UNIVERSITY INSTITUTE OF TECHNOLOGY BARKATULLAH UNIVERSITY, BHOPAL

Department of Computer Science and Engineering

MINOR PROJECT

On

“Comment Classifier”

Submitted In Partial Fulfillment of the Requirement for the

Award of the Degree Of

Bachelor of Technology (B. Tech.)

Year 2023

Barkatullah University, Bhopal

Year 2020-2024

Submitted by:

VINAYAK MODI **ANKIT PATIL**

Under the Guidance

Of

Mr. Bhawani Singh Rathore
Project coordinator
CSE, UIT-BU, Bhopal

Mrs. Kavita Chourasia
Project coordinator
CSE, UIT-BU, Bhopal

Dr. Divakar Singh
Guide & HOD CSE
CSE, UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY BARKATULLAH UNIVERSITY, BHOPAL

Department of computer Science and Engineering

CERTIFICATE

YEAR-2023

This is to certify that minor project entitled **"Comment Classifier"** submitted by **Vinayak Modi and Ankit Patil** in their partial fulfillment of the requirement for the award of degree of Bachelor of Engineering in Computer Science and Engineering, 6th Semester from Barkatullah University Institute of Technology, Bhopal.

Mr. Bhawani Singh Rathore	**Mrs. Kavita Chourasia**
Projector Coordinator	Projector Coordinator
CSE, UIT-BU, Bhopal	CSE, UIT-BU, Bhopal
Dr. Divakar Singh	**Prof. N. K. Gaur**
Guide & HOD CSE	DIRECTOR
UIT-BU, Bhopal	UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY
BARKATULLAH UNIVERSITY, BHOPAL

Department of computer Science and Engineering

DECLARATION

Year 2023

I hereby declare that the work presented in this dissertation work entitled **"Comment Classifier"** in python submitted for the degree of Bachelor of Engineering at the department of Computer Science and Engineering from University of Institute Technology, Barkatullah University, Bhopal, MP for the academic year 2023 Bhopal is in authentic record of my project work.

I declare that the work presented in this dissertation is original, it has not been submitted for any degree in any other universities.

Vinayak Modi **Ankit Patil**

Date- ___

ACKNOWLEDGEMENT

It gives me a great sense of pleasure to present the report of the project work Undertaken during B.Tech. Third Year. I own special debt of gratitude to my project Coordinator **Dr. Divakar Singh**, Department of Computer Science and Engineering, BUIT, Bhopal for his constant support and guideline throughout the course of my work. It is only his cognizant efforts that my endeavors have seen light of the day, my deepest thanks to my Project Guide **Mr. Bhawani Singh Rathore, Asst. Prof CSE-BUIT, Mrs. Kavita Chourasia, CSE-BUIT,** the Guide of the project for guiding and correcting various documents of mine with attention and care. I also take the opportunity to acknowledge the contribution of **Prof. N.K.Gaur, Director, BUIT, Bhopal.**

I also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of my project. Last but not the least, I acknowledge my friends for their contribution in the completion of the project.

Submitted By:

Vinayak Modi

Ankit Patil

ABSTRACT

This project **“Comment Classifier”** is a machine learning model designed to analyze and categorize comments based on their content. It plays a crucial role in automating the process of understanding the sentiment, intent, or topic of a comment, which can be beneficial in a wide range of applications such as social media analysis, content moderation, and customer feedback management. By leveraging Python, Scikit Learn, Natural Language Processing (NLP) techniques, Decision Tree, Gradio(GUI), Count Vectorizer, a comment classifier can accurately classify comments as Abusive, Offensive, or No abusive or offensive, allowing for efficient processing and organization of large volumes of user-generated content. This enables businesses to gain valuable insights from customer feedback, identify potential issues or trends, and respond promptly to user concerns.

Additionally, Comment Classifiers are instrumental in content moderation by automatically detecting and filtering out inappropriate or offensive comments, enhancing the overall user experience and maintaining a safe online environment.

TABLE OF CONTENTS **Page No.**

Chapter 1

INTRODUCTION

INTRODUCTION

A comment classifier is a machine learning model that analyzes and categorizes comments based on their content or sentiment. It is trained to automatically assign comments to predefined categories, such as Abusive, Offensive, or No abusive or offensive. Comment classifiers help automate comment management and moderation tasks, making it easier to organize and handle large volumes of comments in online platforms, social media, and customer service systems.

Comment classifiers are trained on a labeled dataset, where each comment is assigned a specific category or sentiment label. The classifier learns patterns and features from the labeled data and uses them to make predictions on new, unlabeled comments. Comment classifiers can save significant time and effort by automatically identifying and flagging inappropriate, offensive, or spam comments. They can also be used for sentiment analysis, extracting insights from user feedback, and improving overall user experience by efficiently addressing user concerns or issues.

Overall, comment classifiers significantly improve the efficiency of comment management, content moderation, and user engagement by automating the analysis and categorization of comments. They play a crucial role in maintaining a positive user experience and ensuring the quality of online discussions and interactions.

The comment classifier is a machine learning model that automates the analysis and categorization of comments based on their content or sentiment. It plays a crucial role in various applications, such as online platforms, social media, and content moderation systems. The objective of the comment classifier is to streamline comment management and moderation by accurately assigning comments to predefined categories.

1.1 PURPOSE / OBJECTIVE:

The purpose of a comment classifier is to automate the process of analyzing and categorizing comments based on their content or sentiment. By accurately classifying comments into predefined categories, such as Abusive, Offensive, or No abusive or offensive, comment classifiers facilitate efficient comment management and moderation. Platforms and businesses can save time and effort by automating the identification and handling of inappropriate, offensive, or spam comments, ensuring a safe and respectful environment for users. Comment classifiers also contribute to enhancing user engagement and experience by promptly addressing user concerns or issues. Additionally, comment classifiers provide valuable insights through sentiment analysis, enabling platforms to understand user preferences, identify trends, and make data-driven decisions. Overall, the objective of a comment classifier is to streamline comment management, improve content moderation, and gain valuable insights for enhancing user satisfaction and platform performance.

1.2 EXISTING SYSTEM:

Existing comment classifiers are often used in various applications, including social media platforms, online forums, content moderation systems, and customer service platforms. They help automate comment management, filter out spam or offensive content, and improve overall user experience by efficiently addressing user feedback or concerns. These systems are continually refined and updated to adapt to evolving user behaviors, emerging patterns, and new types of comment content.

Comment classifiers are trained on labeled datasets, where each comment is associated with a predefined category or sentiment. Supervised learning algorithms like decision trees, random forests, support vector machines (SVM), or neural networks are commonly used for training the classifier. The performance of the comment classifier is evaluated using various metrics, such as accuracy, precision, recall, and F1 score. These metrics assess the classifier's ability to correctly predict the category or sentiment of comments.

Some comment classifiers are designed to incorporate continuous learning techniques. This allows the classifier to adapt and improve its performance over time by incorporating new labeled data or adjusting its model parameters.

1.3 PROPOSED SYSTEM:

The comment classifier is a machine learning model that automates the analysis and categorization of comments based on their content or sentiment. It plays a crucial role in various applications, such as online platforms, social media, and content moderation systems. By accurately classifying comments into predefined categories, such as Abusive, Offensive, or No abusive or offensive, the objective of the comment classifier is to streamline comment management and moderation by accurately assigning comments to predefined categories.

It utilizes machine learning algorithms and techniques to learn from labeled training data. It extracts features from the comment text, such as word frequencies or semantic information, and trains a model to predict the category or sentiment of unseen comments. This automated process improves the efficiency of handling large volumes of user-generated comments.

Benefits:

- Improved user experience and engagement: By effectively filtering out spam, offensive, or irrelevant comments, the classifier will enhance the overall user experience, encourage meaningful discussions, and foster a safe and positive online environment.

- Enhanced content moderation: The multi-label classifier will provide more granular and accurate categorization of comments, enabling effective identification and handling of various content types, such as offensive language, hate speech, or promotional content.

- Efficient comment management: The classifier will automate the process of organizing and filtering comments based on their labels, allowing for efficient prioritization and handling of different comment categories. This will streamline comment management and improve response times.

- Actionable insights: The classifier will provide valuable insights into user sentiment, preferences, and trending topics through the analysis of comments. These insights can be utilized to refine marketing strategies, optimize content creation, or identify emerging issues.

- Real-time Analysis: Comment classifiers can analyze comments in real-time, enabling immediate responses or actions to address user concerns, identify emerging trends, or detect potential issues promptly.

1.4 FEASIBILTY REPORT:

The feasibility of a comment classifier depends on various factors, including the availability of labeled training data, computational resources, and the complexity of the classification task.

Here are some key considerations regarding the feasibility of a comment classifier:

- **Availability of Labeled Data:** Building an effective comment classifier typically requires a substantial amount of labeled training data.

- **Feature Representation:** Comment classifiers utilize various feature representations to capture the relevant information from the comment text. The feasibility of the classifier depends on the availability of suitable feature representation.

- **Computational Resources:** Training a comment classifier, especially with large datasets or complex models, can require significant computational resources.

- **Scalability and Real-Time Processing:** The feasibility of a comment classifier also depends on its ability to scale to handle large volumes of comments and perform real-time processing.

- **Model Selection and Complexity:** Choosing an appropriate machine learning model is crucial for the feasibility of the comment classifier. The model should strike a balance between accuracy and computational efficiency requirements.

- **Evaluation and Validation:** Proper evaluation and validation of the comment classifier are essential to assess its feasibility.

Considering these factors and conducting a thorough feasibility analysis can help determine the viability and potential success of a comment classifier project. It is essential to assess the available resources, data availability, computational capabilities, and the specific requirements of the target application to ensure a feasible and effective comment classifier solution.

Chapter 2

Requirements

CHAPTER-2

REQUIREMENTS

2.1 SOFTWARE REQUIREMNTS:

- Operating system: Windows 10 Home
- IDE : PYCHAM 2023.1.1
- GUI: Gradio 3.30.3
- Programming Language: Python 3.11.3
- Python Libraires: NLTK, gradio, pandas, numpy, re, sklearn, nltk.corpus.stopwords, string.
- Web Browser: Google Chrome

2.2 HARWARE REQUIREMNTS:

- Processor: Intel Core i3 or higher, AMD processor
- Storage: 10 GB Hard Disk or 500 GB SSD
- RAM: Min. 4GB or higher

Chapter 3

Project Description

Chapter 3

Technical Description

3. Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
Python is the primary programming language used in the provided code for the comment classifier.

Here are the key uses of Python in the project:

- **Data Manipulation:** Python, along with the pandas library, is used to load the dataset from a CSV file. It provides convenient data structures and methods for handling tabular data, allowing the code to read and process the comment data effectively.

- **Data Preprocessing:** Python is utilized to clean and preprocess the comment text. The re module is used for regular expression-based text cleaning, while the nltk library provides functionalities for tokenization, stop word removal, and word stemming. Python's string manipulation capabilities are employed to perform text transformations and remove unwanted characters.

- **Machine Learning:** Python, combined with the scikit-learn library, is used to implement the comment classifier. The code utilizes the Decision Tree Classifier class from scikit-learns tree module to train a decision tree-based model. Python is also used to split the dataset into training and testing sets, extract features using Count Vectorizer, and evaluate the performance of the classifier.

- **Gradio Interface:** Python, along with the Gradio library, is employed to create a user interface for the comment classifier. The code defines the input and output interfaces using Gradio's Textbox and Label components, respectively. Python is also used to define the classify comment function, which performs the classification based on user input.

- **General Programming:** Python serves as the general-purpose programming language for writing the overall logic of the code. It handles tasks such as function definitions, variable assignments, flow control, and interactions with external resources.

Python's simplicity, extensive libraries, and readability make it well-suited for tasks such as data manipulation, preprocessing, machine learning, and creating user interfaces. Its ecosystem of libraries and frameworks provides efficient solutions for various aspects of the comment classifier project.

3.1 PYTHON LIBRAIES USE IN PROJECT:

The libraries used in the provided code are:

- gradio: Used for creating the user interface to interact with the comment classifier.
- pandas: Used for data manipulation and analysis, particularly for loading and handling the dataset.
- numpy: Used for numerical operations and array manipulation.
- sklearn.feature_extraction.text.CountVectorizer: Used for converting text data into numerical features.
- sklearn.model_selection.train_test_split: Used for splitting the dataset into training and testing sets.
- sklearn.tree.DecisionTreeClassifier: Used for training the decision tree classifier.
- re: Used for regular expression operations, specifically for cleaning the comment text.
- nltk: Used for natural language processing tasks, such as downloading stopwords and stemming.
- nltk.corpus.stopwords: Used for obtaining a set of common stopwords.
- `string`: Used for working with string operations and punctuation removal.

Please make sure you have these libraries installed before running the code. You can use `pip install` to install any missing libraries.

3.2 Decision Tree:

A **Decision Tree** is a flowchart-like representation of data that graphically resembles a tree that has been drawn upside down. In this analogy, the root of the tree is a decision that has to be made, the tree's branches are actions that can be taken and the tree's leaves are potential decision outcomes.

In the code for the **Comment Classifier**, a **decision tree algorithm** is used as the machine learning model for classification

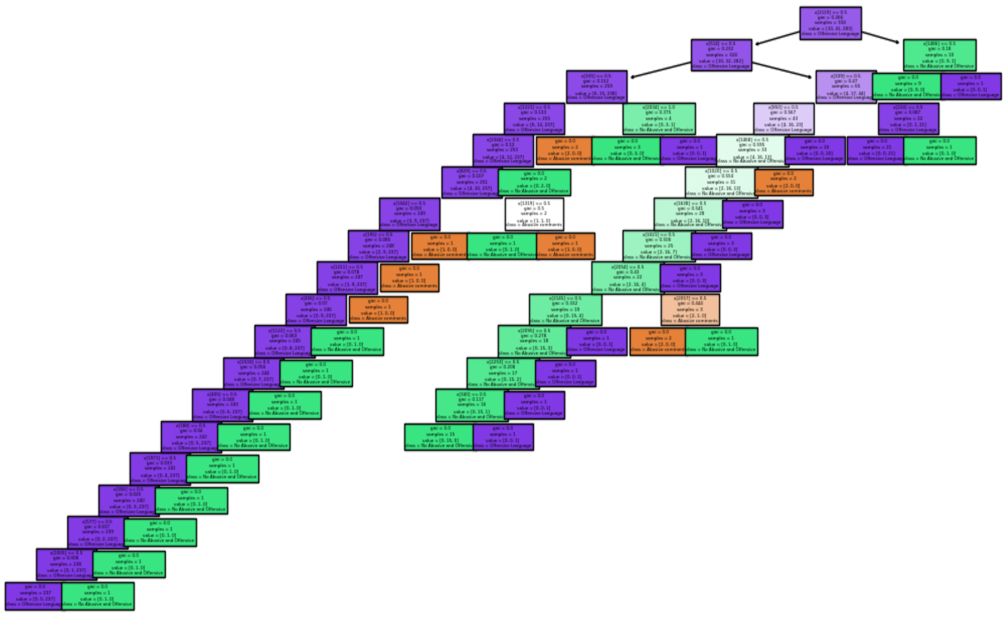


Fig 1: COMMENT CLASSIFIER DECISION TREE

The decision tree algorithm uses a tree-like model of decisions and their possible consequences. It learns patterns and relationships within the input features (comment text) to make predictions about the class labels (categories of comments).

The algorithm splits the data based on feature values and creates decision nodes that lead to subsequent nodes or leaf nodes representing class labels. The training process involves finding the optimal splits and creating the decision tree model based on the provided training data.

The use of the decision tree algorithm:

- Importing the Required Libraries: The code includes the necessary import statement `from sklearn. Tree import **DecisionTreeClassifier`** to import the DecisionTreeClassifier class from the scikit-learn (sklearn) library. This class provides the implementation of the decision tree algorithm.

- Training the Decision Tree Classifier: The code instantiates an object of the **DecisionTreeClassifier** class as **`clf = DecisionTreeClassifier()`.** This creates an instance of the decision tree classifier, which will be used to learn from the training data.

- Fitting the Classifier to the Training Data: The code calls the `fit()` method of the **decision tree classifier (`clf.fit(X_train, y_train)`)** to train the model using the training data. The `X_train` variable represents the input features (comment text features) of the training data, and `y_train` represents the corresponding class labels.

- Prediction with the Classifier: The code defines the **`classify_comment(comment)`** function, which takes a comment as input. Inside this function, the comment is preprocessed and transformed into numerical features using the same preprocessing steps and **CountVectorizer** as the training data. Then, **the `predict()`** method of the decision tree classifier **(`clf.predict(vectorized_comment)`)** is used to predict the class label for the preprocessed comment.

The decision tree classifier in the code uses the features extracted from the comments to make predictions on new, unseen comments. It allows the comment classifier to classify comments into different categories based on the learned decision rules and patterns.

3.3 Natural Language Processing(NLP):

Natural Language Processing (NLP) techniques are used in the provided code for the comment classifier. It provide below the use of NLP in the **Comment Classifier:**

- **Text Cleaning:** NLP is used to clean and preprocess the comment text. The **`re`** module is used for regular expression-based text cleaning, removing unwanted characters, URLs, HTML tags, punctuation, and special characters. This step helps in standardizing the text data and removing noise.

- **Tokenization: NLP** is used for tokenization, which involves breaking down the comment text into individual words or tokens. The code uses Python's string manipulation capabilities to split the comment text into a list of tokens. Tokenization is a crucial step in text processing and feature extraction.

- **Feature Extraction: NLP** is used to extract features from the preprocessed comment text. The comment classifier utilizes the **CountVectorize**r class from the scikit-learn library, which converts the text data into numerical feature vectors. The **CountVectorizer** tokenizes the text, builds a vocabulary of unique words, and counts the occurrence of each word in each comment. This step transforms the comment text into a matrix of numerical features that can be used for machine learning.

- **Stopword Removal**: NLP is used to remove stopwords from the comment text. The code utilizes the **`stopwords`** corpus from the **NLTK (Natural Language Toolkit)** library, which contains a list of common **stopwords**. **Stopwords** are words that do not contribute much to the overall meaning of a sentence, such as "the," "is," and "and." Removing stopwords helps in reducing the dimensionality of the data and focusing on more meaningful words.

- **Word Stemming: NLP** is used for word stemming, specifically using the **SnowballStemmer** from the **NLTK library**. Stemming reduces words to their base or root form, allowing for normalization and collapsing similar variations of words into a single representation. It helps in capturing the core meaning of words and reducing the vocabulary size.

By incorporating NLP techniques, enhances the comment classifier's ability to process and understand the textual data. It handles tasks such as text cleaning, removing stopwords, tokenizing, stemming, and converting text into numerical representations. These NLP techniques contribute to improving the accuracy and effectiveness of the comment classifier.

3.4 Pandas:

The pandas library is used in the comment classifier code for various data manipulation tasks. Here's an explanation of the use of pandas in the code:

- Loading the Dataset: The code uses pandas to load the dataset from a CSV file. The `pd.read_csv()` function is used to read the CSV file and create a pandas DataFrame. The DataFrame is a tabular data structure that allows for easy handling and manipulation of the dataset.

- Data Cleaning and Manipulation: After loading the dataset, pandas is used to perform data cleaning and manipulation operations. For example, the code selects specific columns (`data[["comments", "labels"]]`) from the DataFrame, extracting only the necessary data for classification. The `data["labels"] = data["class"].map(...)` statement maps the numerical class labels to their corresponding textual representations using pandas' `map()` function.

- Data Preprocessing: pandas is involved in the data preprocessing steps as well. For instance, the `apply()` function from pandas is used to apply a cleaning function (`clean()`) to each comment in the "comments" column. This function performs text cleaning operations on the comments using regular expressions and string manipulations.

- Splitting the Data: pandas is utilized to split the dataset into training and testing sets. The `train_test_split()` function from the scikit-learn library is called, and pandas arrays (`X` and `y`) are passed as input to split the data into training and testing subsets based on a specified test size.

- Data Transformation: pandas is used to transform the comment data into a format suitable for machine learning. The `np.array()` function from the numpy library is used to convert the pandas Series objects (`data["comments"]` and `data["labels"]`) into numpy arrays (`x` and `y`). These arrays are then used for feature extraction and model training.

  Overall, pandas simplifies the data handling, cleaning, manipulation, and transformation tasks in the comment classifier code. It provides powerful data structures and functions that enable efficient data preprocessing and preparation for machine learning tasks.

3.5 Numpy:

The NumPy library is used in the comment classifier code to handle numerical operations and array manipulations efficiently.

The explanation of the use of NumPy in the comment classifier code:

- Numerical Computations: NumPy is utilized for numerical computations in the code. While the decision tree classifier (clf) is trained using scikit-learn, NumPy arrays are used to pass the input features (X_train, X_test) and class labels (y_train, y_test) to the classifier's fit() and predict() methods, respectively.

- Array Creation: NumPy is used to create arrays that store the comment text features (x) and the corresponding class labels (y). The code converts the pandas Series objects (data["comments"] and data["labels"]) into NumPy arrays using np.array().

- Array Manipulation: NumPy provides various functions and methods for array manipulation. In the code, NumPy is used to perform array operations like reshaping, splitting, and combining arrays. For instance, the code splits the data into training and testing sets using the train_test_split() function from scikit-learn, which accepts NumPy arrays as input.

- Data Manipulation: NumPy offers various methods for data manipulation, such as indexing, slicing, and filtering arrays. While the main data manipulation tasks in the code are performed using pandas, NumPy can be utilized for additional array operations if needed.

NumPy enhances the computational efficiency and convenience of array-based operations in the comment classifier code. It seamlessly integrates with other libraries, such as pandas and scikit-learn, to facilitate data handling, numerical computations, and array manipulations required in the classification process.

3.6 CountVectorizer:

The CountVectorizer is a class from the scikit-learn library that is used in the comment classifier code to convert the comment text into numerical feature vectors.

There is an explanation of the use of CountVectorizer in the code:

- **Creating Vocabulary**: The CountVectorizer builds a vocabulary of unique words from the comment text. It scans the entire corpus of comments and identifies all unique words, which become the columns or features of the resulting feature matrix.

- **Vectorization**: Once the vocabulary is created, the CountVectorizer converts each comment into a sparse matrix representation. The sparse matrix represents the count of each word from the vocabulary in the respective comment. The matrix is sparse because most comments only contain a subset of the vocabulary, and the majority of word counts are zero.

- **Handling Stopwords**: The CountVectorizer has an optional parameter to handle stopwords, which are common words that may not carry significant meaning in the classification task. In the code, stopwords are removed from the comment text before using CountVectorizer by utilizing the NLTK library's stopwords corpus.

- **Feature Extraction**: The primary purpose of CountVectorizer is to extract features from the comment text. This process converts the text data into numerical representations that machine learning algorithms can understand.

- **Preprocessing and Normalization**: The CountVectorizer allows for additional preprocessing and normalization options, such as converting all words to lowercase, removing punctuation, and applying stemming. In the code, some of these preprocessing steps are performed using regular expressions and the NLTK SnowballStemmer, while others can be set as parameters in the CountVectorizer constructor.

The CountVectorizer in the code plays a crucial role in converting the comment text into a numerical representation suitable for machine learning algorithms. It tokenizes the text, builds a vocabulary, and generates a sparse matrix that represents the occurrence of words in the comments. This allows the comment classifier to use the resulting feature matrix for training and predicting the comment classes.

3.7 Gradio(GUI):

Gradio is a Python library that provides a simple and intuitive interface for building and deploying interactive web-based interfaces for machine learning models. In the comment classifier code, Gradio is used to create the user interface for interacting with the trained classifier.

Gradio handles the communication between the UI and the model. It automatically converts user input into the appropriate format for the model, passes it to the model for prediction or processing, and displays the output in the UI. This makes it easy to create interactive applications where users can directly interact with the model and see the real-time results.

With Gradio, you can define various types of input components such as textboxes, dropdowns, checkboxes, and image uploaders to collect user input. You can also specify output components like labels, images, plots, or custom HTML elements to display the results of the machine learning model.

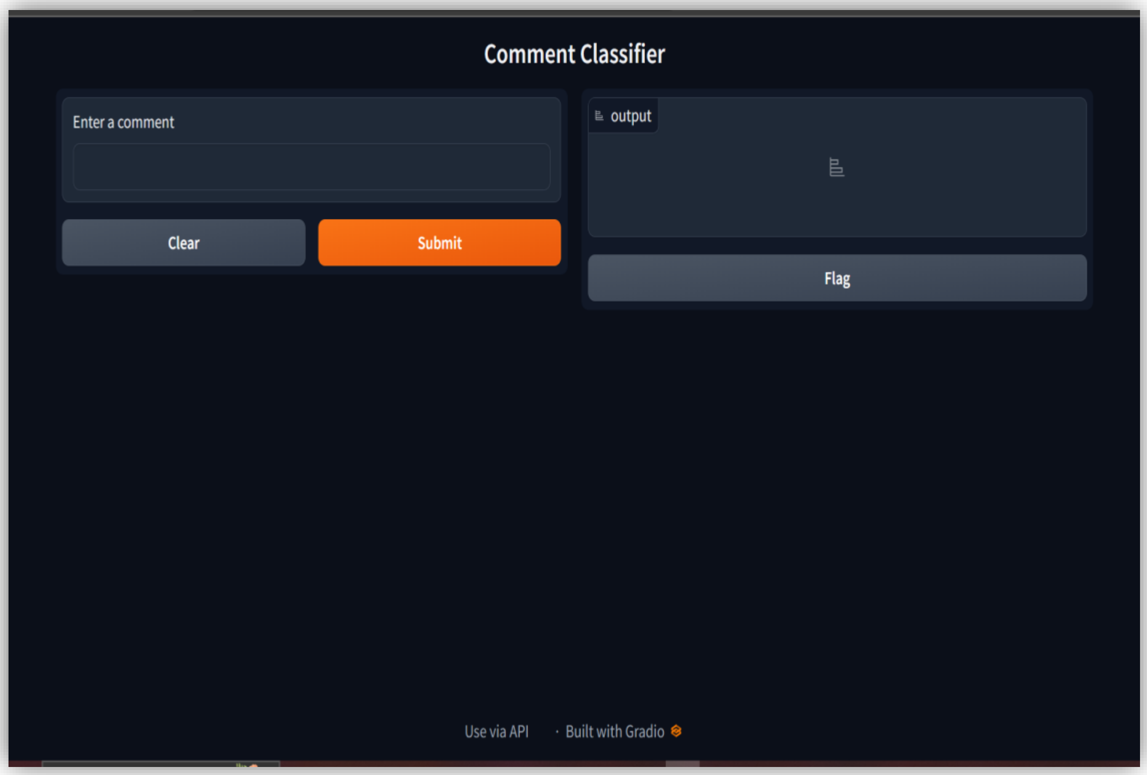


Fig 2. GUI using Gradio

The use of Gradio in the code:

- Input Interface: Gradio is used to create an input interface for entering a comment. The gr.inputs.Textbox class is used to define a text input box where users can enter their comments. This input interface allows users to interact with the comment classifier by providing their own text input for classification.

- Output Interface: Gradio is used to create an output interface for displaying the predicted class label. The gr.outputs.Label class is used to define the output area where the predicted class label is displayed. This output interface provides users with the classification result based on their input comment.

- Interface Function: Gradio is used to define the interface function that connects the input and output interfaces. The fn parameter in gr.Interface is set to the classify_comment function, which takes a comment as input, performs preprocessing and classification using the trained classifier, and returns the predicted class label.

- Launching the Interface: Gradio is used to launch the created interface. The launch() method is called on the interface object to start the web-based interface. This allows users to access the interface through a web browser and interact with the comment classifier in real-time.

Gradio simplifies the process of creating a user interface for the comment classifier. It provides the necessary tools to define the input and output interfaces, connect them to the classifier function, and launch the interface for users to interact with. Gradio enables easy deployment and usage of the comment classifier as a web-based application.

<u>Chapter 4</u>

<u>Code and Snapshot</u>

4.1 CODE:

```
#gradio for buliding GUI

import gradio as gr


#pandas used for data manipulation and analysis

import pandas as pd


#numpy used for numerical operations and array manipulation

import numpy as np


# CountVectorizer used for converting text data into numerical features.

from sklearn.feature_extraction.text import CountVectorizer


#used for splitting the dataset into training and testing sets.

from sklearn.model_selection import train_test_split


#decision tree used for training the decision tree classifier.

from sklearn.tree import DecisionTreeClassifier


#regular expression operations, for cleaning the comment text such as stopwords,
#specialcharacters(#<$%^&*@!#,.....etc).

import re


# NLTK for natural language processing tasks, such as downloading stopwords and stemming

import nltk
```

```
#importing stop words for removal from the dataset
from nltk.corpus import stopwords

#here string use for punction removal and working with the string.
import string

#use for plotting the decision tree
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("commentdataset.csv")

#labelling the data set with classifier classes according to which classifications has to perform
data["labels"] = data["class"].map({0: "Offensive Language", 1: "Abusive comments", 2: "No
Abusive and Offensive"})
data = data[["comments", "labels"]]

# Download NLTK resources
nltk.download('stopwords')
stopword = set(stopwords.words('english'))
stemmer = nltk.SnowballStemmer("english")

# Clean data
def clean(text):
    text = str(text).lower()
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"that's", "that is", text)

    text = re.sub(r"what's", "that is", text)

    text = re.sub(r"where's", "where is", text)

    text = re.sub(r"how's", "how is", text)

    text = re.sub(r"'ll", " will", text)

    text = re.sub(r"'ve", " have", text)

    text = re.sub(r"'re", " are", text)

    text = re.sub(r"i'm", "i am", text)

    text = re.sub(r"r", "", text)

    text = re.sub(r"he's", "he is", text)

    text = re.sub(r"'d", " would", text)

    text = re.sub(r"'re", " are", text)

    text = re.sub(r"won't", "will not", text)

    text = re.sub(r"can't", "cannot", text)

    text = re.sub(r"n't", " not", text)

    text = re.sub(r"n'", "ng", text)

    text = re.sub(r"'bout", "about", text)

    text = re.sub(r"'til", "until", text)

    text = re.sub('\[.*?\]', '', text)

    text = re.sub('https?://\S+|www\.\S+', '', text)

    text = re.sub('<.*?>+', '', text)

    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    text = re.sub('\n', '', text)

    text = re.sub('\w*\d\w*', '', text)

    text = [word for word in text.split(' ') if word not in stopword]

    text = " ".join(text)

    text = [stemmer.stem(word) for word in text.split(' ')]
    text = " ".join(text)
    return text
data["comments"] = data["comments"].apply(clean)
# Extract features
x = np.array(data["comments"])
y = np.array(data["labels"])
cv = CountVectorizer()
X = cv.fit_transform(x)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Train the classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Function to classify comments
def classify_comment(comment):
    cleaned_comment = clean(comment)
    vectorized_comment = cv.transform([cleaned_comment])
    prediction = clf.predict(vectorized_comment)
    return prediction[0]

# Create the input and output interfaces
comment_input = gr.inputs.Textbox(label="Enter a comment")
classification_output = gr.outputs.Label(num_top_classes=1)
```

```
# Create the Gradio interface

interface=gr.Interface(fn=classify_comment,inputs=comment_input,
outputs=classification_output, title="Comment Classifier")

interface.launch()


#plotting the decision tree of the comment classifier

plt.figure(figsize=(20,15))
plot_tree(clf, feature_names=cv.get_feature_names(), class_names=clf.classes_, filled=True)
plt.show()
```

4.2 Output Snapshot:

- **Output with abusive comment detection:**

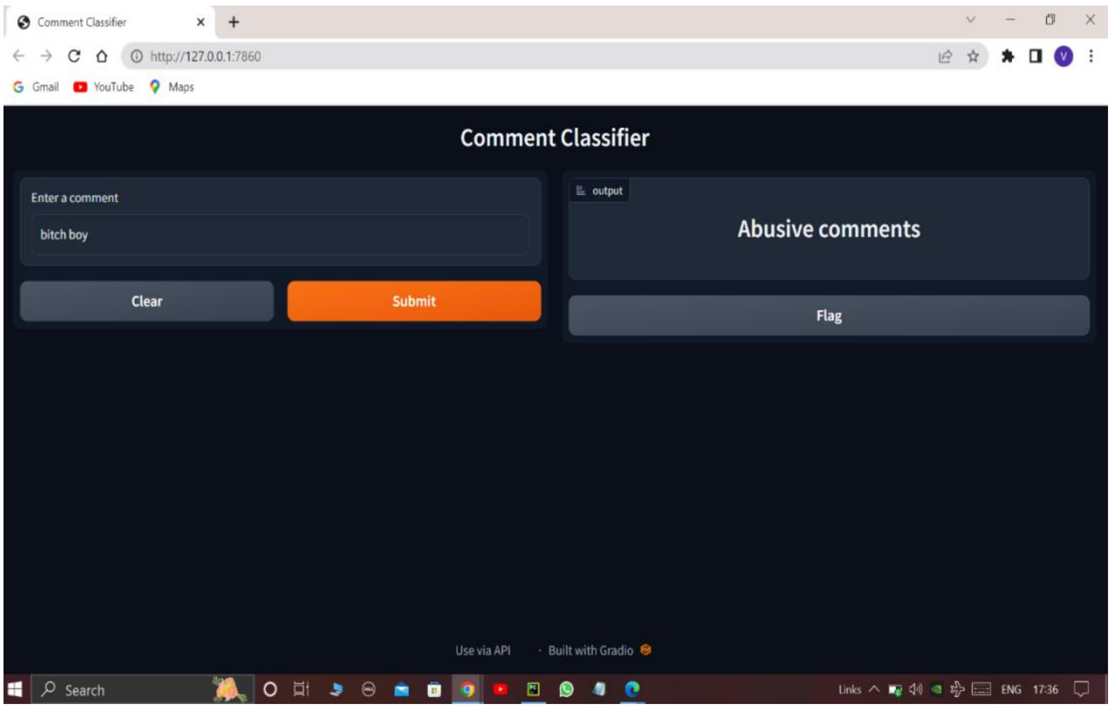


Fig 4.1 Abusive Comment

- **Output with Offensive comment detection:**

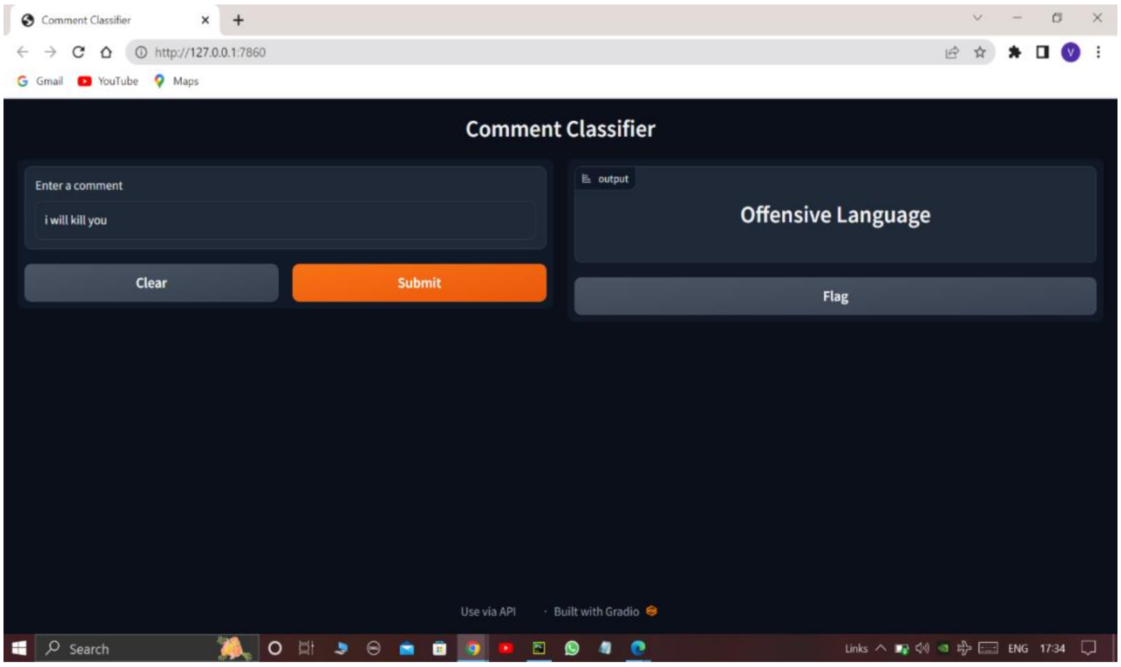
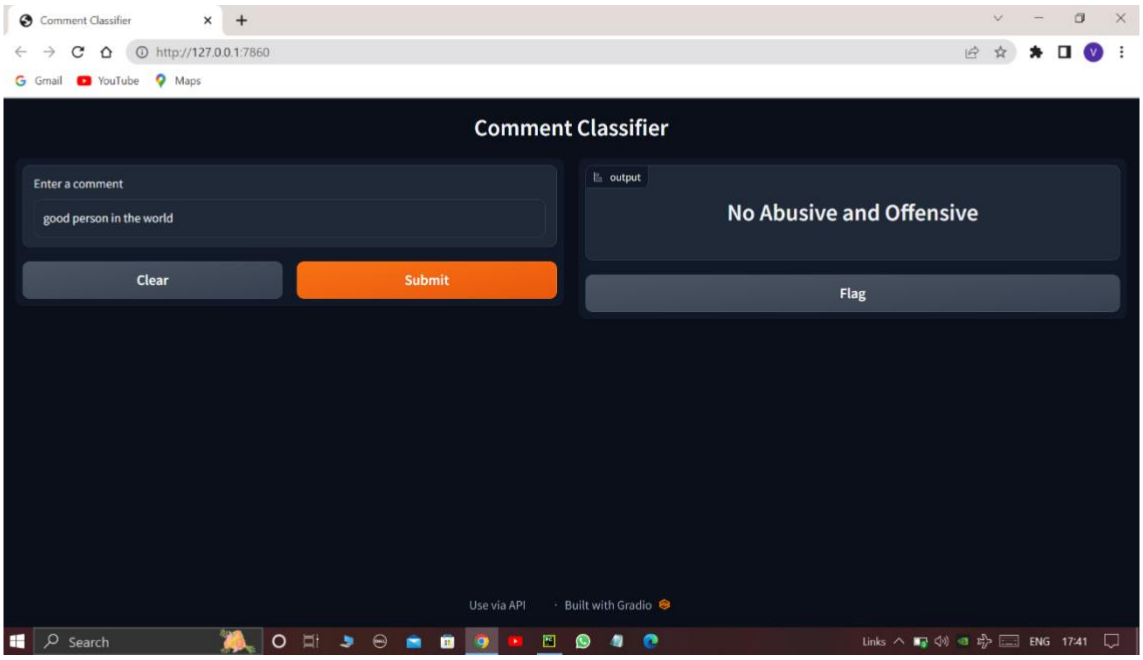


Fig 4.2 Offensive comment

- **Output with No Offensive and Abusive detection:**



Comment Classifier

Enter a comment

good person in the world

No Abusive and Offensive

Fig 4.3 No Offensive and Abusive

Chapter 5

TESTING

5. TESTING :-

To test the Comment Classifier code, you can follow these steps:

1. Ensure that you have the required libraries installed, such as Gradio, pandas, numpy, scikit-learn, and nltk. You can install them using the pip package manager:

   - pip install gradio pandas numpy scikit-learn nltk

2. Prepare your dataset by creating a CSV file with comment data. The file should have two columns: "comments" containing the comment text and "class" representing the class label for each comment.

3. Update the code with the appropriate file path for your dataset. In the provided code, the dataset file is assumed to be named "commentdatset.csv". Modify the following line to match your file name:

   - ‘python
   - data = pd.read_csv("commentdatset.csv")

4. Run the code using a Python IDE or from the command line:

   -
     ```python
   - python comment_classifier.py

5. Once the code is running, it will launch the Gradio interface, which provides a textbox for entering a comment. Type a comment into the textbox and press Enter or click the "Classify" button.

6. The model will process the input comment and provide the predicted class label as the output.

7. Repeat step 5 with different comments to test the classifier's performance on various inputs.

By following these steps, you can test the comment classifier and observe its predictions for different comments.

## 5.1 How testing works:

In software development, testing is a crucial process that involves evaluating the functionality, performance, and quality of a software application or system. Testing is typically performed in a systematic and structured manner to identify defects, ensure that the software meets the specified requirements, and verify its behavior under different conditions.
     ```

The testing process typically involves the following steps:

- Test Planning: In this phase, the testing objectives, scope, and strategy are defined.
- Test Design: Test design involves creating detailed test cases and test scenarios that cover different aspects of the software.
- Test Execution: Test cases are executed using the designed test data and scripts.
- Defect Reporting: Defects may be classified based on their severity, priority, and impact on the software system.
- Defect Resolution: Here we fix the defects and release updated versions of the software.
- Test Closure: Test closure activities may include documenting lessons learned, conducting post-mortem meetings, and archiving test artifacts.

The testing process, different techniques and approaches may be used, such as functional testing, regression testing, performance testing, security testing, and user acceptance testing.

The specific testing approach and techniques depend on the nature of the software, project requirements, and available resources.

Testing plays a crucial role in ensuring the quality and reliability of software applications. It helps identify defects, validate the functionality, and provide confidence in the software's performance and behavior under different conditions.

5.3 Types Of Testing:

Different types of testing that can be performed on the comment classifier code include:

- **Unit Testing:** Unit testing involves testing individual components or functions of the code in isolation to ensure they produce the expected output for different input scenarios. We can write unit tests for functions like `clean()`, `classify_comment()`, and other helper functions to verify their correctness.
- **Integration Testing:** In the case of the comment classifier, you can perform integration testing to ensure that the data preprocessing, feature extraction, model training, and prediction stages work together correctly.

- **Functional Testing:** This includes testing the Gradio interface to ensure it correctly accepts user input, passes it to the classifier, and displays the predicted class label accurately.

- **Performance Testing:** Performance testing is conducted to assess the speed and efficiency of the comment classifier..

- **Boundary Testing:** Boundary testing involves testing the comment classifier with extreme or boundary inputs.

- **Stress Testing:** Stress testing involves subjecting the comment classifier to high loads or extreme conditions to evaluate its robustness and stability. This can include testing the classifier with a large volume of comments or simulating concurrent user interactions with the Gradio interface.

- **Cross-Validation Testing:** Cross-validation testing is used to evaluate the performance of the comment classifier on different subsets of the dataset.

By conducting these different types of testing, you can ensure the correctness, reliability, performance, and robustness of the comment classifier code.

Chapter 6

Advantages and Disadvantages

Advantages:-

Comment classifiers offer several advantages in various applications:

- **Efficiency:** Comment classifiers automate the process of analyzing and categorizing comments, allowing for efficient handling of large volumes of user-generated content.

- **Scalability & Consistency:** That model process a vast number of comments quickly, comment classifiers are highly scalable, more useable making them suitable for platforms with high user engagement or large user bases.

  Comment classifiers provide consistent categorization and analysis of comments.

- **Real-time Analysis:** Comment classifiers can analyze comments in real-time, enabling immediate responses or actions to address user concerns, identify emerging trends, or detect potential issues promptly.

- **User Experience Improvement:** By automatically filtering out inappropriate or offensive comments, comment classifiers contribute to creating a safer and more positive user experience on online platforms, fostering a healthier community environment.

- **Content Moderation:** Comment classifiers assist in identifying and flagging inappropriate, spam, or harmful content, aiding in content moderation efforts and reducing the burden on human moderators.

- **Personalization:** Comment classifiers can be used to tailor content recommendations or responses based on user comments, providing a more personalized and relevant user experience.

- **Time Savings:** By automating comment analysis and categorization, comment classifiers free up time for human moderators to focus on more complex or subjective tasks that require human judgment and intervention.

Disadvantages:

While comment classifiers offer several advantages, they also have some limitations and potential disadvantages:

- **Privacy Concerns:** Comment classifiers analyze and process user-generated content, which raises privacy concerns. Users may worry about the collection, storage, and potential misuse of their personal data or comments.

- **Subjectivity and Contextual Understanding:** Comment classifiers may struggle with understanding the context, sarcasm, or nuanced language used in comments.

- **Bias and Fairness:** Comment classifiers can be susceptible to bias, reflecting the biases present in the training data or the underlying algorithms.

- **Evading Techniques:** As comment classifiers become more advanced, people may attempt to manipulate or deceive them by using obfuscated or coded language.

- **Lack of Human Judgment:** Comment classifiers lack human judgment and may struggle with accurately capturing the subtleties and nuances of human communication.

To mitigate these disadvantages, it is important to regularly evaluate and address biases, incorporate user feedback, combine automated systems with human moderation, and implement transparency measures to increase accountability and trust in the comment classification process.

Chapter 7

CONCULSION

7. CONCULSION:

The **Comment Classifier project** presents a comprehensive solution for categorizing text comments using machine learning techniques. By automating the classification process, it streamlines content management tasks and enables more efficient analysis of user-generated comments. It utilizes a decision tree algorithm and natural language processing techniques to effectively classify comments into different categories, such as offensive language, abusive comments, or non-abusive and non-offensive comments.

The code demonstrates the use of Python libraries like pandas, numpy, and sklearn for data manipulation, numerical operations, and machine learning functionalities. It also employs the Gradio library to create an intuitive interface for inputting comments and obtaining classification results.

The code follows a well-defined pipeline, including data preprocessing, feature extraction with CountVectorizer, model training using a decision tree classifier, and a function for classifying new comments. It can be expanded to handle multilingual comments, incorporate advanced NLP techniques, or integrate with other applications.

The comment classifier showcases the potential and effectiveness of machine learning and NLP techniques in analyzing and categorizing comments. It offers a scalable solution for managing and moderating large volumes of user comments in diverse domains.

Chapter 8

FUTURE SCOPE

8. Future Scope:

The future scope of comment classifiers is promising, with several potential advancements and applications on the horizon. There are various development can be made in coming future as there contextual understanding, multilingual and cross cultural analysis, enhanced filtering and moderation, integration with chatbots, privacy and data protection.

Future development that implement in this model:

- Contextual Understanding: Improving the ability to interpret contextual nuances.
- Multilingual and Cross-Cultural Analysis: Expanding capabilities to analyze comments in multiple languages.
- Emotion and Intent Recognition: Advancing to recognize specific emotions and intents expressed in comments for more nuanced analysis.
- Bias Mitigation and Fairness: Addressing biases through fairness measures and regular audits to ensure equitable treatment.
- Enhanced Filtering and Moderation: Improving the detection and filtering of harmful or inappropriate content.
- User Engagement and Recommendations: Leveraging comment classifiers to provide personalized content recommendations.
- Explain ability and Transparency: Enhancing explain ability to foster trust and accountability.
- Integration with Chatbots and Conversational AI: Integrating comment classifiers with chatbot systems for context-aware responses.
- Continuous Learning and Adaptation: Updating models to adapt to evolving comment patterns and user behavior.

Some potential achievements above discuss that makes our model more efficient, accurate, can be use at various places(chatbot, AI software, on social media platforms,..etc).

References:-

- https://www.kaggle.com/
- https://www.google.com/
- http://127.0.0.1:7860/
- https://www.python.org/
- https://www.jetbrains.com/pycharm/
- https://scikit-learn.org/stable/
- https://www.nltk.org/
- https://www.youtube.com/
- https://gradio.app/docs/
- https://pandas.pydata.org/
- https://matplotlib.org/