

# Toxic Comment Classification For Social Media

## 1.INTRODUCTION:

Toxic Comment Classification for Social Media using NLP involves developing systems that automatically identify harmful and abusive language in online comments. This is crucial for creating safer and more inclusive digital environments. By leveraging Natural Language Processing (NLP) techniques, this approach can efficiently detect toxic behavior such as hate speech, threats, and harassment. It helps in flagging and moderating inappropriate content, thus protecting users and promoting positive interactions on social media platforms. The integration of machine learning models with NLP allows for continuous improvement in accuracy and adaptability to evolving language patterns.

### 1.1Project Objectives:

By the end of this project:

You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.

You will be able to know how to pre-process/clean the data using different data pre-processing techniques.

Applying different algorithms according to the dataset

You will be able to know how to find the accuracy of the model.

You will be able to build web applications using the Flask framework.

### 1.2 Objectives

#### Objectives of toxic comment classification for social using nlp

The objectives of Toxic Comment Classification for Social Media using NLP are:

**Identify and Classify Toxic Comments:** Automatically detect and categorize toxic language, including hate speech, harassment, and offensive comments.

**Improve User Experience:** Create a safer and more inclusive online environment by reducing exposure to harmful content.

**Support Moderation:** Aid human moderators by flagging potential toxic comments for review, making the moderation process more efficient.

**Enhance Community Guidelines Enforcement:** Ensure adherence to community standards by swiftly identifying and addressing violations.

**Adapt to Evolving Language:** Continuously learn and adapt to new forms of toxic language and slang to maintain high detection accuracy.

**Promote Positive Interaction:** Encourage healthier and more constructive communication among users on social media platforms.

### **Pre-requisites:**

To complete this project, you must require the following software, concepts, and packages.

#### **1. IDE Installation:**

Spyder/ Jupyter is Ideal to complete this project

To install Spyder IDE, please refer to Spyder IDE Installation Steps

To install Jupyter, please refer to Jupyter IDE Installation Steps

To install Pycharm IDE ,please refer to Pycharm IDE installation steps

#### **2. Python Packages**

If you are using the vs code, follow the below steps to download the required packages:

Open the vs code.

- Type “pip install pandas” and click enter.

- Type “pip install numpy ” and click enter

Type “pip install matplotlib” and click enter

Type “pip install scikit-learn” and click enter..

- Type “pip install flask” and click enter.

### **Prior Knowledge:**

You must have prior knowledge of the following topics to complete this project.

- Supervised and Unsupervised learning: [https://youtu.be/kE5QZ8G\\_78c](https://youtu.be/kE5QZ8G_78c)
- Neural Networks: <https://youtu.be/vpOLiDyhNUA>
- Flask: [https://youtu.be/lj4I\\_CvBnt0](https://youtu.be/lj4I_CvBnt0)

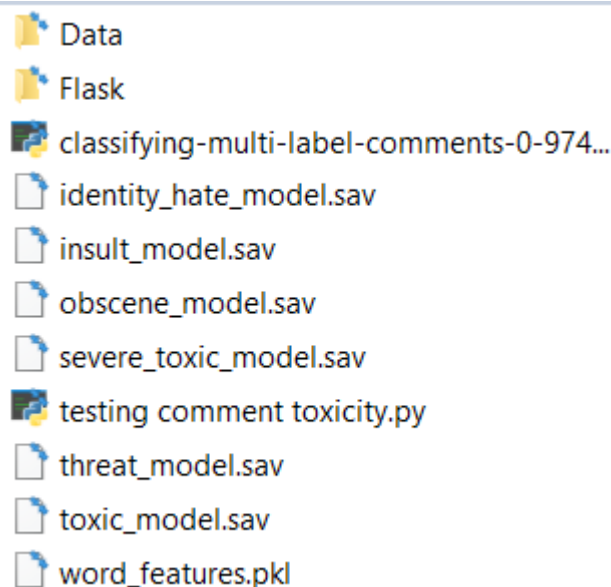
## Project Flow:

Find below the project flow to be followed while developing the project.

- Download the dataset.
- Preprocess the textual data.
- Classify the dataset into train and test sets.
- Add the neural network layers.
- Load the trained data and fit the model.
- Test the model.
- Save the model and its dependencies.
- Build a Web application using flask that integrates with the model built.

## Project Structure:

Create a Project folder which contains files as shown below.



- “classifying-multi-label-comments-0-9741-lb.py” has all the mastery model building architecture, that Collects Data from “train” and “test”, Import necessary packages, Pre-process text and passes on to Network Model, and saves Model Weights into “word\_features.pkl”.
- “commentApp.py” takes weights and inputs from “User Interface” to Predict output.

## Milestone 1: Collection of Data

To start with, we have to select or identify an accurate dataset.

The dataset which contains a set of features through which toxic comments can be identified is to be collected. You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc.

### **Download the Dataset-**

[https://drive.google.com/file/d/1vuJw8Npjmow3EtLheDu7IzYFBLgtI3fm/view?usp=drive link](https://drive.google.com/file/d/1vuJw8Npjmow3EtLheDu7IzYFBLgtI3fm/view?usp=drive_link)

## **2.Project Initialization and Planning Phase**

**2.1 Define Problem Statement:** Social media platforms are inundated with user-generated content, a significant portion of which can be toxic, including hate speech, harassment, and abusive language. This toxic content not only negatively impacts users' experiences but also poses risks to mental health and overall community safety. Manual moderation of such a vast volume of content is impractical due to time, scale, and resource constraints.

**Objective:** The goal is to develop an automated system using Natural Language Processing (NLP) techniques to accurately detect and classify toxic comments. This system should efficiently identify harmful content, thereby aiding in timely moderation and fostering a safer and more inclusive online environment. The solution should adapt to evolving language patterns and slang to maintain high detection accuracy over time.

This problem statement outlines the challenges and goals of implementing a toxic comment classification system for social media.

### **2.2 Project Proposal:**

**Data Collection and Preparation:** Collect and clean a large dataset of social media comments, using NLP techniques for text preprocessing and feature extraction.

**Model Development:** Train and evaluate machine learning and deep learning models for accurate toxic comment classification.

**Deployment and Improvement:** Deploy the model via an API for real-time detection and continuously update it with new data to improve accuracy and adapt to evolving language patterns.

### **2.3 Initial Project Planning:**

#### **1. Define Objectives and Scope**

**Objective:** Develop a model to classify toxic comments and ensure safer online interactions.

**Scope:** Include various types of toxic comments like hate speech, harassment, and offensive language.

#### **2. Data Collection**

**Sources:** Identify and collect data from social media platforms, public datasets like Kaggle, and online forums.

**Labeling:** Ensure the dataset is labeled for different types of toxicity.

#### **3. Data Preparation**

**Cleaning:** Preprocess the text data to remove noise and ensure consistency.

**Feature Extraction:** Use NLP techniques such as tokenization, lemmatization, and vectorization (e.g., TF-IDF).

#### 4. Model Development

**Model Selection:** Experiment with machine learning models (Logistic Regression, SVM, Random Forest) and deep learning models (RNN, LSTM, Transformers).

**Training:** Split the data into training and validation sets, and train the models.

#### 5. Model Evaluation

**Metrics:** Evaluate models using accuracy, precision, recall, F1-score, and AUC-ROC.

**Cross-Validation:** Ensure robustness through cross-validation.

#### 6. Deployment

**API Development:** Create a RESTful API using Flask or FastAPI for real-time classification.

**Integration:** Deploy the model for use in social media platforms.

#### 7. Continuous Improvement

**Feedback Loop:** Continuously update the model with new data to improve accuracy.

**Adaptability:** Ensure the model adapts to new language trends and slang.

##### Activity 1: Import the required libraries

```
#importing some of the required libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
#get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
import re
import pickle
```

##### Activity 2: Read the Datasets

```
# We've separate data available for training and testing.
# Load training and test data
train_df = pd.read_csv('Data/train.csv')
test_df = pd.read_csv('Data/test.csv')

#DATA DESCRIPTION

# In the training data, the comments are labelled as one or more of the
# six categories; toxic, severe toxic, obscene, threat, insult and identity hate.
# This is essentially a multi-label classification problem.
cols_target = ['insult', 'toxic', 'severe_toxic', 'identity_hate', 'threat', 'obscene']
```

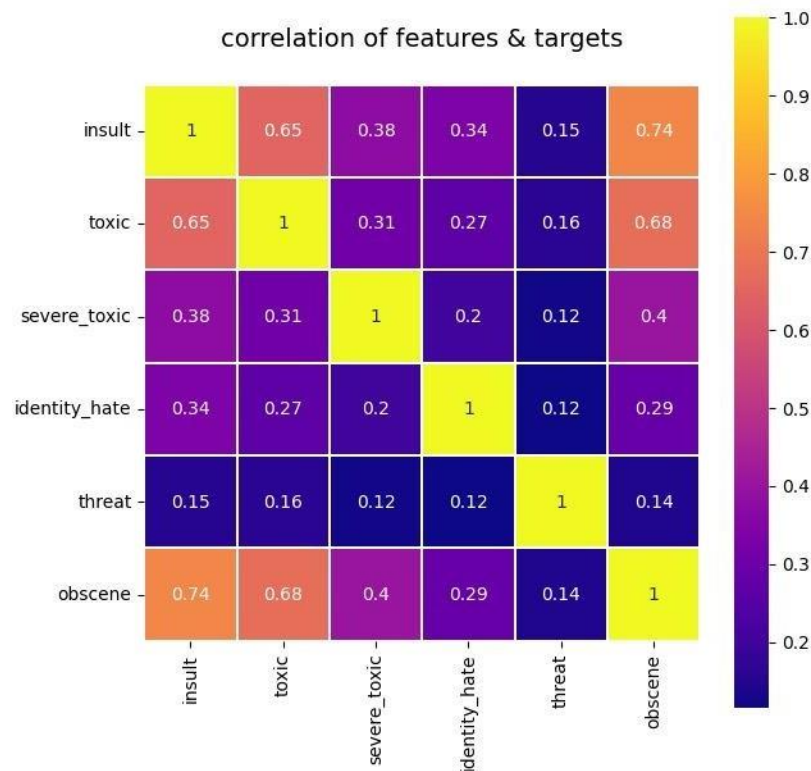
##### Activity3:CheckingNullValues

```
# check for null comments in test_df
print(test_df.isnull().any())
# no null values in test_df
```

#### Activity 4:Text PreProcessing: Checking the correlation between the variables.

```
[18]: data = test_df[cols_target]

[20]: colormap = plt.cm.plasma
      plt.figure(figsize=(7,7))
      plt.title('correlation of features & targets',y=1.05,size=14)
      sns.heatmap(data.astype(float).corr(),linewidths=0.1,vmax=1.0,square=True,cmap=colormap,linecolor='white',annot=True)
```

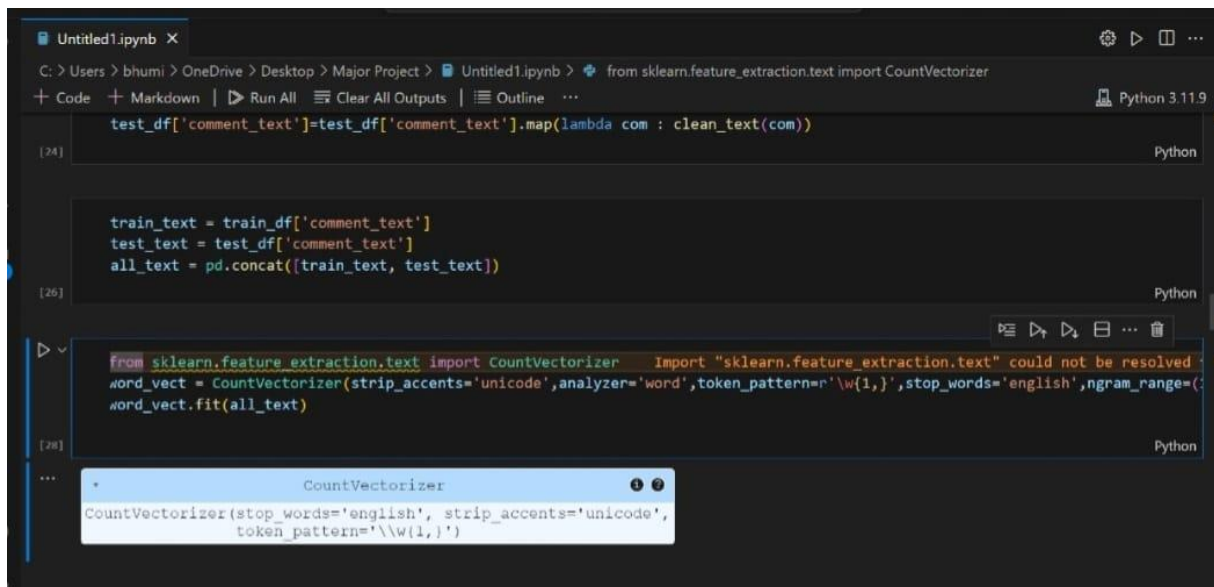


```
#DATA Pre-Processing

# Define a function to clean up the comment text, basic NLP
def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text

# clean the comment_text in both the datasets.
train_df['comment_text'] = train_df['comment_text'].map(lambda com : clean_text(com))
test_df['comment_text'] = test_df['comment_text'].map(lambda com : clean_text(com))
```

## Activity 5: Vectorizing the Data



```
from sklearn.feature_extraction.text import CountVectorizer

test_df['comment_text'] = test_df['comment_text'].map(lambda com : clean_text(com))

train_text = train_df['comment_text']
test_text = test_df['comment_text']
all_text = pd.concat([train_text, test_text])

word_vect = CountVectorizer(strip_accents='unicode', analyzer='word', token_pattern=r'\w{1,}', stop_words='english', ngram_range=(1, 1))
word_vect.fit(all_text)
```

CountVectorizer  
CountVectorizer(stop\_words='english', strip\_accents='unicode', token\_pattern='\\w{1,}')

**Count Vectorizer:** Forming a Count Vectorizer (Converts a collection of text documents to a matrix of token counts) by combining the test and the train sets.

```
# Define all_text from entire train & test data for use in tokenization by Vectorizer
train_text = train_df['comment_text']
test_text = test_df['comment_text']
all_text = pd.concat([train_text, test_text])

# Vectorize the data
# import and instantiate CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
word_vect = CountVectorizer(
    strip_accents='unicode',
    analyzer='word',
    token_pattern=r'\w{1,}',
    stop_words='english',
    ngram_range=(1, 1)
)

# learn the vocabulary in the training data, then use it to create a document-term matrix
word_vect.fit(all_text)
```

Fitting the vectorizer, transforming the vocabulary learned into a document term matrix and saving the vectorizer in pickle or .pkl format.

```
# transform the data using the earlier fitted vocabulary, into a document-term matrix
train_features = word_vect.transform(train_text)
test_features = word_vect.transform(test_text)
```

```
# saving word vectorizer vocab as pkl file to be loaded afterwards
pickle.dump(word_vect.vocabulary_, open('word_feats.pkl', 'wb'))
```

```

pickle.dump(word_vect.vocabulary_,open('word_feats.pkl','wb'))

from sklearn.linear_model import LogisticRegression    Import "sklearn.linear_model" could not be resolve
from sklearn.metrics import accuracy_score    Import "sklearn.metrics" could not be resolved from source
logreg=LogisticRegression(C=16.0)

submission_binary =pd.read_csv('submission_binary.xls')

print(test_df.columns)

Index(['id', 'comment_text', 'toxic', 'severe_toxic', 'obscene', 'threat',
       'insult', 'identity_hate'],
      dtype='object')

```

## Activity 6: Model Building

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms can be chosen according to the objective.

**Choose the appropriate model:** Working with the Logistic Regression model Building a multi-label classifier using Logistic Regression. For each target variable, the model is fitted and saved.

```

# Binary Relevance - build a multi-label classifier using Logistic Regression
# Model Building and Fitting

# import and instantiate the Logistic Regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logreg = LogisticRegression(C=12.0)

```

```

mapper = {}
for label in cols_target:
    mapper[label] = logreg
    filename = str(label+'_model.sav')
    print(filename)
    print('... processing {}'.format(label))
    y= test_df[label]
    mapper[label].fit(test_features,y)
    pickle.dump(mapper[label],open(filename,'wb'))
    y_pred_X=mapper[label].predict(test_features)
    print('Training accuracy is {}'.format(format(accuracy_score(y,y_pred_X))))
    test_y_prob = mapper[label].predict_proba(train_features)[:,:1]
    submission_binary[label] = test_y_prob

```



```

... toxic_model.sav
... processing toxic
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Training accuracy is 0.9674878267354344
severe_toxic_model.sav
... processing severe_toxic
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Training accuracy is 0.9922542316586347
threat_model.sav

```

```

... processing threat
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Training accuracy is 0.9980447575060631
obscene_model.sav
... processing obscene
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Training accuracy is 0.9806543795551823
insult_model.sav
... processing insult

```

```

C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Training accuracy is 0.9728710103966259
identity_hate_model.sav
... processing identity_hate
Training accuracy is 0.9924234353359946
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(

```

```

submission_binary.to_csv('submission_binary.csv',index=False)

# Clean the input comment
comment = " bad"
cleaned_comment = clean_text(comment)

# Transform the cleaned comment using the same vectorizer used during training
comment_features = word_vect.transform([cleaned_comment]) # Make sure to pass a list

# Load the models and get predictions for each label
predictions = {}
for label in cols_target:
    # Load the pre-trained model for each label
    model = pickle.load(open(f'{label}_model.sav', 'rb'))

    # Get the probability for the positive class (index 1)
    prob = model.predict_proba(comment_features)[: , 1]

    # Store the probability in the predictions dictionary
    predictions[label] = prob[0]

```

```

# Get the probability for the positive class (index 1)
prob = model.predict_proba(comment_features)[: , 1]

# Store the probability in the predictions dictionary
predictions[label] = prob[0]

# Print the predicted probabilities for each target
for label, prob in predictions.items():
    print(f"{label}: {prob:.4f}")

```

```

toxic: 0.0674
severe_toxic: 0.0181
threat: 0.0023
obscene: 0.0467
insult: 0.0530
identity_hate: 0.0215

```

## Activity 7: Application Building

We use HTML to create the front-end part of the web page.

- Here, we have created 3 HTML pages- home.html, predict.html, and result.html
- home.html displays the home page.
- Predict.html file provides a user-friendly interface for submitting comments to be classified as toxic or non-toxic.
- The result.html file displays the classification result of the submitted comment, indicating whether it is toxic or non-toxic. We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages. Link:CSS, JS

### Create toxic\_com.py (Python Flask) file: -

Write the below code in Flask app.py python. The toxic\_com.py file contains the code for processing and predicting the toxicity of comments.

```
sk > toxic_com.py > ...
1  from flask import Flask,render_template,request
2  import pandas as pd
3  import re
4  from sklearn.feature_extraction.text import CountVectorizer
5  import pickle
6
7  loaded = CountVectorizer(decode_error='replace',vocabulary=pickle.load(open(r"word_feats.pkl",'rb')))
8
9  app = Flask(__name__)
10
11 def clean_text(text):
12     if text is None:
13         return ""
14     text = str(text) # Ensure it's a string before proceeding
15     text = text.lower()
16     text=re.sub(r"what's","what is",text)
17     text=re.sub(r"\'s"," ",text)
18     text=re.sub(r"\'ve","have",text)
19     text=re.sub(r"can't","cannot",text)
20     text=re.sub(r"n't","not",text)
21     text=re.sub(r"i'm","i am",text)
22     text=re.sub(r"\'re","are",text)
23     text=re.sub(r"\'d","would",text)
24     text=re.sub(r"\'ll","will",text)
25     text=re.sub(r"\'seuse","excuse",text)
```

```
def clean_text(text):
    text=re.sub(r"\W"," ",text)
    text=re.sub(r"\S+"," ",text)
    text=text.strip(' ')
    return text

@app.route('/')
def homepage():
    flag=0
    return render_template('home.html')

@app.route('/predict',methods=['GET','POST'])
def predict():
    if request.method=='GET':
        return render_template()
    if request.method=='POST':
        comment = request.form['comment']
        new_row={'comment_text':comment}
        user_df=pd.DataFrame(columns=['comment_text'])
        user_df = pd.concat([user_df, pd.DataFrame([new_row])], ignore_index=False)
        user_features = loaded.transform(user_df['comment_text'])
        cols_target=['insult','toxic','severe_toxic','identity_hate','threat','obscene']
        lst =[]
        mapper ={}
        for i in range(len(cols_target)):
            if cols_target[i] in mapper:
                mapper[cols_target[i]] += user_features[i].sum()
            else:
                mapper[cols_target[i]] = user_features[i].sum()
        return mapper
```

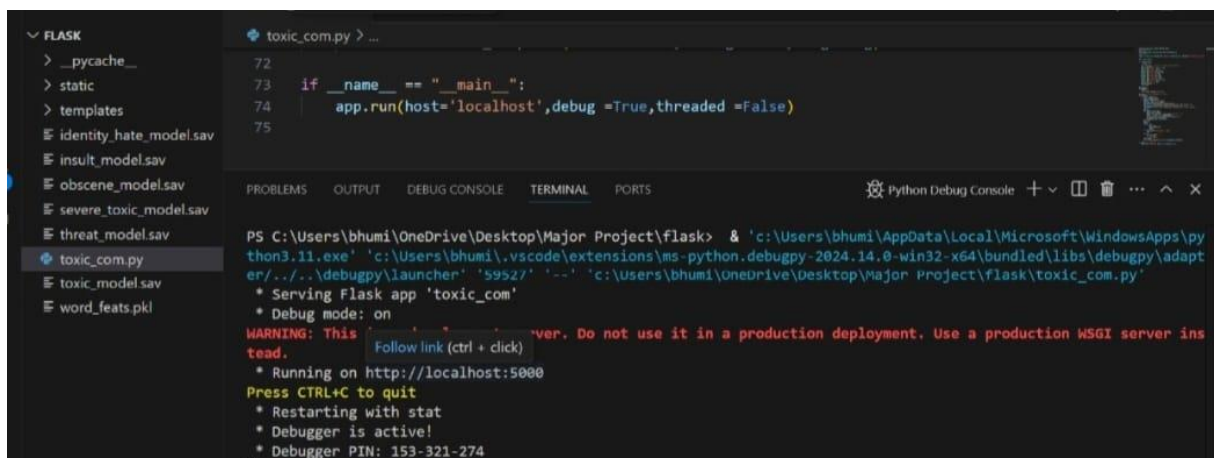
```

for label in cols_target:
    mapper[label] = loaded
    model=pickle.load(open(f'flask\{label}_model.sav','rb'))
    print('...processing {}'.format(label))
    user_y_prob=model.predict_proba(user_features)[: ,1]
    print(label,":",user_y_prob[0])
    lst.append([label,user_y_prob])

print(lst)
final =[]
flag=0
for i in lst:
    if i[1]> 0.5:
        final.append(i[0])
        flag=2
if not len(final):
    text ="yaayy !!the comment is clean"
    flag= 1
else:
    text="the comment is "
    for i in final:
        text=text+i+" "
print(text)
return render_template('toxic.html',message=text,flag=flag)

```

Getting local host in the terminal while running toxic\_com.py:

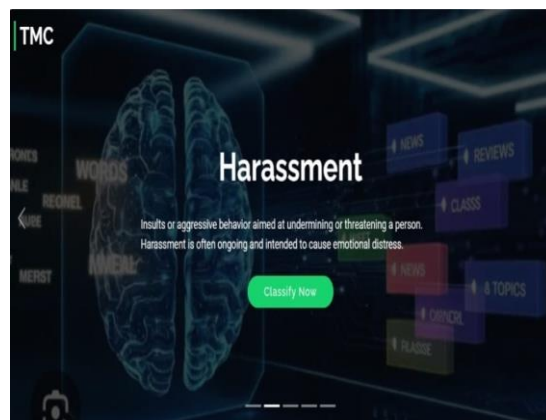
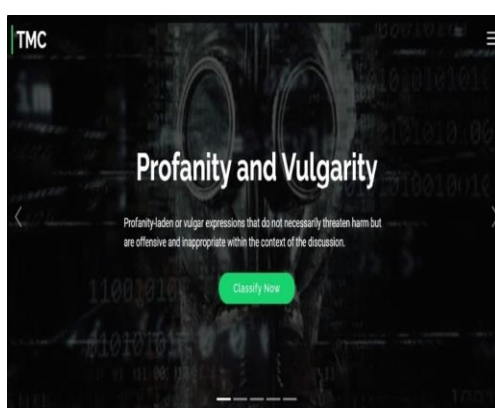


```

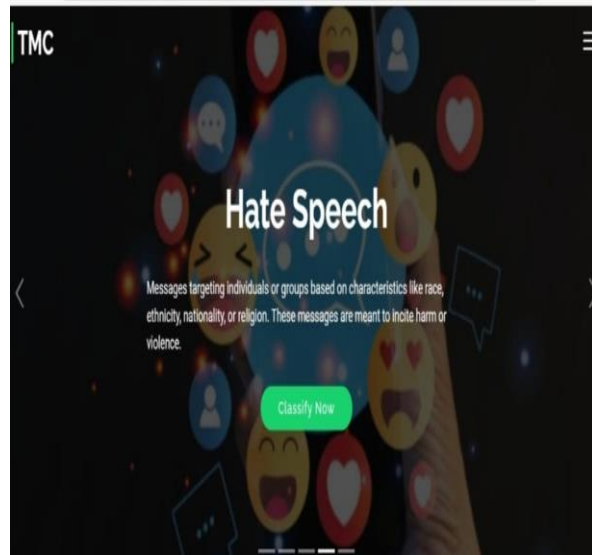
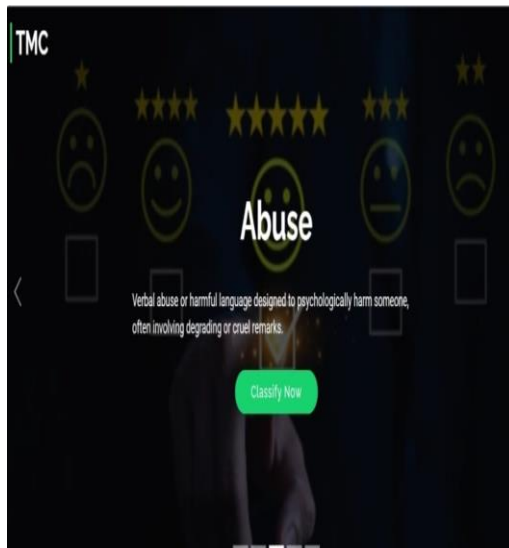
PS C:\Users\bhumi\OneDrive\Desktop\Major Project\flask> & 'c:\Users\bhumi\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\bhumi\.vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '59527' '-.' 'c:\Users\bhumi\OneDrive\Desktop\Major Project\flask\toxic_com.py'
* Serving Flask app 'toxic_com'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 153-321-274

```

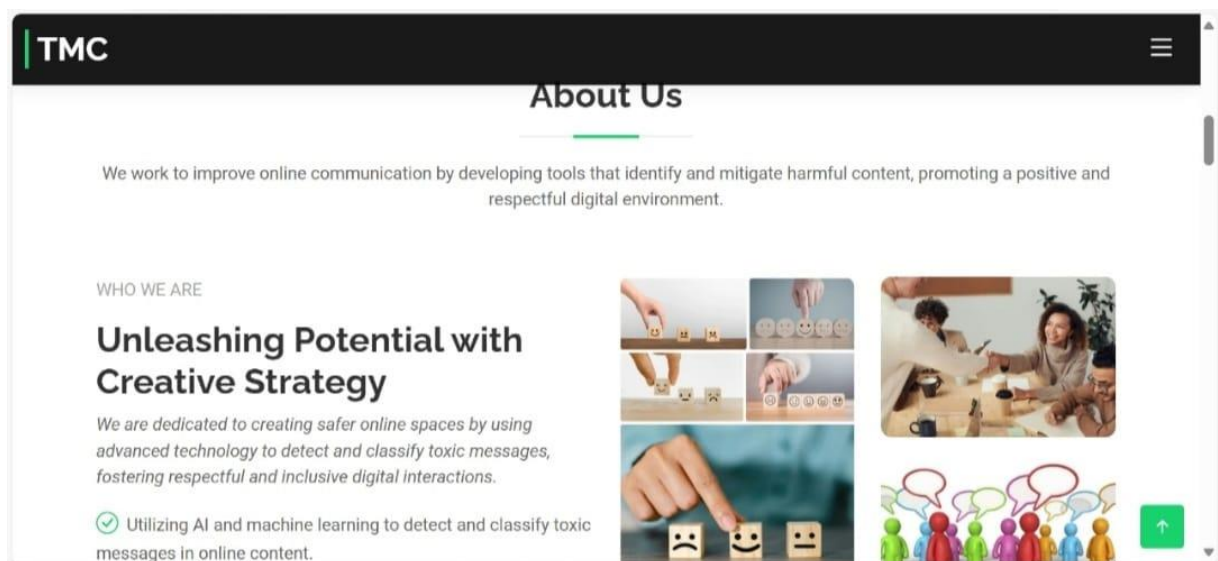
Index.html is displayed below:



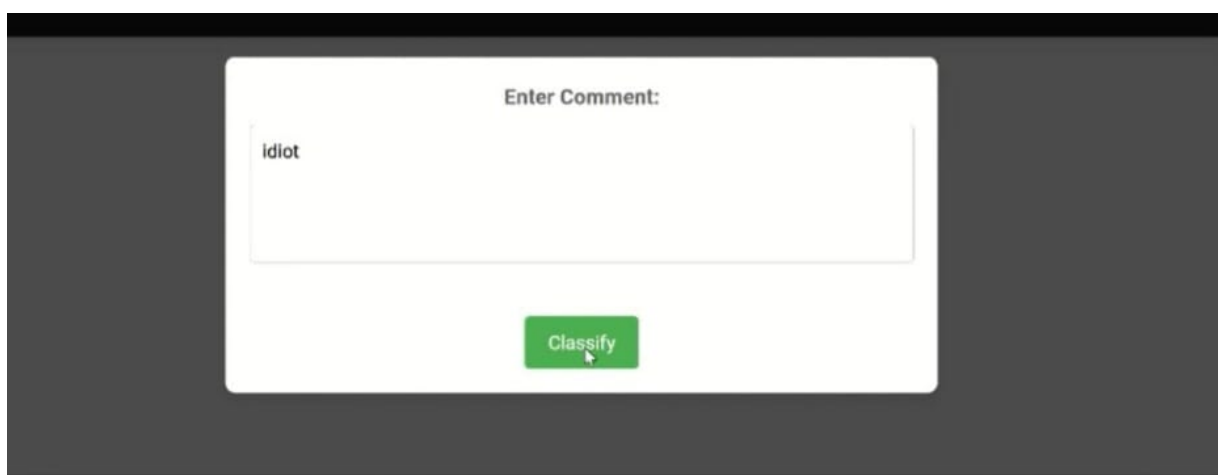
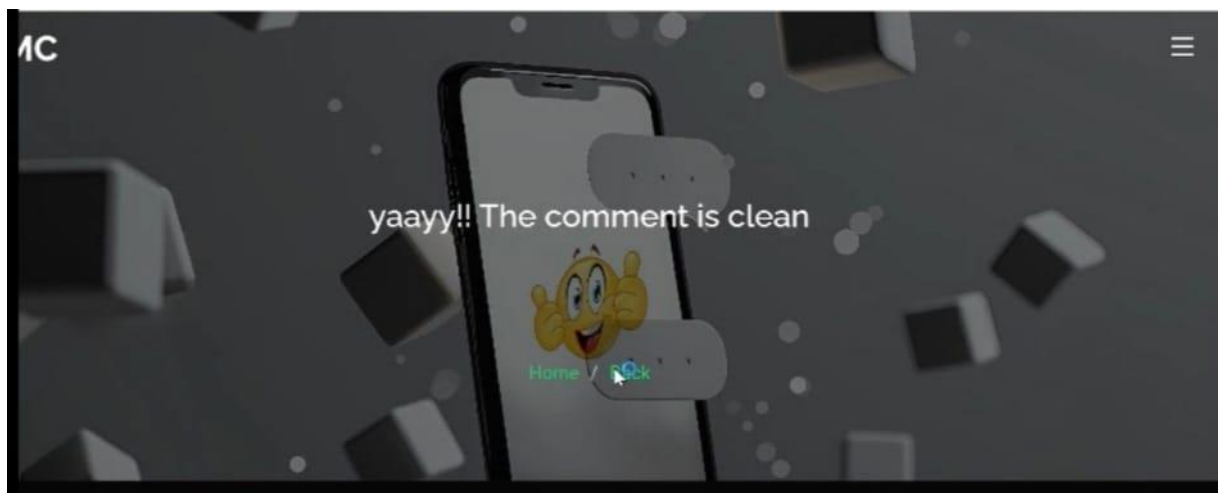
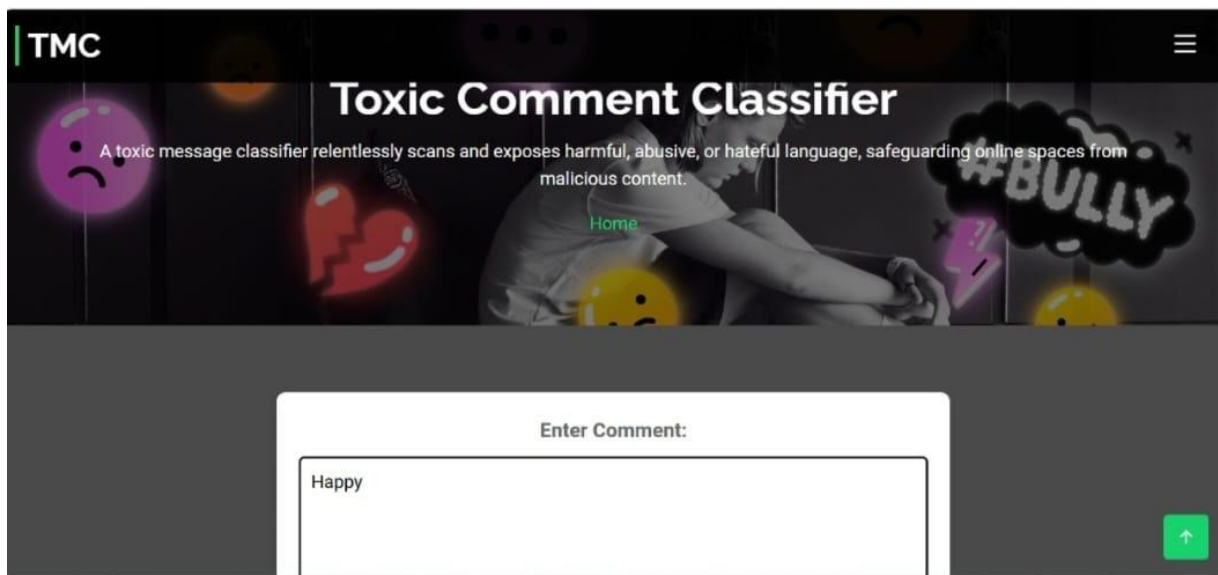


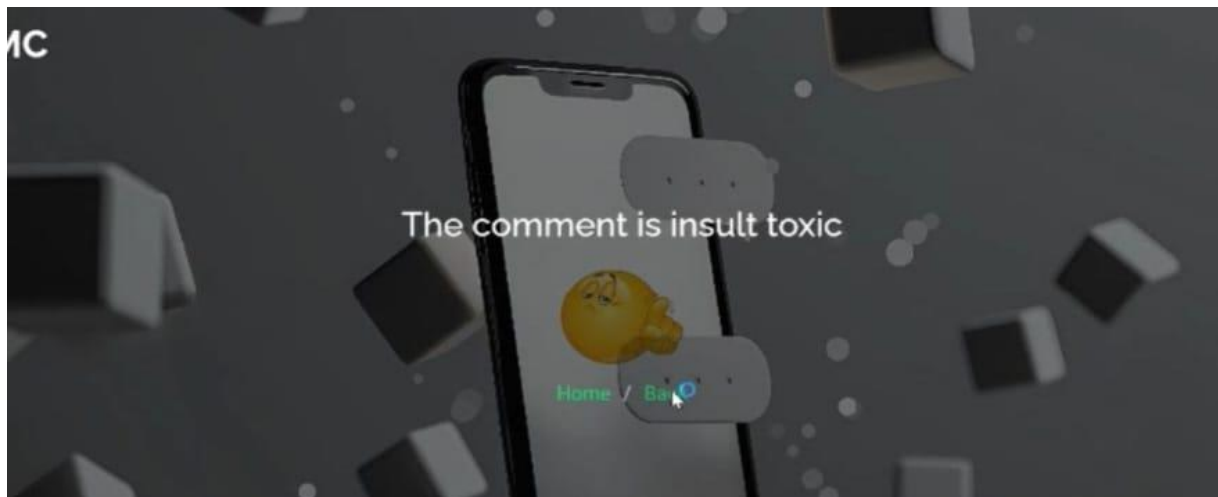


About Section is displayed below:



## Output





## **Advantages & Disadvantages**

### **Advantages of Toxic Comment Classification for Social Media using NLP:**

**Improved User Experience:** Reduces exposure to harmful and abusive language, creating a safer and more inclusive online environment.

**Scalability:** Automates the moderation process, handling large volumes of comments efficiently.

**Consistency:** Ensures uniform enforcement of community guidelines without human bias.

**Real-time Detection:** Provides immediate identification and flagging of toxic comments, enabling quick action.

### **Disadvantages of Toxic Comment Classification for Social Media using NLP:**

**False Positives/Negatives:** May incorrectly classify comments, leading to either unnecessary censorship or missed toxic content.

**Evolving Language:** Requires constant updates to adapt to new slang and evolving toxic language patterns.

**Context Understanding:** Struggles with nuanced understanding of context, sarcasm, and cultural references.

**Resource Intensive:** Requires significant computational resources for training and maintaining models.

## **Conclusion**

Toxic Comment Classification for Social Media using NLP is a powerful tool for creating safer and more inclusive online environments. By leveraging advanced NLP techniques and machine learning models, we can automatically detect and flag harmful comments, reducing the burden on human moderators and ensuring consistent enforcement of community guidelines.

This technology not only enhances user experience by minimizing exposure to toxic content but also adapts to evolving language patterns, ensuring long-term effectiveness. While challenges such as false positives, context understanding, and resource requirements exist, continuous improvements and updates can mitigate these issues.

Overall, this approach significantly contributes to maintaining healthy and respectful interactions on social media platforms, fostering a more positive and constructive digital community.

## **Future Scope**

**Advanced Language Understanding:** Development of more sophisticated models that can better understand context, sarcasm, and cultural nuances to reduce false positives and negatives.

**Multilingual Support:** Expansion to support multiple languages, making the technology effective across diverse global communities.

**Integration with Social Platforms:** Seamless integration with various social media platforms for real-time moderation and user reporting features.

**Hybrid Models:** Combining machine learning with rule-based systems to enhance accuracy and flexibility in identifying toxic content.



**User Customization:** Allowing users to set personal filters and sensitivity levels for toxic content, tailoring the experience to individual preferences.

**Continuous Learning:** Implementing adaptive systems that continuously learn from new data to stay updated with evolving language and slang.

**Enhanced Privacy and Security:** Ensuring data privacy and security while implementing such models to build user trust.

**Collaborative Moderation:** Engaging with community moderators to provide better tools and insights, creating a collaborative approach to content moderation.

## Appendix

### Source Code:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import re
import pickle
train_df= pd.read_csv('test.xls')
test_df = pd.read_csv('train.xls')
cols_target = ['toxic','severe_toxic','threat','obscene','insult','identity_hate']
print(test_df.isnull().any())
data = test_df[cols_target]
colormap = plt.cm.plasma
plt.figure(figsize=(7,7))
plt.title('correlation of features & targets',y=1.05,size=14)
sns.heatmap(data.astype(float).corr(),linewidths=0.1,vmax=1.0,square
=True,cmap=colormap,linecolor='white',annot=True)
def clean_text(text):
    text = text.lower()
    text=re.sub(r"what's","what is ",text)
    text = re.sub(r"\s"," ",text)
```

```

text=re.sub(r"'ve","have",text)
text=re.sub(r"can't","cannot",text)
text=re.sub(r"n't","not",text)
text=re.sub(r"n't","not",text)
text=re.sub(r"i'm","i am",text)
text=re.sub(r"'re","are",text)
text=re.sub(r"'d","would",text)
text=re.sub(r"'ll","will",text)
text=re.sub(r"'scuse","excuse",text)
text=re.sub(' \w',' ',text)
text=re.sub(' \s+',' ',text)
text=text.strip(' ')
return text
train_df['comment_text'] = train_df['comment_text'].map(lambda com
: clean_text(com))
test_df['comment_text']=test_df['comment_text'].map(lambda com :
clean_text(com))
train_text = train_df['comment_text']
test_text = test_df['comment_text']
all_text = pd.concat([train_text, test_text])
from sklearn.feature_extraction.text import CountVectorizer
word_vect = CountVectorizer(strip_accents='unicode',analyzer='word',token_patter
n=r'\w{1,}','stop_words='english',ngram_range=(1,1))
word_vect.fit(all_text)
train_features = word_vect.transform(train_text)
test_features = word_vect.transform(test_text)
pickle.dump(word_vect.vocabulary_,open('word_feats.pkl','wb'))
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logreg=LogisticRegression(C=16.0)
submission_binary =pd.read_csv('submission_binary.xls')
print(test_df.columns)
mapper = {}
for label in cols_target:

```

```

mapper[label] = logreg
filename = str(label+'_model.sav')
print(filename)
print('... processing {}'.format(label))
y= test_df[label]
mapper[label].fit(test_features,y)
pickle.dump(mapper[label],open(filename,'wb'))
y_pred_X=mapper[label].predict(test_features)
print('Training accuracy is
{}'.format(format(accuracy_score(y,y_pred_X))))
test_y_prob = mapper[label].predict_proba(train_features)[: ,1]
submission_binary[label] = test_y_prob
submission_binary.to_csv('submission_binary.csv',index=False)
# Clean the input comment
comment = " bad"
cleaned_comment = clean_text(comment)

# Transform the cleaned comment using the same vectorizer used
during training
comment_features = word_vect.transform([cleaned_comment]) #
Make sure to pass a list

# Load the models and get predictions for each label
predictions = {}
for label in cols_target:
    # Load the pre-trained model for each label
    model = pickle.load(open(f'{label}_model.sav', 'rb'))

    # Get the probability for the positive class (index 1)
    prob = model.predict_proba(comment_features)[: , 1]

    # Store the probability in the predictions dictionary
    predictions[label] = prob[0]

# Print the predicted probabilities for each target

```

```
for label, prob in predictions.items():  
    print(f' {label}: {prob:.4f}')
```

**GITHUB LINK:** <https://github.com/Poojithadharla512/Toxic-Comment-Classification-for-Social-Media>

**Demo**

**Link:**<https://drive.google.com/file/d/1XTSyjf5ugCwmTFNjabOCvmppsriYfPRX/view?usp=sharing>