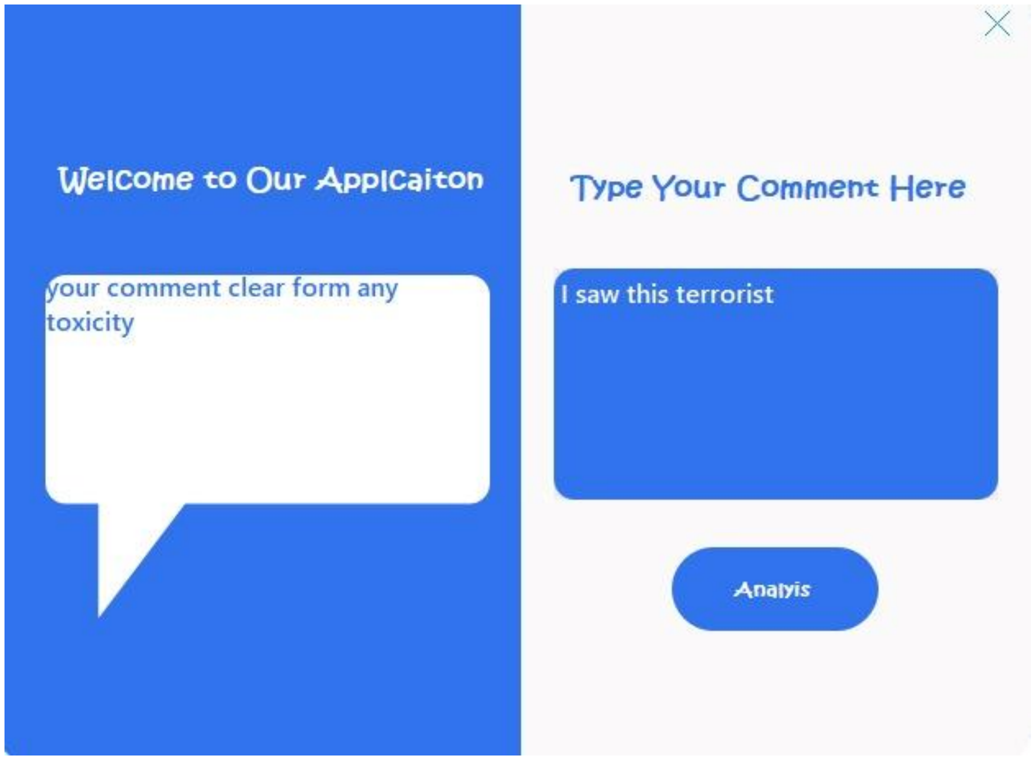
And that is correct.



this comment has a: Toxicity ,Obscene

I really hat this product it sucks

Here we can see our model didn't classify the comment correctly as we have problem in dataset it doesn't contain enough diplomatic words to classify it successfully

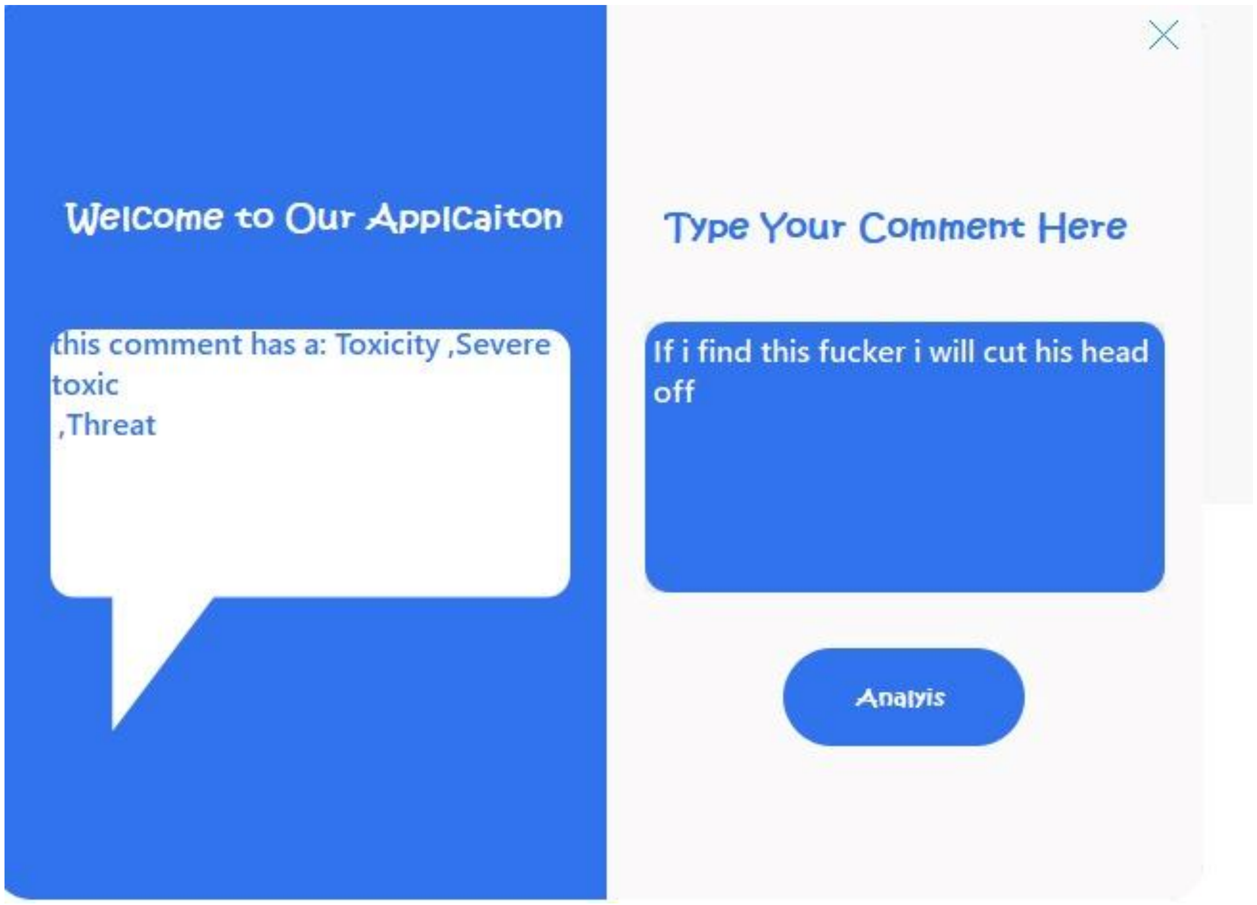


your comment clear form any toxicity

I saw this terrorist

- Finally, we tested one of the most toxic comments to see if the model will classify it correctly

And for the results below the model succeeded to classify the following toxic threat comment:



this comment has a: Toxicity ,Severe toxic
,Threat

If i find this fucker i will cut his head off

XI. Innovativeness:

Our project's innovativeness lies in its holistic approach to addressing toxicity in online comments in Wikipedia. By combining advanced feature engineering techniques like TF-IDF and N-grams with a diverse range of classification algorithms including Logistic Regression, SVM, and XGBoost, we achieve comprehensive insights and predictive accuracy across multiple toxicity types.

The integration of K-Means clustering enhances our understanding of comment patterns. Our rigorous error analysis identifies challenges in the insult class and sheds light on potential improvements. Through extensive visualization, we provide clear interpretations of complex results. This project's unique strength lies in its thorough exploration of text data, choice of algorithms, and detailed analysis, offering a multi-faceted solution to the pervasive issue of online toxicity.