VIPS
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION
VIVEKANANDA SCHOOL OF ENGINEERING AND TECHNOLOGY

Minor Project Report
“Unveiling Audience Insights”

Under the Guidance
of
Ms. Vidushi

Submitted By:

Manas Saluja , 09117711921 , AI-DS , B

Shruti Thareja ,10417711921 , AI-DS , B

DECLARATION

This is to certify that Minor Project Report titled **Unveiling Audience Insights**, for the Minor Project with AIDS451/AIML451/IIoT451 is submitted by **Manas Saluja , (09117711921) and Shruti Thareja (10417711921)** in partial fulfillment of the requirement for the award of degree B.Tech. in Artificial Intelligence and Data Science/, VIPS-TC, GGSIP University, Dwarka, Delhi. It comprises of our original work. The due affirmation has been made within the report for utilizing the referenced work.

Date: **Signature < Manas Saluja -09117711921>**

Signature < Shruti Thareja (10417711921) >

Certificate by Supervisor

This is to certify that Minor Project Report titled **Unveiling Audience Insights**, for the Minor Project with AIDS451/AIML451/IIoT451 is submitted by **Manas Saluja (09117711921) and Shruti Thareja (10417711921)** in partial fulfillment of the requirement for the award of degree B.Tech in Artificial Intelligence and Machine Learning/ Artificial Intelligence and Data Science/ Industrial Internet of Things, VIPS-TC, GGSIP University, Dwarka, Delhi. It is a record of the candidates own work carried out by them under my supervision. The matter embodied in this Report is original and has not been submitted for the award of any other degree.

Date: **(Signature)**

SUPERVISOR : Ms. Vidushi

<Signature of HOD> **<Signature of Programme Coordinator>**

Acknowledgment

We would like to express my heartfelt gratitude to all those who have contributed to the successful completion of our minor project report. First and foremost, we would like to extend our deepest appreciation to our supervisor, [Supervisor's Name], for their constant guidance, valuable insights, and unwavering support throughout this journey. We are also indebted to the staff and faculty members of VIPS-TC, GGSIPU, Dwarka, Delhi, whose expertise and cooperation were instrumental in enhancing our learning experience. Additionally, we are grateful to our fellow students who provided us with invaluable feedback and encouragement.

(Signature of the students)

Table of Contents

List of Figures

Abstract

With the exponential increase in textual data, particularly in the form of user-generated content, there has been a growing interest in applying machine learning (ML) and natural language processing (NLP) techniques to analyze this information. Sentiment analysis of YouTube comments, a rich source of public opinion, has become a popular research area. Despite the abundance of comments and feedback on various videos, extracting meaningful trends from these comments remains challenging due to issues such as low consistency and data quality. In this study, we focus on performing sentiment analysis on YouTube comments related to trending topics, leveraging various machine learning algorithms.

We explore how sentiment analysis, by identifying patterns, seasonality, and forecasting, can offer insights into the impact of real-world events on public sentiment. Our findings show that user sentiment trends correlate strongly with the occurrence of significant real-world events associated with specific keywords in the comments. The primary objective of this research is to assist researchers in identifying quality research papers on sentiment analysis.

For this study, we perform sentiment analysis on YouTube comments using a pre-existing annotated corpus of 1500 citation sentences. The dataset is cleaned using various data normalization techniques to remove noise from the comments. We apply six different machine learning algorithms—Naïve Bayes (NB), Logistic Regression (LR), Decision Tree (DT), Linear SVC, and Random Forest (RF)—to classify the data. The effectiveness of these algorithms is evaluated based on performance metrics like F-score and Accuracy score, providing a comprehensive view of how sentiment analysis can be conducted on YouTube comments and its potential applications in understanding public sentiment.

To further enhance the reliability of sentiment analysis, we also explore the application of advanced preprocessing techniques such as tokenization, stopword removal, and stemming. By refining the quality of the input data, we aim to improve the performance of the classification models. The integration of these advanced techniques provides a more nuanced understanding of user sentiments and offers a potential pathway for more accurate trend prediction. Ultimately, this research seeks to demonstrate how automated sentiment analysis can be a powerful tool in social media monitoring, allowing businesses, governments, and researchers to track public opinion and make data-driven decisions in real time.

Chapter 1: Introduction

In this work, we aim to collect data from YouTube comments to measure the sentiments of users regarding the aspects of a video they describe in text form. Sentiment analysis allows us to quickly gain insights from large volumes of text data, making it a powerful tool for understanding user opinions. Sentiment analysis, also known as opinion mining, involves identifying and classifying the sentiments expressed in text, whether they are positive, negative, or neutral. This analysis can reveal emotions, attitudes, impressions, and opinions about a particular subject.

With over 1 billion unique users watching more than 6 billion hours of video each month, YouTube is a dominant platform in the digital landscape, contributing to 20% of web traffic and 10% of total internet traffic. YouTube offers a variety of mechanisms to gauge user opinions and feedback through features such as voting, ratings, favorites, shares, and comments. Users can not only interact with videos but also engage with one another through comments, subscriptions, and social networking features. This unique user-to-user interaction on YouTube differentiates it from other content platforms and provides a rich source of data for sentiment analysis.

Text analytics plays a crucial role in analyzing the unstructured data found in natural language text. Using various machine learning tools and techniques, text analysis offers a cost-effective method for assessing public opinion. In this study, we focus on performing sentiment analysis on YouTube comments using a specialized annotated corpus made up of 1,500 citation sentences. The corpus is manually annotated to assign polarity (positive, negative, neutral) to each sentence, serving as a foundation for training our machine learning models.

We have implemented six different machine learning algorithms—Naïve Bayes, Logistic Regression, Linear SVC, Decision Tree, and Random Forest—to classify the sentiment of the comments. The accuracy of these algorithms is evaluated using various metrics such as F-score and accuracy score to assess the correctness of the classification system. To further enhance the performance of our system, we employ several feature selection techniques, including lemmatization, n-grams, tokenization, stopword removal, and punctuation removal. These techniques help improve the quality of the input data, leading to more accurate sentiment predictions.

Chapter 2 : Related Work

Several researchers have performed sentiment analysis of social networks like Twitter and YouTube. These works affect comments, tweets, and other metadata collected from the social network profiles of users or of public events that are collected and analyzed to get significant and interesting insights about the usage of these social network websites by the overall mass of individuals. The work most closely associated with ours is by Siersdorfer et al. They analyzed quite 6 million comments collected from 67,000 YouTube videos to identify the connection between comments, views, comment ratings and topic categories. The authors show promising leads to predicting the comment ratings of latest unrated comments by building prediction models using the already rated comments. Pang, Lee and Vaithyanathan perform sentiment analysis on 2053 movie reviews collected from the web Movie Database (IMDb). They examined the hypothesis that sentiment analysis are often treated as a special case of topic-based text classification. Their work depicted that standard machine learning techniques such as Naive Bayes or Support Vector Machines (SVMs) outperform manual classification techniques that involve human intervention.

However, the accuracy of sentiment classification falls in need of the accuracy of ordinary topic-based text categorization that uses such machine learning techniques. They reported that the simultaneous presence of positive and negative expressions (thwarted expectations) within the reviews make it difficult for the machine learning techniques to accurately predict the emotions.

Another work on the YouTube comments was done by Smita Shree and Josh Brolin where the authors proposed an unsupervised lexicon-based approach to detect sentiment polarity of user comments in YouTube. They adopted a knowledge driven approach and ready a social media list of terms and phrases expressing the user sentiment and opinion. But their results also showed that recall of negative sentiment is poorer compared to the positives, which can flow from to the wide linguistic variation utilized in expressing frustration and dissatisfaction.

Web scraping is a process of information extraction from the world wide web (www), accomplished by writing automated script routines that request data by querying the desired web server and retrieving the data by using different parsing techniques [4]. Scraping helps in transforming unstructured HTML data into various structured data formats like CSV, spreadsheets. As it is known, the nature of web data is changing frequently, using an easy-to-use language like python which accepts dynamic inputs can be highly productive, as code changes are easily done to keep up with the speed of web updates. Using the wide collection of python libraries, such as requests, pandas, csv, webdriver can ease the process of fetching URLs and pulling out information from web pages[1]

Chapter 3 : Problem Statement and Objectives

Problem Statement

YouTube has become a major hub for user-generated content, ranging from entertainment, education, to product reviews. Every day, millions of videos are uploaded, and an even larger number of comments are generated by viewers. These comments represent the opinions, feelings, and reactions of audiences towards the content they consume, providing valuable insights into public sentiment. Analyzing these comments can help content creators, marketers, and businesses understand their audience better, improving both their content strategy and business decisions.

Despite the potential insights embedded within these comments, manually analyzing thousands of comments on popular videos is impractical and time-consuming. This project addresses the need for an automated system capable of collecting, processing, and analyzing these comments. By utilizing Natural Language Processing (NLP) techniques, this system can perform sentiment analysis on a large scale, providing users with actionable insights into how their content is being received.

Objectives

The objective of this project is to develop an system that can collect, process, and analyze YouTube comments using Natural Language Processing (NLP) techniques to perform sentiment analysis on a large scale. By doing so, the system aims to provide content creators, marketers, and businesses with actionable insights into how their content is being received by the audience. The project focuses on efficiently processing and analyzing large volumes of comments, extracting valuable insights related to public sentiment, and improving content strategy and business decision-making. This will help users gain a better understanding of their audience's opinions, emotions, and reactions towards their content, thereby enhancing their overall engagement and decision-making process.

Chapter 4 : Project Analysis and Design

4.1: Hardware and Software Requirements

Hardware Requirements:

- Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- RAM: 8 GB (7.65 GB usable)
- System Type: 64-bit, x64-based processor
- Storage: 512GB SSD
- Graphics: Intel Iris Xe Graphics
- Internet: Stable connection for web scraping and API usage

Software Requirements:

- Operating System: Windows 10/11, macOS, or any Linux distribution
- Programming Language: Python 3.x
- Data Visualization: Chart.js/D3.js: For visualizing sentiment results
- Dataset Source: YouTube: For real-time comment data

4.2: Flow Chart

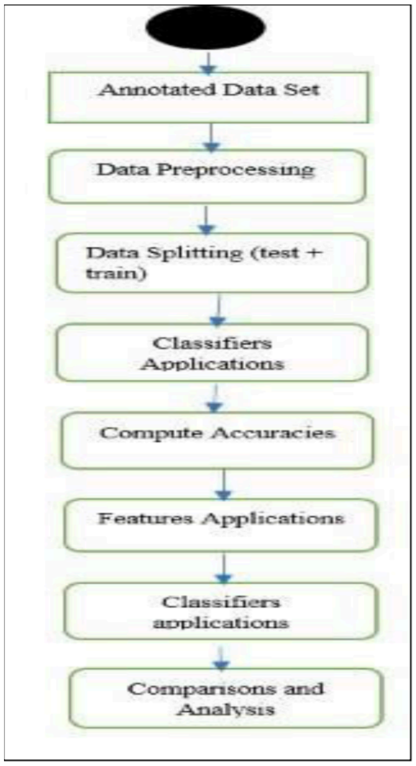


Fig 1. Flow chart of system

Chapter 5: Proposed Work and Methodology Adopted

Methodology

The purpose of the methodology is defined in this section. Our methodology is depicted in Fig. 1. First of all, we used the annotated dataset. We used the python-based machine learning library named Scikit-Learn for implementing the system. Scikit-Learn is a well-known machine learning library tightly integrated with the Python language and provides an easy-to-interact interface.

First of all, our system reads the data stored in the file having (Tab Separated Values) format. After reading, a pre-processing phase is applied to clean and prepare the data for the use of machine learning algorithms. Directly text data cannot be given to machine learning algorithms, it should be converted into a suitable type. Using Scikit-Learn module named "count-vectorizer," the text data is firstly converted into numeric format and prepared into the matrix of tokens count.

Now the data is ready for machine learning algorithms. Then, 70% of the data is split randomly to train the classifier and 30% for testing the classifier's accuracy.

We have used multiple machine learning algorithms by which we can get best feature by seeing it s , 'Accuracy' 'Precision Score' 'Recall Score' and 'F1-Score' We then select the best feature which is giving the best result and which classifier is better in a specific scenario.

The evaluation of any research product decides the status and quality of that specific research work. This section briefly describes the metrics used to evaluate the sentiment analysis system we developed. The performance of the sentiment analysis system is evaluated by computing the accuracy of the classification results given by the system. Accuracy of the system is to be mentioned in the form of some units that include F-score and Accuracy score.

In our evaluation phase, we have calculated both Macro-F Score as well as Micro-F Score, where FP is considered an error of type-1 (false positive) and FN is considered an error of type-2 (false negative). F-score is commonly used as a harmonic mean between precision and recall.

DATA PRE-PROCESSING

As corpus used for sentimental analysis classification is prepared or constructed. This dataset is comprised of a total of 18400 citation sentences annotated as positive, negative, and neutral after applying rules. From the total citation sentences, 70% of sentences were chosen randomly for training the classifier and the rest of the 30% data was used for classifier testing. The dataset was cleaned to get the highest accuracy of the system.

A. Feature Selection

For the sake of developing a system for sentiment analysis, different features are provided by ML. We have used various features, e.g., lemmatization, stop words, and term-document frequency to evaluate the classifier’s accuracy. Later, the evaluation results will be displayed.

B. Lemmatisation

Lemmatisation is a process of normalising the inflected forms of words. Homographic words cause ambiguity that disturbs searching accuracy, and this ambiguity may also occur due to inflectional word forms. For instance, words like "Talking," "Talks," and "Talked" are the inflected forms of the word "Talk." The process of lemmatization and stemming is similar with minor changes, while the benefits of both approaches are the same. We have applied only lemmatization and avoided stemming due to the problems of the stemming process. The stemming process is worthwhile for short retrieval lists, while our system has to deal with a large dataset and processing lists, so we did not apply stemming. Stemming performs normalization of inflected words by keeping different variations of words along with their derivation process.

C. Stop Words and Punctuation

English text contains a lot of meaningless and non-informative words called stop words. These are not required in classification because their presence just increases the size of data. So, we applied stop words removal technique in order to cleanse the data for better and efficient classification. Some research works support the stop words removal from the dataset to reduce the dimensions of data.

ALGORITHMS USED

This work attempted to utilize six machine learning techniques for the task of sentiment analysis. The modeling of all techniques is briefly discussed below.

■Classification Classifiers
After pre-processing and feature selection, the very next step is to apply classification algorithms. Many text classifiers have been purposed in literature. We have used 5 algorithms of machine learning including:Random Forest Classifier , Logistic Regression ,Multinomial Naive Bayes, Decision Tree Classifier and Linear Support-Vector Classifier

A) Random Forest Classifier
The Random Forest Classifier is an ensemble learning method that combines multiple decision trees to create a more robust and accurate classification model. It works by generating a large number of decision trees, each trained on different subsets of the data, and then making predictions based on the majority vote of all the trees. The Random Forest algorithm is known for its ability to handle overfitting issues, manage high-dimensional data, and provide efficient and discriminative classification. It is especially useful when dealing with large datasets and is often considered an interesting classifier due to its effectiveness in both classification and regression tasks. In comparison to other classifiers, Random Forest tends to deliver high accuracy and resilience against noise in the data.

B) Logistic Regression
Logistic Regression is a statistical method used for binary classification tasks. It models the probability of a categorical dependent variable based on one or more independent variables. Logistic Regression uses the logistic function to transform the linear combination of input features into a value between 0 and 1, representing the probability of a certain class. It is simple, fast, and interpretable, making it a popular choice for binary classification problems. Despite its simplicity, it performs well in cases where the relationship between features and the target variable is approximately linear, but it can struggle when dealing with non-linear patterns.

C) Multinomial Naive Bayes
The Multinomial Naive Bayes classifier is based on Bayes' theorem and is particularly effective for classification tasks involving discrete features, especially text classification problems. It assumes that the features are conditionally independent given the class label, which simplifies the computation. The Multinomial Naive Bayes classifier is well-suited for multi-class classification problems, and it is commonly used in spam filtering and sentiment analysis tasks. Its main strength lies in its ability to handle large vocabulary sizes and categorical data, though its assumption of independence between features can limit performance if this assumption does not hold true.

D) Decision Tree Classifier
The Decision Tree Classifier is a supervised machine learning algorithm that splits the dataset into subsets based on feature values, creating a tree-like structure. Each node in the tree represents a decision based on a feature, and each leaf node represents the class label. Decision trees are easy to understand, interpret, and visualize. They can handle both numerical and categorical data. However, they are prone to overfitting, especially when the tree is very deep, and may not generalize well to new, unseen data. Techniques like pruning or ensemble methods (e.g., Random Forest) are often applied to improve performance.

E) Linear Support-Vector Classifier
The Linear Support-Vector Classifier (SVC) is a supervised learning algorithm that works by finding the hyperplane that best separates the data into two classes. The algorithm maximizes the margin between the two classes, ensuring that the decision boundary is as far as possible from any data point. Linear SVC is particularly effective when the data is linearly separable. However, it can struggle with non-linearly separable data unless the kernel trick is applied. It is computationally efficient and performs well in high-dimensional spaces, but its performance can degrade when the classes are not linearly separable or when the data contains noise.

WEB SCRAPING FOR DATASET

Web scraping is the process of extracting information from the World Wide Web using automated scripts that request and retrieve data from web servers. It involves parsing unstructured HTML data into structured formats like CSV or spreadsheets. Since web data often changes rapidly, Python is a popular choice for web scraping because it allows for quick modifications in response to web updates. Libraries such as requests, pandas, csv, and BeautifulSoup make it easier to send HTTP requests, parse content, and extract data from web pages. For more dynamic websites, tools like Selenium WebDriver can handle JavaScript-rendered content.

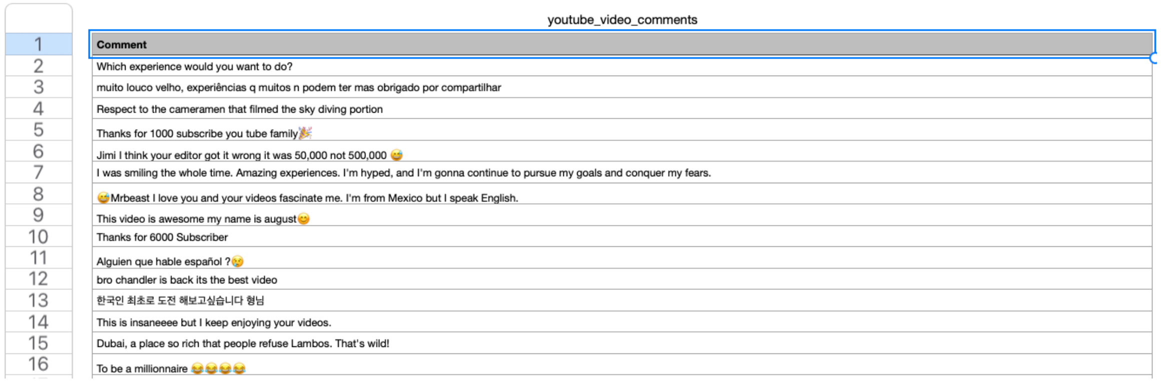
In addition to traditional scraping, YouTube provides an API (YouTube Data API) that allows developers to retrieve structured data from the platform, such as video details, comments, and channel information. Using the YouTube API is often preferred over direct HTML scraping, as it ensures compliance with YouTube's terms of service and offers easier access to data. To use the YouTube API, developers need to generate an API key from the Google Cloud Console, which serves as authentication for secure access. With this API, developers can easily gather video statistics, retrieve comments, and analyze trends, making it an efficient tool for working with YouTube content.

Chapter 6: Results and Discussion

6.1 Data collection

- web scraping using YouTube data API's [2]
Where we fetch comments into csv file using YouTube link provided as input for which code is given

The output dataset is further save in csv file name as youtube_video_comments.csv which keeps updated once the input YouTube link is changed



youtube_video_comments

1	**Comment**
2	Which experience would you want to do?
3	muito louco velho, experiências q muitos n podem ter mas obrigado por compartilhar
4	Respect to the cameramen that filmed the sky diving portion
5	Thanks for 1000 subscribe you tube family🎉
6	Jimi I think your editor got it wrong it was 50,000 not 500,000 😅
7	I was smiling the whole time. Amazing experiences. I'm hyped, and I'm gonna continue to pursue my goals and conquer my fears.
8	😅Mrbeast I love you and your videos fascinate me. I'm from Mexico but I speak English.
9	This video is awesome my name is august😊
10	Thanks for 6000 Subscriber
11	Alguien que hable español ?😢
12	bro chandler is back its the best video
13	한국인 최초로 도전 해보고싶습니다 형님
14	This is insaneeee but I keep enjoying your videos.
15	Dubai, a place so rich that people refuse Lambos. That's wild!
16	To be a millionnaire 😂😂😂😂

Fig 2 . Dataset after Web Scraping

6.2 Model selection

For making a best model selection we have taken a dataset which already labelled as negative , neutral and positive which look like below

	Unnamed: 0	Video ID	Comment	Likes	Sentiment
0	0	wAZZ-UWGVHI	Let's not forget that Apple Pay in 2014 requir...	95.0	1.0
1	1	wAZZ-UWGVHI	Here in NZ 50% of retailers don't even have co...	19.0	0.0
2	2	wAZZ-UWGVHI	I will forever acknowledge this channel with t...	161.0	2.0
3	3	wAZZ-UWGVHI	Whenever I go to a place that doesn't take App...	8.0	0.0
4	4	wAZZ-UWGVHI	Apple Pay is so convenient, secure, and easy t...	34.0	2.0

Fig 3 . Pre labelled dataset used for model building

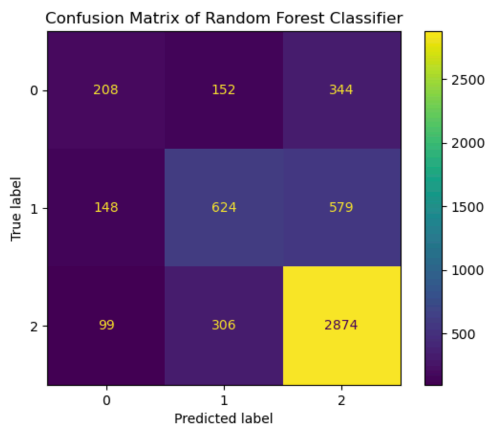
Data preprocessing for the labeled dataset involves cleaning the text by removing unnecessary characters, handling non-string values, and eliminating stopwords (common, non-informative words). The text is then tokenized into individual words or phrases to prepare it for further analysis or NLP tasks, ensuring the data is clean and ready for processing.

Further with the cleaned labelled dataset we work on 5 different machine learning Techniques Random Forest Classifier , Logistic Regression ,Multinomial Naive Bayes, Decision Tree Classifier and Linear Support-Vector Classifier

And we choose to work on 70 % train data and 30% test dataset which look like Training set size: 12445 , and Testing set size: 5334

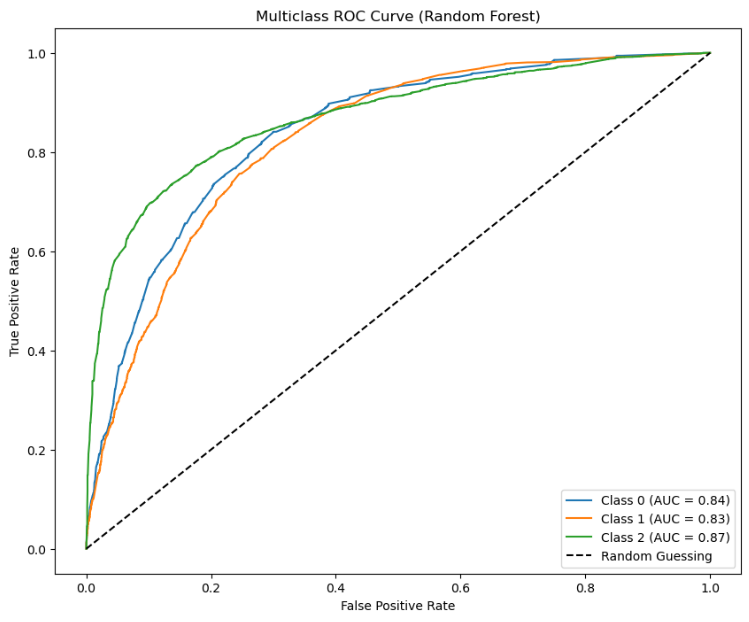
A) Random Forest Classifier

```
              precision    recall  f1-score   support

         0.0       0.46      0.30      0.36       704
         1.0       0.58      0.46      0.51      1351
         2.0       0.76      0.88      0.81      3279

    accuracy                           0.69      5334
   macro avg       0.60      0.54      0.56      5334
weighted avg       0.67      0.69      0.68      5334
```



A. Classification Report

B. Confusion matrix

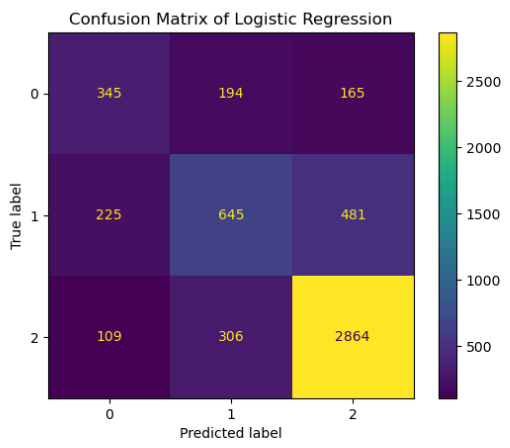


C. Multiclass Roc curve

Fig 4 . Random forest Classifier report

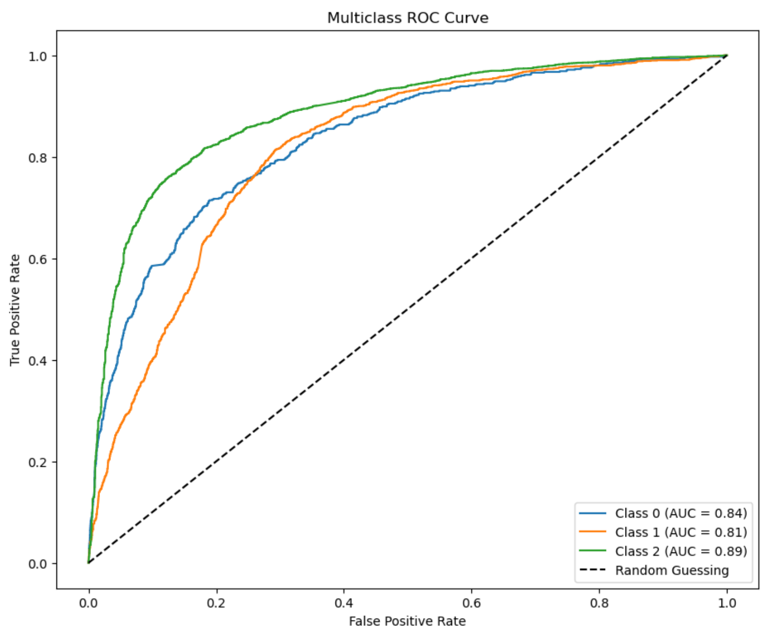
B) Logistic Regression

```
              precision    recall  f1-score   support

         0.0       0.51      0.49      0.50       704
         1.0       0.56      0.48      0.52      1351
         2.0       0.82      0.87      0.84      3279

    accuracy                           0.72      5334
   macro avg       0.63      0.61      0.62      5334
weighted avg       0.71      0.72      0.72      5334
```



A. Classification Report

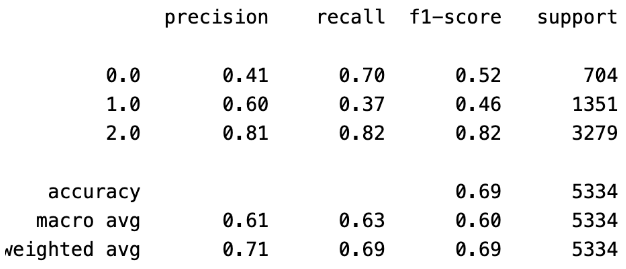
B. Confusion matrix



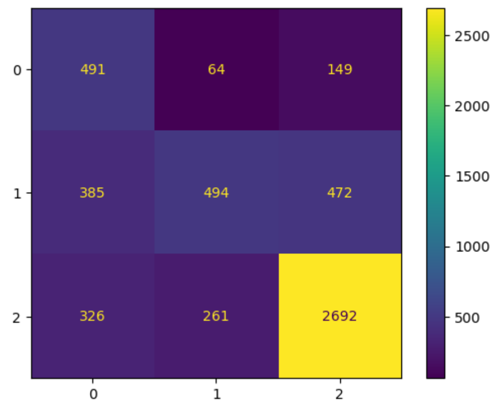
C. Multiclass Roc curve

Fig 5 . Logistic Regression report

C) Multinomial Naive Bayes,

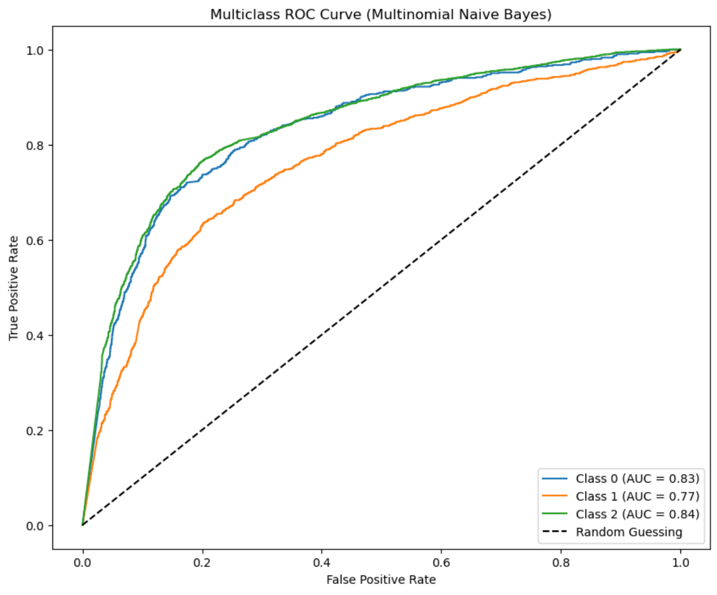


```
              precision    recall  f1-score   support

         0.0       0.41      0.70      0.52       704
         1.0       0.60      0.37      0.46      1351
         2.0       0.81      0.82      0.82      3279

    accuracy                           0.69      5334
   macro avg       0.61      0.63      0.60      5334
weighted avg       0.71      0.69      0.69      5334
```



A. Classification Report

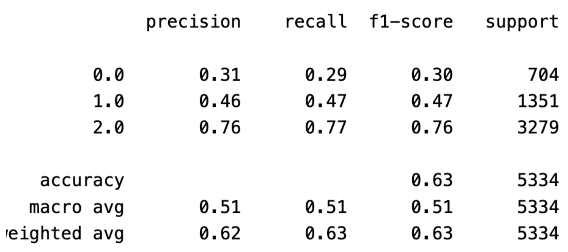
B. Confusion matrix



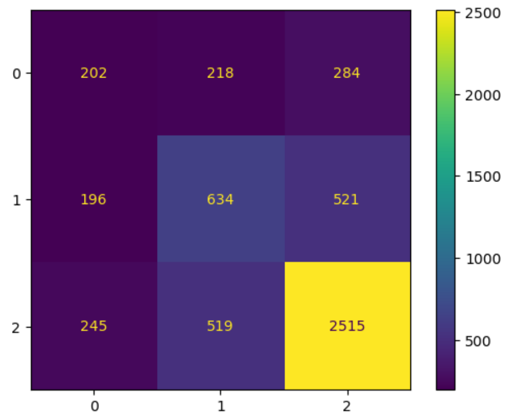
C. Multiclass Roc curve

Fig 6 . Multinomial Naive Bayes report

D) Decision Tree Classifier

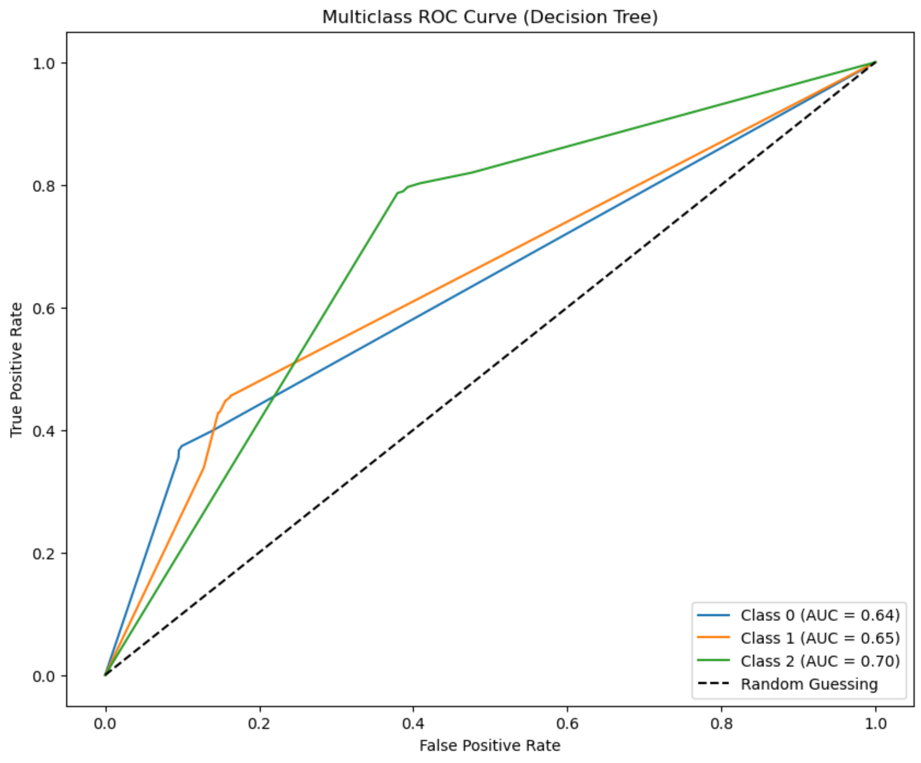


```
              precision    recall  f1-score   support

         0.0       0.31      0.29      0.30       704
         1.0       0.46      0.47      0.47      1351
         2.0       0.76      0.77      0.76      3279

    accuracy                           0.63      5334
   macro avg       0.51      0.51      0.51      5334
eighted avg       0.62      0.63      0.63      5334
```



A. Classification Report

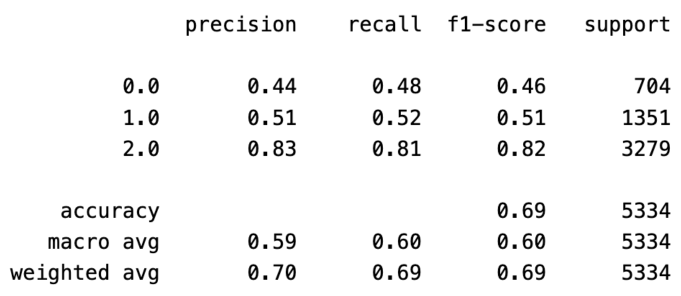
B. Confusion matrix



C. Multiclass Roc curve

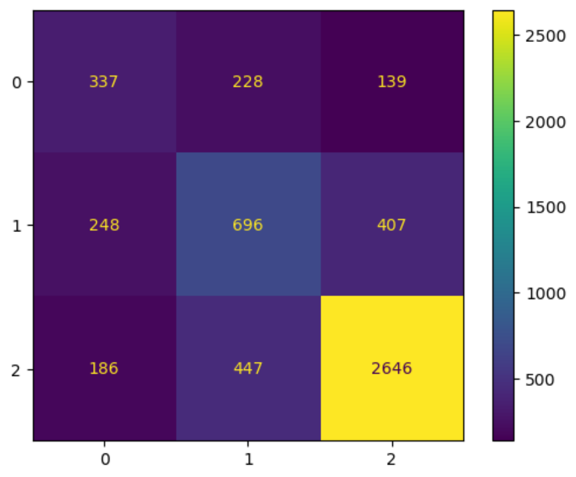
Fig 7 . Decision Tree Classifier report

E) LinearSVC

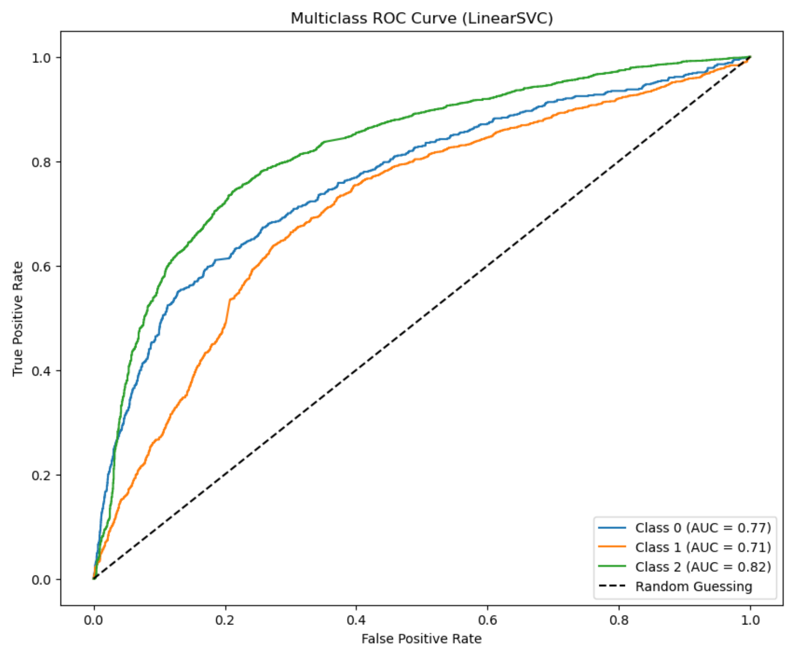


```
              precision    recall  f1-score   support

         0.0       0.44      0.48      0.46       704
         1.0       0.51      0.52      0.51      1351
         2.0       0.83      0.81      0.82      3279

    accuracy                           0.69      5334
   macro avg       0.59      0.60      0.60      5334
weighted avg       0.70      0.69      0.69      5334
```

A. Classification Report



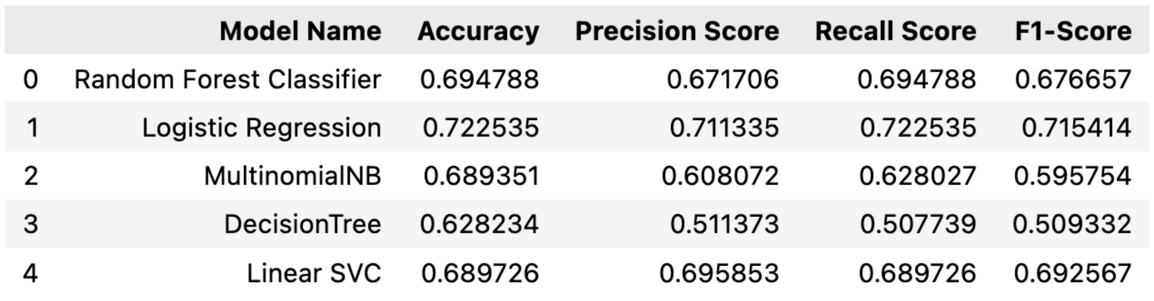
B. Confusion matrix



C. Multiclass Roc curve

Fig 8 . LinearSVC report

So after making these 5 different models we can see which one suits the best to work oby seeing this table.



	Model Name	Accuracy	Precision Score	Recall Score	F1-Score
0	Random Forest Classifier	0.694788	0.671706	0.694788	0.676657
1	Logistic Regression	0.722535	0.711335	0.722535	0.715414
2	MultinomialNB	0.689351	0.608072	0.628027	0.595754
3	DecisionTree	0.628234	0.511373	0.507739	0.509332
4	Linear SVC	0.689726	0.695853	0.689726	0.692567

Fig 9 . Result of all 5 models

By using model of logistic regression we can assume the label of any statement a user provide

Example :
Sentence = "I think I'd like more deformable terrain (breaking glass, environments affected by weather systems, world changes that aren't static and that change etc) and improved methods of moving (like verticality rather than just WASD). It feels like more fidelity comes at the expense of exploring deeper game modes. IMHO."

Predicted Class: 1.0, Probability: 0.98
Probability of 0.0: 0.02
Probability of 1.0: 0.98
Probability of 2.0: 0.00

0, 1, 2 represents a negative, neutral and positive sentiments respectively.

6.3 converting unlabelled data from scraping to insights

Now choosing the best model from above we can predict the label and probability of each sentence of a dataset we got after scraping

Example : we use a YouTube video link and got a data set as fig [2] by preprocessing on it we can get insights like some plots we can see the emotion of a user

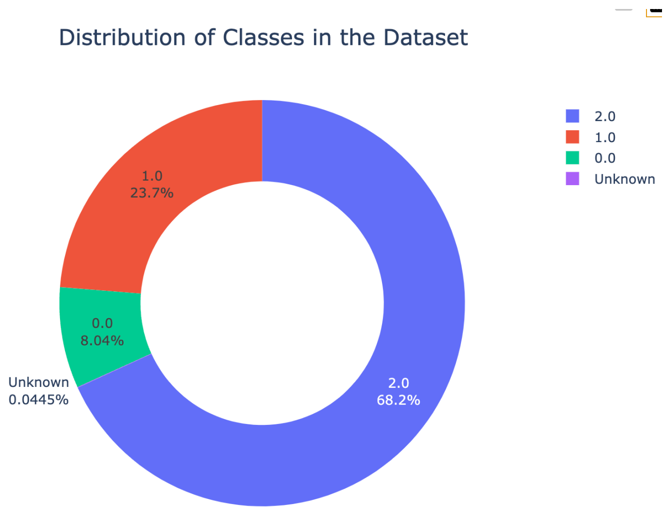


Fig 10 . Pie chart for class distribution of user labelled dataset

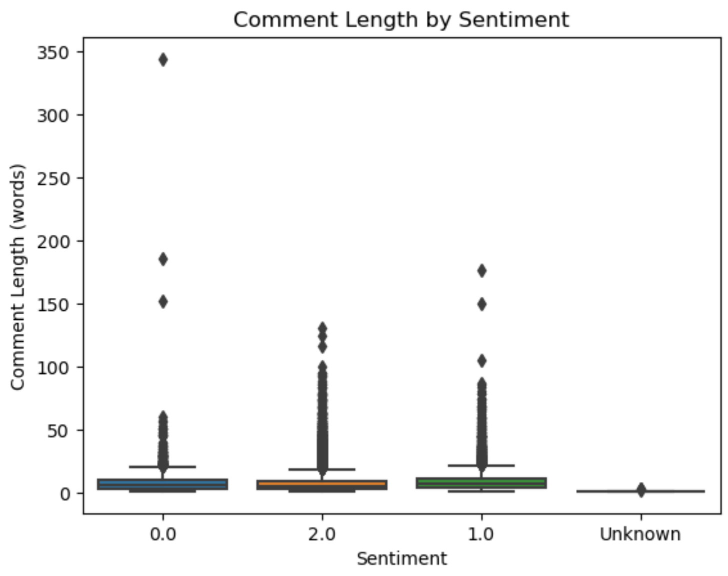


Fig 11 . Sentence length by sentiment

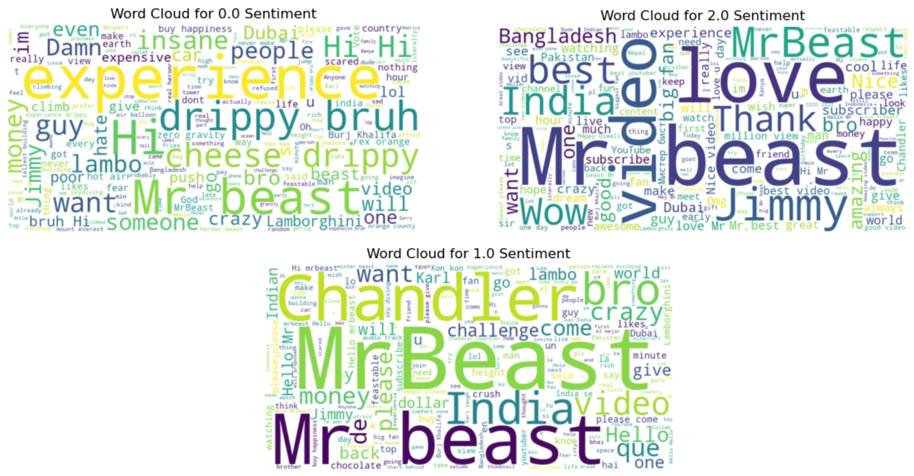


Fig 12 . Word cloud for each sentiment

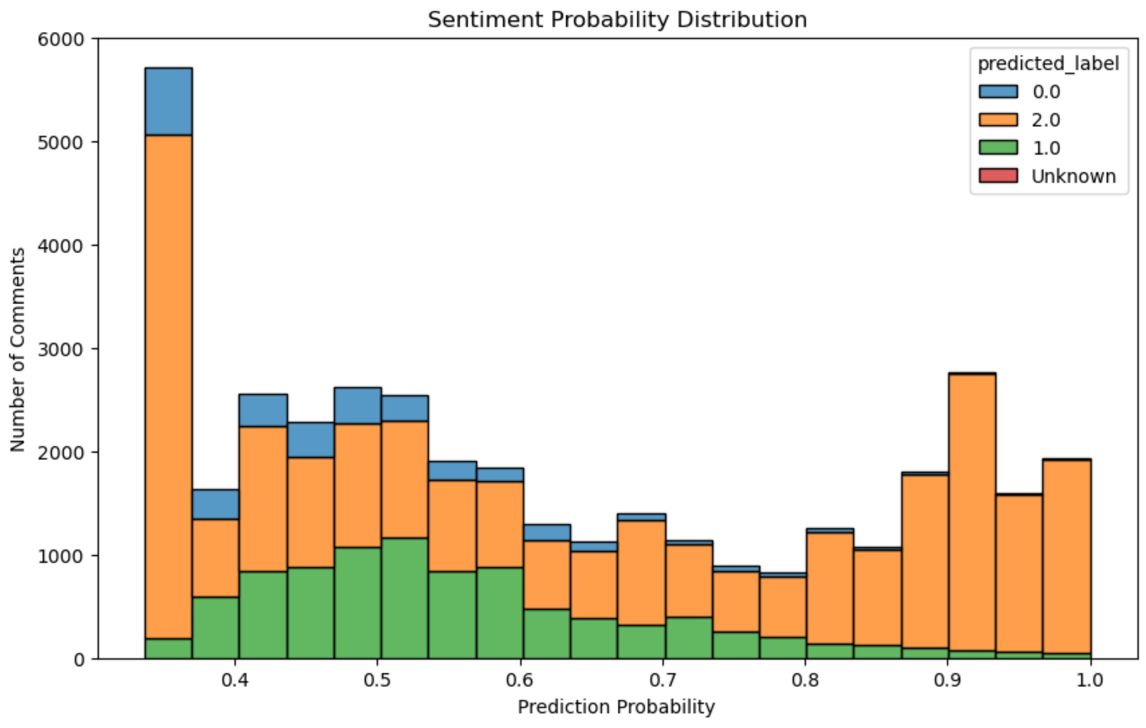


Fig 13 . Sentiment probability distribution

Chapter 7: Conclusion

The process described involves collecting, preprocessing, and modeling YouTube comment data for sentiment analysis. Initially, web scraping is performed using the YouTube Data API to fetch comments, which are then saved into a CSV file. The dataset is continuously updated whenever the input YouTube link is changed, ensuring that the latest comments are collected.

For model selection, a labeled dataset containing sentiment categories (negative, neutral, and positive) is preprocessed by cleaning the text, handling non-string values, and removing stopwords. After preprocessing, the dataset is split into a 70% training set and a 30% testing set, with 12,445 samples for training and 5,334 samples for testing. Various machine learning techniques—Random Forest Classifier, Logistic Regression, Multinomial Naive Bayes, Decision Tree Classifier, and Linear Support Vector Classifier are applied to predict sentiment labels and probabilities.

The best-performing model, in this case, Logistic Regression, is used to classify the sentiment of user comments as its accuracy is one of the best among all . By analyzing the predictions and visualizing the data (e.g., through plots), insights into the emotions of the users can be gained. This methodology allows for an efficient and dynamic way to analyze YouTube comment sentiment, providing valuable insights into audience engagement and feedback.

By leveraging sentiment analysis on YouTube comments, businesses, content creators, and marketers can better understand their audience's reactions to specific content. This approach helps identify trends, common opinions, and potential areas for improvement, enabling them to tailor their content more effectively. Furthermore, with the ability to classify comments in real-time, the model can offer ongoing insights that help track shifts in audience sentiment over time, providing valuable feedback loops for content strategy and user interaction.

Chapter 8: Future Scope of Work

The current sentiment analysis model can be expanded in several ways to enhance its effectiveness:

1. **Multilingual Support**: Expanding the model to handle multiple languages would increase its global applicability, allowing it to analyze comments from diverse audiences across different regions.

2. **Real-time Analysis**: Implementing real-time sentiment analysis would allow for immediate feedback as new comments are posted, offering timely insights for live events or content creators.

3. **Emotion and Sarcasm Detection**: Adding the ability to detect deeper emotions and sarcasm would improve the model's accuracy, addressing nuances that basic sentiment analysis might miss.

4. **Trend Analysis**: Predictive models could be developed to forecast future sentiment trends based on historical comment data, helping to anticipate audience reactions.

5. **Cross-Platform Integration**: Integrating the analysis across multiple platforms, such as Twitter and Facebook, would provide a broader view of audience sentiment, enhancing its relevance for brands.

6. **User Interaction Insights**: Analyzing how engagement metrics (likes, shares) relate to sentiment could offer more meaningful insights into how users interact with content.

These advancements would improve model accuracy, deliver richer insights, and make the system more adaptable for content creators and marketers.

REFERENCES

1. IOSR Journal of Computer Engineering, "Web scraping using YouTube data API," *International Organization of Scientific Research (IOSR)*, vol. 23, no. 3, pp. 2-10, 2021. [Online]. Available: https://www.iosrjournals.org/iosr-jce/papers/Vol23-issue3/Series-2/A2303020105.pdf

2. Google Developers, "YouTube Data API v3," *Google*, 2021. [Online]. Available: https://developers.google.com/youtube/v3/docs/

3. P. B. Bansal, "PMC10026245," *PubMed Central (PMC)*, 2020. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10026245/

4. A. V. Saurkar, K. G. Pathare, and S. A. Gode, "An overview of web scraping techniques and tools," *International Journal on Future Revolution in Computer Science and Communication Engineering*, vol. 6, no. 4, pp. 14-19, Apr. 2018.

5. S. M. K. Ukkusuri, M. N. S. Ali, and L. J. R. Ye, "Sentiment analysis on social media," *ResearchGate*, 2013. [Online]. Available: https://www.researchgate.net/publication/230758119_Sentiment_Analysis_on_Social_Media

6. M. R. Williams and D. S. Zhang, "The impact of sentiment analysis on customer satisfaction," *Springer*, vol. 4, no. 1, pp. 32-44, Jun. 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42979-021-00592-x

7. D. M. Soni, R. P. Sharma, and R. Y. Patel, "Sentiment analysis of YouTube videos using machine learning," *MDPI*, vol. 11, no. 9, pp. 2098-2104, Sep. 2021. [Online]. Available: https://www.mdpi.com/2227-7390/11/9/2098

8. A. Athar, "Sentiment analysis of scientific citations," University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-856, 2014.

9. A. Athar and S. Teufel, "Detection of implicit citations for sentiment detection," in *Proceedings of the Workshop on Detecting Structure in Scholarly Discourse*, July 2012, pp. 18-26.

10. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, ... and J. Vanderplas, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

APPENDIX

Web scraping :

```
from googleapiclient.discovery import build
import pandas as pd
import os

# Function to get YouTube comments using YouTube Data API
def get_youtube_comments(video_id, api_key):
    # Build the YouTube API client
    youtube = build('youtube', 'v3', developerKey=api_key)

    # List to hold all comments
    comments = []

    # Fetch the first batch of comments
    response = youtube.commentThreads().list(
        part="snippet",
        videoId=video_id,
        maxResults=100,
        textFormat="plainText"
    ).execute()

    # Loop through each page of results
    while response:
        for item in response['items']:
            comment = item['snippet']['topLevelComment']['snippet']
['textDisplay']
            comments.append(comment)

        # Check if there are more comments to fetch
        if 'nextPageToken' in response:
            response = youtube.commentThreads().list(
                part="snippet",
                videoId=video_id,
                pageToken=response['nextPageToken'],
                maxResults=100,
                textFormat="plainText"
            ).execute()
        else:
            break

    return comments

# Function to save comments to a CSV file
def save_comments_to_csv(comments, video_title):
    # Create a pandas DataFrame
    df = pd.DataFrame(comments, columns=["Comment"])

    # Save DataFrame to CSV
    filename = f"{video_title}_comments.csv"
    df.to_csv(filename, index=False, encoding='utf-8')
    print(f"Comments saved to {filename}")

# Extract video ID from YouTube URL
def extract_video_id(url):
    import re
    # Regular expression to extract the video ID from the URL
    video_id_match = re.search(r"v=([a-zA-Z0-9_-]{11})", url)
    if video_id_match:
        return video_id_match.group(1)
    else:
        raise ValueError("Invalid YouTube URL")

# Main function to input URL and process comments
def main():
    # Get YouTube URL and API key from user
    video_url = input("Enter YouTube URL: ")
    api_key = input("Enter YouTube Data API key: ")

    # Extract video ID from the URL
    video_id = extract_video_id(video_url)

    # Get the comments
    comments = get_youtube_comments(video_id, api_key)

    # Get video title for the CSV file name (you can improve this part)
    video_title = "youtube_video"  # Replace with logic to get actual
title if needed

    # Save comments to CSV
    save_comments_to_csv(comments, video_title)

if __name__ == "__main__":
    main()
```

Machine learning technique :

```
#Importing all Important Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import gensim


import nltk
from gensim.parsing.preprocessing import remove_stopwords, STOPWORDS
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from sklearn.naive_bayes import MultinomialNB
from  sklearn.metrics  import  confusion_matrix,  classification_report,
accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
import seaborn as sns
from plotly import graph_objs as go
from wordcloud import WordCloud
import pickle
import warnings
warnings.filterwarnings("ignore")
nltk.download('wordnet')
nltk.download('wordnet2022')
youtube_comments = pd.read_csv('comments.csv')
First_corpus = pd.DataFrame(columns=['Text','Label'])
First_corpus['Text'],First_corpus['Label'] =
youtube_comments['Comment'],youtube_comments['Sentiment']
First_corpus.head(40)
# Dropping duplicate rows
First_corpus.drop_duplicates(inplace=True)

# Checking if 'Text' column has all unique values
print(First_corpus.Text.is_unique)

# Printing the number of samples after removing duplicates
print(f'Number of Samples after removing duplicates:
{len(First_corpus)}')

# Retain all sentiment classes and proceed with multi-class
classification
# No need to drop any rows or modify the labels for binary
classification
import plotly.graph_objects as go

# Get the label distribution using value_counts
labels = df['Label'].value_counts().index
values = df['Label'].value_counts().values

# Create the pie chart (donut chart by setting the hole)
fig = go.Figure(data=[go.Pie(labels=labels,
                              values=values,
                              hole=0.6,  # This creates the "donut"
effect
                              textinfo='percent+label',  # Show
percentages and labels
```

```
                              title='Distribution of Classes in the
Dataset')])

# Update the layout for a clean look
fig.update_layout(title_font_size=20, title_x=0.5, width=750,
height=550)

# Display the pie chart
fig.show()
#Defining a function that remove unnecessary characters
def cleaning(text):
    text = re.sub(r'#\w+','', text)                 # Removing Hashtags
    text = re.sub(r'http\S+','', text)              # Removing Links &
URLs
    text = re.sub(r'@\w+','', text)                 # Removing Mentions
    text = re.sub('[()!?.\';:<>`$%',]', '', text)   # Removing
Punctuations with different forms
    text = re.sub(r'[^a-zA-Z]',' ', text)           # Removing digits
    text = re.sub(r'([a-zA-Z])\1{2,}','\1', text)   # Reduce duplicated
character (> 3) to only one
    return text

#Defining the most used list of Abbreviations to be expanded to its
original form
abbreviations = {'fyi': 'for your information',
                 'lol': 'laugh out loud',
                 'loza': 'laughs out loud',
                 'lmao': 'laughing',
                 'rofl': 'rolling on the floor laughing',
                 'vbg': 'very big grin',
                 'xoxo': 'hugs and kisses',
                 'xo': 'hugs and kisses',
                 'brb': 'be right back',
                 'tyt': 'take your time',
                 'thx': 'thanks',
                 'abt': 'about',
                 'bf': 'best friend',
                 'diy': 'do it yourself',
                 'faq': 'frequently asked questions',
                 'fb': 'facebook',
                 'idk': 'i don\'t know',
                 'asap': 'as soon as possible',
                 'syl': 'see you later',
                 'nvm': 'never mind',
                 'frfr':'for real for real',
                 'istg':'i swear to god',
    }
#Data cleaning fucntion to call the above cleaning function, converting
the characters to lower case
#and expanding the most used abbreviations

def DataCleaning(corpus):
    corpus['Text'] = corpus['Text'].apply(cleaning)     # Calling
cleaning function (1-7)
    corpus['Text'] = corpus['Text'].str.lower()         # Normalize all
characters to lowercase
    for abbreviation, full_form in abbreviations.items(): # Expanding
the Abbreviations
```

```
        corpus['Text'] = corpus['Text'].str.replace(abbreviation, 
full_form)
    return corpusimport re

# Function to clean text data and handle non-string values
def cleaning(text):
    if not isinstance(text, str):
        return ''  # Return an empty string for non-string values 
(e.g., NaN or float)

    text = re.sub(r'#\w+', '', text)                 # Removing 
Hashtags
    text = re.sub(r'http\S+', '', text)              # Removing Links & 
URLs
    text = re.sub(r'@\w+', '', text)                 # Removing 
Mentions
    text = re.sub('[()!?.\';:<>`$%’,]', '', text)    # Removing 
Punctuations with different forms
    text = re.sub(r'[^a-zA-Z]', ' ', text)           # Removing digits 
and non-alphabetical characters
    text = re.sub(r'([a-zA-Z])\1{2,}', r'\1', text)  # Reduce 
duplicated characters (> 3) to only one

    # Converting text to lower case
    text = text.lower()

    # Expanding abbreviations
    for abbreviation, expansion in abbreviations.items():
        text = re.sub(r'\b' + abbreviation + r'\b', expansion, text)  # 
Replace abbreviations

    return text
# Data cleaning function that applies the cleaning function to the 
corpus
def DataCleaning(corpus):
    # Apply the cleaning function to the 'Text' column
    corpus['Text'] = corpus['Text'].apply(cleaning)

    return corpus
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer

# Assuming df is already defined and contains 'Text' and 'Label' 
columns

# Initialize the TF-IDF Vectorizer
vectorizer = TfidfVectorizer(
    strip_accents='ascii',   # Remove non-ASCII characters
    analyzer='word',         # Tokenization at word level
    smooth_idf=True,         # Smooth IDF calculation
    norm=None,               # No normalization
    min_df=10                # Ignore terms that appear in less than 10 
documents
)

# Fit and transform the text data
X = vectorizer.fit_transform(df['Text'])
```

```
y = df['Label'].astype('Int64')  # Convert labels to Int64 type for
compatibility

# Displaying the shape of the TF-IDF matrix and a sample of the
vocabulary
print("The shape of the dataset:", X.shape)
print("\nSome of the vocabulary: ",
list(vectorizer.vocabulary_.items())[:10])
print("\n")

# Retrieve and plot the IDF scores
idf_scores = vectorizer.idf_

# Plotting the TF-IDF scores of words
plt.figure(figsize=(10, 8))
plt.scatter(range(len(idf_scores)), idf_scores, alpha=0.7, marker='o',
color='seagreen')
plt.title('TF-IDF Scores of Words', fontsize=16)
plt.xlabel('Word Index', fontsize=14)
plt.ylabel('TF-IDF Score', fontsize=14)
plt.grid(True)  # Add a grid for better readability
plt.show()
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming X and y are already defined and contain the TF-IDF features
and labels respectively

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,       # 30% of the data will be used for testing
    shuffle=True,         # Shuffle the data before splitting
    random_state=41       # For reproducibility
)

# Defining a DataFrame to hold the evaluation metrics for model
comparison
result = pd.DataFrame(columns=['Model Name', 'Accuracy', 'Precision
Score', 'Recall Score', 'F1-Score'])

# Optional: Display the sizes of the training and testing sets for
reference
print(f"Training set size: {X_train.shape[0]}")
print(f"Testing set size: {X_test.shape[0]}")
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.model_selection import cross_val_score

# List to hold the results
t_result = []

# Hyperparameter tuning for n_estimators
for i in range(1, 11):
    # Initialize Random Forest Classifier with current n_estimators
    RF = RandomForestClassifier(n_estimators=i, criterion='entropy',
n_jobs=-1)

    # Use cross-validation to evaluate the model
    scores = cross_val_score(RF, X_train, y_train, cv=5)  # 5-fold
cross-validation
    RF.fit(X_train, y_train)  # Fit the model on the entire training
set

    # Predict on the training set
    prediction = RF.predict(X_train)

    # Calculate evaluation metrics
    t_result.append((
        i,
        scores.mean(),  # Use the mean accuracy from cross-validation
        precision_score(y_train, prediction, average='weighted'),
        recall_score(y_train, prediction, average='weighted'),
        f1_score(y_train, prediction, average='weighted')
    ))

# Create a DataFrame to store the results
results_df = pd.DataFrame(t_result, columns=['n_estimators', 'Mean
Accuracy', 'Precision Score', 'Recall Score', 'F1 Score'])

# Display the results
print(results_df)
import matplotlib.pyplot as plt

# Assuming you have your t_result from the previous code block
# Create a DataFrame for plotting
result_df = pd.DataFrame(t_result, columns=['num. of estimator',
'Accuracy', 'Precision', 'Recall', 'F1-score'])

# Plotting the Training Performance
plt.figure(figsize=(10, 8))
import matplotlib.pyplot as plt

# Assuming you have your t_result from the previous code block
# Create a DataFrame for plotting
result_df = pd.DataFrame(t_result, columns=['num. of estimator',
'Accuracy', 'Precision', 'Recall', 'F1-score'])

# Plotting the Training Performance
plt.figure(figsize=(10, 8))

plt.plot(result_df['num. of estimator'], result_df['Accuracy'],
marker='o', label='Accuracy Score')
plt.plot(result_df['num. of estimator'], result_df['Precision'],
marker='o', label='Precision Score')
plt.plot(result_df['num. of estimator'], result_df['Recall'],
marker='o', label='Recall Score')
plt.plot(result_df['num. of estimator'], result_df['F1-score'],
marker='o', label='F1-Score')

plt.xlabel("Number of Estimators", fontsize=12)
plt.ylabel("Score", fontsize=12)
plt.title("Training Performance of Random Forest Classifier",
fontsize=14)
plt.xticks(result_df['num. of estimator'])  # Set x-ticks to match
number of estimators
plt.legend()
plt.grid()
plt.show()

plt.plot(result_df['num. of estimator'], result_df['Accuracy'],
marker='o', label='Accuracy Score')
plt.plot(result_df['num. of estimator'], result_df['Precision'],
marker='o', label='Precision Score')
plt.plot(result_df['num. of estimator'], result_df['Recall'],
marker='o', label='Recall Score')
plt.plot(result_df['num. of estimator'], result_df['F1-score'],
marker='o', label='F1-Score')

plt.xlabel("Number of Estimators", fontsize=12)
plt.ylabel("Score", fontsize=12)
plt.title("Training Performance of Random Forest Classifier",
fontsize=14)
plt.xticks(result_df['num. of estimator'])  # Set x-ticks to match
number of estimators
plt.legend()
plt.grid()
plt.show()
# Training the classifier on 10 estimators for best performance
RF = RandomForestClassifier(n_estimators= 10, criterion='entropy',
n_jobs=-1)
RF.fit(X_train,y_train)prediction = RF.predict(X_test)
print(classification_report(y_test,prediction))


from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
# Assuming you have your predictions ready
prediction = RF.predict(X_test) # Make sure this is done after
fitting the model
# Compute the confusion matrix
cm_matrix = confusion_matrix(y_test, prediction)
# Check unique classes to update display labels accordingly
unique_labels = set(y_test)
display_labels = ['Negative', 'Positive'] if len(unique_labels) == 2
else list(unique_labels)
# Plotting the confusion matrix
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm_matrix,
display_labels=display_labels)
cm_display.plot()
plt.title("Confusion Matrix of Random Forest Classifier")
plt.show(from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# Storing the evaluation Result of the Random Forest Classifier
model_result = ['Random Forest Classifier',
accuracy_score(y_test, prediction),
precision_score(y_test, prediction,
average='weighted'),
recall_score(y_test, prediction, average='weighted'),
f1_score(y_test, prediction, average='weighted')]
result.loc[len(result)] = model_result
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
# Assuming X_train, y_train, X_test, and y_test are already defined
# Train the Random Forest model
RF = RandomForestClassifier(random_state=42)
RF.fit(X_train, y_train)# Predict probabilities for each class
y_pred_proba = RF.predict_proba(X_test)
# Binarize the true labels for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))

# Calculate ROC and AUC for each class
n_classes = y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve (Random Forest)')
plt.legend(loc='lower right')
plt.show()from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import pandas as pd
import numpy as np
# Assuming X_train, y_train have already been defined and are
available
# Initialize an empty list to store results
t_result = []
# Set initial value and increment for C
itr = 0.1
# Perform hyperparameter tuning for C regularization
while itr <= 1.0:
LGR = LogisticRegression(penalty='l2', C=itr, fit_intercept=True,
random_state=41, solver='sag',
max_iter=1000)LGR.fit(X_train, y_train)
# Predict on the training set
prediction = LGR.predict(X_train)
# Append the results (C, accuracy, precision, recall, F1-score)
t_result.append((
itr,
accuracy_score(y_train, prediction),
precision_score(y_train, prediction, average='weighted'), #
Using weighted average for multiclass
recall_score(y_train, prediction, average='weighted'),
f1_score(y_train, prediction, average='weighted')
))
# Increment C
itr += 0.1
```

```
# Convert results to DataFrame for easy visualization
result_df = pd.DataFrame(t_result, columns=['C value', 'Accuracy',
'Precision', 'Recall', 'F1-score'])
# Print the results
print(result_df)
# Optionally, plot the results
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
plt.plot(result_df['C value'], result_df['Accuracy'], label='Accuracy
Score')
plt.plot(result_df['C value'], result_df['Precision'],
label='Precision Score')
plt.plot(result_df['C value'], result_df['Recall'], label='Recall
Score')
plt.plot(result_df['C value'], result_df['F1-score'], label='F1
Score')
plt.xlabel("C value", fontsize=12)
plt.ylabel("Score", fontsize=12)
plt.title("Performance of Logistic Regression with Varying C
Regularization", fontsize=14)
plt.legend()
plt.grid(True)
plt.show()from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
# Assuming your labels are in y_test and your model is LGR
# Predict probabilities for each class
y_pred_proba = LGR.predict_proba(X_test)
# Binarize the true labels for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
# Calculate ROC and AUC for each classn_classes =
y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve')
plt.legend(loc='lower right')
plt.show()# Training the Logistic Regression for C = 0.9 for best
performance
LGR = LogisticRegression(penalty='l2', dual=False, C=0.9,
fit_intercept=True
, random_state = 41, solver = 'sag',
max_iter=1000)
LGR.fit(X_train, y_train)
LogisticRegression(C=0.9, max_iter=1000, random_state=41,
solver='sag')
prediction = LGR.predict(X_test)
print(classification_report(y_test,prediction))
precision recall f1-score support
0.0 0.51 0.49 0.50 704
```

```
1.0 0.56 0.48 0.52 1351
2.0 0.82 0.87 0.84 3279
accuracy 0.72 5334
macro avg 0.63 0.61 0.62 5334
weighted avg 0.71 0.72 0.72 5334
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Make predictions on the test set
prediction = LGR.predict(X_test)
# Displaying the Confusion Matrix of Logistic Regression
cm_matrix = confusion_matrix(y_test, prediction)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm_matrix)
# Update display labels based on unique classes
cm_display.display_labels = np.unique(y_test) # Set display labels to
unique classes
cm_display.plot()
plt.title("Confusion Matrix of Logistic Regression")
plt.show()# Storing the evaluation results of Logistic Regression
model_result = [
'Logistic Regression',
accuracy_score(y_test, prediction),
precision_score(y_test, prediction, average='weighted'), # Use
weighted average for multiclass
recall_score(y_test, prediction, average='weighted'), # Use
weighted average for multiclass
f1_score(y_test, prediction, average='weighted') # Use
weighted average for multiclass
]
result.loc[len(result)] = model_result
NB = MultinomialNB()
NB.fit(X_train,y_train)
MultinomialNB()
# Training Classification Report
print("\t\tThe Training Classification Report")
train_prediction = NB.predict(X_train)
train_report = classification_report(y_train, train_prediction,
output_dict=True)print(classification_report(y_train,
train_prediction)) # Display the
Training Classification Report
print('\n')
# Evaluation Classification Report
print("\t\tThe Evaluation Classification Report")
test_prediction = NB.predict(X_test)
test_report = classification_report(y_test, test_prediction,
output_dict=True)
print(classification_report(y_test, test_prediction))from
sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
# Assuming X_train, y_train, X_test, and y_test are already defined
# Train the Multinomial Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
# Predict probabilities for each class
y_pred_proba = nb_model.predict_proba(X_test)# Binarize the true labels
for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
```

```
# Calculate ROC and AUC for each class
n_classes = y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve (Multinomial Naive Bayes)')
plt.legend(loc='lower right')
plt.show()# Displaying the Confusion matrix of Multinomial Naive Bayes
cm_matrix = confusion_matrix(y_test,test_prediction)
cm_display = ConfusionMatrixDisplay(confusion_matrix = cm_matrix,
display_labels = ['Negative','Positive'])
cm_display.plot()
plt.title("Confusion Matrix of Multinomial Naive Bayes Classifier")
plt.show()

import matplotlib.pyplot as plt
# Extracting metrics for both training and testing sets
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
train_scores = [train_report['accuracy'],
train_report['macro avg']['precision'],
train_report['macro avg']['recall'],
train_report['macro avg']['f1-score']]
test_scores = [test_report['accuracy'],
test_report['macro avg']['precision'],
test_report['macro avg']['recall'],test_report['macro avg']['f1-
score']]
plt.figure(figsize=(10, 6))
plt.plot(metrics, train_scores, marker='o', linestyle='-'
,
label='Training Set', color='forestgreen')
plt.plot(metrics, test_scores, marker='o', linestyle='-', label='Test
Set', color='deeppink')

# Annotating points with score values
for metric, train_score, test_score in zip(metrics, train_scores,
test_scores):
plt.text(metric, train_score, f'{train_score:.2f}', ha='right',
va='bottom', fontsize=10)
plt.text(metric, test_score, f'{test_score:.2f}', ha='right',
va='bottom', fontsize=10)
# Setting plot titles and labels
plt.title('Performance Metrics of Multinomial Naive Bayes Classifier',
fontsize=14)
plt.xlabel('Metrics', fontsize=12)
plt.ylabel('Scores', fontsize=12)
plt.legend()
plt.grid(True) # Optional: add grid for better visibility
plt.show()# Generate predictions for the test set using the Multinomial
```

```
Naive Bayes model
test_prediction = NB.predict(X_test) # Assuming NB is your trained

MultinomialNB model
# Storing the evaluation Result of Multinomial Naive Bayes
model_result = ['MultinomialNB',
accuracy_score(y_test, test_prediction),
precision_score(y_test, test_prediction,
average='macro'),
recall_score(y_test, test_prediction,
average='macro'),
f1_score(y_test, test_prediction, average='macro')]
# Append results to the DataFrame
result.loc[len(result)] = model_result
#choosing basic hyperparameters for the DTClassifier, with entropy
criterion
DT = DecisionTreeClassifier(criterion='entropy', max_features='sqrt')
DT.fit(X_train,y_train)
prediction = DT.predict(X_train) # Training Performance will
be measured
print("\t\tThe Training Classification Report")
prediction = DT.predict(X_train)
print(classification_report(y_train,prediction)) #Print the
Training Classification Report
print('\n')
print("\t\tThe Evaluation Classification Report")
prediction = DT.predict(X_test)
print(classification_report(y_test,prediction))



# Extracting metrics for both training and testing sets
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
# Using 'macro' as an example; you can switch to 'weighted' if needed
train_scores = [
accuracy_score(y_train, DT.predict(X_train)),
precision_score(y_train, DT.predict(X_train), average='macro'),
recall_score(y_train, DT.predict(X_train), average='macro'),
f1_score(y_train, DT.predict(X_train), average='macro')
]
test_scores = [
accuracy_score(y_test, DT.predict(X_test)),
precision_score(y_test, DT.predict(X_test), average='macro'),
recall_score(y_test, DT.predict(X_test), average='macro'),
f1_score(y_test, DT.predict(X_test), average='macro')
]
# Plotting the performance metrics
plt.figure(figsize=(10, 6))
plt.plot(metrics, train_scores, marker='o', linestyle='-'
,
label='Training Set', color='forestgreen')
plt.plot(metrics, test_scores, marker='o', linestyle='-', label='Test
Set', color='deeppink')

# Annotate the points on the plot
for metric, train_score, test_score in zip(metrics, train_scores,
test_scores):
plt.text(metric, train_score, f'{train_score:.2f}', ha='right',
va='bottom', fontsize=10)
```

```
plt.text(metric, test_score, f'{test_score:.2f}', ha='right',
va='bottom', fontsize=10)
plt.title('Performance Metrics of Decision Tree Classifier',
fontsize=14)
plt.xlabel('Metrics', fontsize=12)
plt.ylabel('Scores', fontsize=12)
plt.legend()
plt.show()#Displaying the Confusion matrix of Decision Tree Classifier
cm_matrix = confusion_matrix(y_test,prediction)
cm_display = ConfusionMatrixDisplay(confusion_matrix = cm_matrix,
display_labels = ['Negative','Positive'])
cm_display.plot()
plt.title("Confusion Matrix of Decision Tree Classifier")
plt.show()

# Assuming `prediction` is the predicted labels for the test set
# If you haven't already, ensure to get predictions before this step
prediction = DT.predict(X_test) # Make sure DT is your trained
Decision Tree model
# Storing the evaluation Result of Decision Tree Classifier
model_result = [
'DecisionTree',
accuracy_score(y_test, prediction),
precision_score(y_test, prediction, average='macro'), # Change
average to 'macro'
recall_score(y_test, prediction, average='macro'), # Change
average to 'macro'
f1_score(y_test, prediction, average='macro') # Change
average to 'macro'
]# Append results to the DataFrame
result.loc[len(result)] = model_result


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
# Assuming X_train, y_train, X_test, and y_test are already defined
# Train the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
DT.fit(X_train, y_train)
# Predict probabilities for each class
y_pred_proba = DT.predict_proba(X_test)
# Binarize the true labels for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
# Calculate ROC and AUC for each class
n_classes = y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve (Decision Tree)')
plt.legend(loc='lower right')
plt.show()from sklearn.svm import LinearSVC


from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# Assuming you have your training data X_train and y_train defined
LSVC = LinearSVC() # Define the LSVC model
LSVC.fit(X_train, y_train) # Fit the model to the training data
prediction = LSVC.predict(X_train) # Make predictions on the training
data
# Use 'macro' or 'weighted' for multiclass evaluation
t_result.append((itr,
accuracy_score(y_train, prediction),
precision_score(y_train, prediction,
average='macro'), # Change to 'macro' or 'weighted'
recall_score(y_train, prediction, average='macro'),
# Change to 'macro' or 'weighted'
f1_score(y_train, prediction, average='macro')))# Change to 'macro' or
'weighted'
itr += 0.1

import matplotlib.pyplot as plt
import pandas as pd
# Assuming t_result is populated with the performance metrics
result_df = pd.DataFrame(t_result, columns=['C', 'Accuracy',
'Precision', 'Recall', 'F1-score'])
# Plotting the training performance
plt.figure(figsize=(10, 8))
plt.plot(result_df['C'], result_df['Accuracy'], label='Accuracy
Score', marker='o', linestyle='-')
plt.plot(result_df['C'], result_df['Precision'], label='Precision
Score', marker='s', linestyle='--')
plt.plot(result_df['C'], result_df['Recall'], label='Recall Score',
marker=
'^', linestyle='-.')
plt.plot(result_df['C'], result_df['F1-score'], label='F1-Score',
marker='d', linestyle=':')
plt.xlabel("Values of C", fontsize=12)
plt.ylabel("Rate", fontsize=12)
plt.title("The Training Performance of Logistic Regression
Classifier", fontsize=14)
plt.legend()
plt.grid() # Add grid for better readability
# Optional: Rotate x-axis labels if there are many values
plt.xticks(rotation=45)
# Show plot
plt.tight_layout() # Adjust layout to make room for rotated labels
plt.show()# Training the Linear SVC for C = 0.9 for best performance
LSVC = LinearSVC(penalty='l2',dual=False, C=0.9)
LSVC.fit(X_train,y_train)
LinearSVC(C=0.9, dual=False)
prediction = LSVC.predict(X_test)
print(classification_report(y_test,prediction))# Assuming `prediction`
is the predicted labels for the test set
```

```
# If you haven't already, ensure to get predictions before this step
prediction = DT.predict(X_test) # Make sure DT is your trained
Decision Tree model
# Storing the evaluation Result of Decision Tree Classifier
model_result = [
'DecisionTree',
accuracy_score(y_test, prediction),
precision_score(y_test, prediction, average='macro'), # Change
average to 'macro'
recall_score(y_test, prediction, average='macro'), # Change
average to 'macro'
f1_score(y_test, prediction, average='macro') # Change
average to 'macro'
]# Append results to the DataFrame
result.loc[len(result)] = model_result

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
# Assuming X_train, y_train, X_test, and y_test are already defined
# Train the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
DT.fit(X_train, y_train)
# Predict probabilities for each class
y_pred_proba = DT.predict_proba(X_test)
# Binarize the true labels for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
# Calculate ROC and AUC for each class
n_classes = y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve (Decision Tree)')
plt.legend(loc='lower right')
plt.show()from sklearn.svm

import LinearSVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# Assuming you have your training data X_train and y_train defined
LSVC = LinearSVC() # Define the LSVC model
LSVC.fit(X_train, y_train) # Fit the model to the training data
prediction = LSVC.predict(X_train) # Make predictions on the training
data
# Use 'macro' or 'weighted' for multiclass evaluation
t_result.append((itr,
accuracy_score(y_train, prediction),
precision_score(y_train, prediction,
average='macro'), # Change to 'macro' or 'weighted'
recall_score(y_train, prediction, average='macro'),
# Change to 'macro' or 'weighted'
f1_score(y_train, prediction, average='macro')))# Change to 'macro' or
'weighted'
itr += 0.1

import matplotlib.pyplot as plt
import pandas as pd
# Assuming t_result is populated with the performance metrics
result_df = pd.DataFrame(t_result, columns=['C', 'Accuracy',
'Precision', 'Recall', 'F1-score'])
# Plotting the training performance
plt.figure(figsize=(10, 8))
plt.plot(result_df['C'], result_df['Accuracy'], label='Accuracy
Score', marker='o', linestyle='-')
plt.plot(result_df['C'], result_df['Precision'], label='Precision
Score', marker='s', linestyle='--')
plt.plot(result_df['C'], result_df['Recall'], label='Recall Score',
marker=
'^', linestyle='-.')
plt.plot(result_df['C'], result_df['F1-score'], label='F1-Score',
marker='d', linestyle=':')
plt.xlabel("Values of C", fontsize=12)
plt.ylabel("Rate", fontsize=12)
plt.title("The Training Performance of Logistic Regression
Classifier", fontsize=14)
plt.legend()
plt.grid() # Add grid for better readability
# Optional: Rotate x-axis labels if there are many values
plt.xticks(rotation=45)
# Show plot
plt.tight_layout() # Adjust layout to make room for rotated labels
plt.show()# Training the Linear SVC for C = 0.9 for best performance
LSVC = LinearSVC(penalty='l2',dual=False, C=0.9)
LSVC.fit(X_train,y_train)
LinearSVC(C=0.9, dual=False)
prediction = LSVC.predict(X_test)
print(classification_report(y_test,prediction))

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# Saving the evaluation Result of Linear SVC
model_result = ['Linear SVC',
accuracy_score(y_test, prediction),
precision_score(y_test, prediction,
average='weighted'), # Use 'weighted' for multiclass
recall_score(y_test, prediction, average='weighted'),
# Use 'weighted' for multiclass
f1_score(y_test, prediction, average='weighted')]
# Use 'weighted' for multiclass
# Append to the results DataFrame
result.loc[len(result)] = model_result

from sklearn.svm import LinearSVC
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
from sklearn.calibration import CalibratedClassifierCV
```

```
import matplotlib.pyplot as plt
import numpy as np
# Assuming X_train, y_train, X_test, and y_test are already defined
# Train the LinearSVC model and calibrate it to get probability
estimatesLSVC = LinearSVC(random_state=42)
calibrated_LSVC = CalibratedClassifierCV(LSVC, method='sigmoid',
cv='prefit')
LSVC.fit(X_train, y_train)
calibrated_LSVC.fit(X_train, y_train)
# Predict probabilities for each class
y_pred_proba = calibrated_LSVC.predict_proba(X_test)
# Binarize the true labels for multiclass ROC calculation
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
# Calculate ROC and AUC for each class
n_classes = y_test_binarized.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:,
i])
auc_score = roc_auc_score(y_test_binarized[:, i], y_pred_proba[:,
i])
plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc_score:.2f})')
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve (LinearSVC)')
plt.legend(loc='lower right')
plt.show()#Printing the DataFrame containing the Testing Performance of
each
Model
result
# Displaying a bar to compare between Models according to their
Accuracyplt.subplots(figsize=(13,5))
sns.barplot(x="Model Name",
y="Accuracy",data=result,palette='hot',edgecolor=sns.color_palette('da
rk',7))
plt.xticks(rotation=90)
plt.title('Models Testing & Evaluation Accuracy Comparison')
plt.show()
# Displaying a bar to compare between Models according to their
Precision
plt.subplots(figsize=(13,5))
sns.barplot(x="Model Name", y="Precision
Score",data=result,palette='hot',edgecolor=sns.color_palette('dark',7)
)
plt.xticks(rotation=90)
plt.title('Models Testing & Evaluation Precision Comparison')
plt.show()# Displaying a bar to compare between Models according to
their Recall
plt.subplots(figsize=(13,5))
sns.barplot(x="Model Name", y="Recall
Score",data=result,palette='hot',edgecolor=sns.color_palette('dark',7)
)
plt.xticks(rotation=90)
plt.title('Models Testing & Evaluation Recall Comparison')
plt.show()# Displaying a bar to compare between Models according to
their F1-
```

```
score
plt.subplots(figsize=(13,5))
sns.barplot(x="Model Name", y="F1-
Score",data=result,palette='hot',edgecolor=sns.color_palette('dark',7)
)
plt.xticks(rotation=90)
plt.title('Models Testing & Evaluation F1-Score Comparison')
plt.show()
# Putting All together in Pipeline
LGR_pipe = Pipeline([('vectorizer',vectorizer),('LGR',LGR)])
# Saving the Pipeline for Later using
pickle.dump(LGR_pipe, open('LGR_Model.sav', 'wb'))
from sklearn import set_config
set_config(display='diagram')
LGR_pipe
Pipeline(steps=[('vectorizer',
TfidfVectorizer(min_df=10, norm=None,
strip_accents='ascii')),
('LGR',
LogisticRegression(C=0.9, max_iter=1000,random_state=41,
solver='sag'))])
# Get the coefficients of the logistic regression model
coefficients = LGR.coef_[0]
# Sort the coefficients by their absolute values
sorted_indices = np.argsort(np.abs(coefficients))[::-1]
top_n = 20 # Change this to display top N coefficients
top_coefficients = coefficients[sorted_indices][:top_n]
top_features = np.array(vectorizer.get_feature_names_out())
[sorted_indices][:top_n]
threshold = 0.0 # Threshold value
colors = ['skyblue' if coef > threshold else 'tomato' for coef in
top_coefficients]
# Plot the top N coefficients and their corresponding features
plt.figure(figsize=(10, 6))
plt.barh(np.arange(top_n), top_coefficients, color=colors)
plt.yticks(np.arange(top_n), top_features)
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Top {} Logistic Regression Coefficients'.format(top_n))
plt.gca().invert_yaxis() # Invert y-axis to display highest
coefficients at the top
plt.show()
```

<u>Labelling the unlabelled dataset after web scraping :</u>

```
import pickle
import pandas as pd
from sklearn.pipeline import Pipeline
# Load the pre-trained model
LGR_Model = pickle.load(open('LGR_Model.sav', 'rb'))
# Load the unlabeled dataset
unlabeled_df = pd.read_csv('youtube_video_comments.csv')
# Define the preprocessing functions if not already defined
# These functions should handle lowercase, stop words removal,
tokenization, and lemmatization
def preprocess_text(text):
if isinstance(text, str): # Check if the text is a string
lower_sen = text.lower() #
Lowercase
```

```
cleaned_sentence = remove_stopwords(lower_sen) #
Remove Stop Words
tken_sentence = word_tokenize(cleaned_sentence) #
Tokenize
lemmatized_sen = lemmatization(tken_sentence) #
Lemmatization
processed_sen = ' '.join(lemmatized_sen) #
Regroup
return processed_sen
return "" # Return an empty string if the text is not valid
# Initialize lists to store results
predicted_labels = []
predicted_probabilities = []
# Iterate over each comment in the dataset
for comment in unlabeled_df['Comment']:
processed_comment = preprocess_text(comment)
# Only predict if there is valid text to process
if processed_comment:
sentiment = LGR_Model.predict([processed_comment])
sentiment_probabilities =
LGR_Model.predict_proba([processed_comment])# Store predicted class and
probability
predicted_class = sentiment[0]
predicted_class_index =
list(LGR_Model.classes_).index(predicted_class)
predicted_probability = sentiment_probabilities[0]
[predicted_class_index]
else:
predicted_class = "Unknown"
predicted_probability = None
predicted_labels.append(predicted_class)
predicted_probabilities.append(predicted_probability)
# Add the predictions to the original dataset
unlabeled_df['predicted_label'] = predicted_labels
unlabeled_df['predicted_probability'] = predicted_probabilities
# Save the labeled dataset
unlabeled_df.to_csv('labeled_youtube_comments.csv', index=False)
print("Labeled dataset saved as labeled_youtube_comments.csv")

import pandas as pd
import plotly.graph_objects as go
# Load the labeled dataset
df = pd.read_csv('labeled_youtube_comments.csv')
# Get the label distribution using value_counts
labels = df['predicted_label'].value_counts().index
values = df['predicted_label'].value_counts().values
# Create the pie chart (donut chart by setting the hole)
fig = go.Figure(data=[go.Pie(labels=labels,
values=values,
hole=0.6, # This creates the "donut"
effect
textinfo='percent+label')])
# Update the layout for a clean look
fig.update_layout(title='Distribution of Classes in the Dataset',
title_font_size=20, title_x=0.5, width=750,
height=550)
# Display the pie chart
fig.show()import seaborn as sns
```

```
df['comment_length'] = df['Comment'].apply(lambda x:
len(str(x).split())) plt.title('Comment Length by Sentiment')
# Count words in each comment
plt.show()
sns.boxplot(data=df, x='predicted_label', y='comment_length')
plt.xlabel('Sentiment')
plt.ylabel('Comment Length (words)')from wordcloud import WordCloud

import matplotlib.pyplot as plt
for sentiment in df['predicted_label'].unique():
text = ' '.join(df[df['predicted_label'] == sentiment]
['Comment'].dropna().astype(str))
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation='bilinear')
plt.title(f'Word Cloud for {sentiment} Sentiment')
plt.axis('off')
plt.show()##0-negative 1 - neutral 2- positive

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='predicted_probability',
hue='predicted_label', multiple="stack", bins=20)
plt.title('Sentiment Probability Distribution')
plt.xlabel('Prediction Probability')
plt.ylabel('Number of Comments')
plt.show()
```