Project 2 Report

HateTron

HATE SPEECH DETECTION CHATBOT

- By Group 8

BML MUNJAL UNIVERSITY™

Department of CSE
SOET

April 2023

Team Members:

S.no	Name	Registration No.	Section
1.	Anish Borkar	210C2030036	CSE 1
2.	Anoushka Srivastava	210C2030048	CSE 1
3.	Ankur Kaushal	210C2030218	CSE 4
4.	Dyuti Dasmahapatra	210C2030142	CSE 1
5.	Godavarthi Sai Nikhil	210C2030030	CSE 1
6.	Nichenametla Karthik Raja	210C2030113	CSE 1
7.	Vandamasu Sai Sumanth	210C2030060	CSE 2
8.	D Veera Harsha Vardhan Reddy	210C2030061	CSE 2
9.	Himanshu Sharma	210C2030207	CSE 4

Table Of Contents

Introduction:

1.1_Introduction

a. Motivation:

In contemporary international, hate speech, intolerance, and bigotry are all too time-honoured, specifically in on-line settings like social media websites, chat rooms, and boards. These terrible actions not simplest exacerbate warfare and hostility, but additionally they have adverse consequences at the goals' mental health and general wellbeing. Making a hate speech detection chatbot mobile app can be a useful and significant attempt to address this developing challenge. To address the worrying surge of hate speech on online platforms, a hate speech detection chatbot mobile app is being developed. The development of a mobile chatbot app with hate speech detection features intends to solve this problem by giving users a tool for quickly identifying and flagging instances of hate speech. This gives people the ability to combat hate speech by reporting it, obstructing or filtering it, and encouraging a more civil and welcoming online community.

The app's capability to discover hate speech proactively whilst customers write, or upload feedback may be a powerful deterrent to the unfold of hate speech and promote superb on-line interactions. Furthermore, the incentive for growing a hate speech detection chatbot cellular app stem from the choice to guard prone communities and promote social justice. Hate speech discriminates in opposition to human beings based on their race, ethnicity, religion, gender, sexual orientation, and other characteristics, inflicting emotional misery and perpetuating discrimination (Kumar, 2022). The chatbot mobile app can guard prone users by way of detecting and flagging hate speech in real time, allowing them to take suitable defensive measures and are searching for help. Furthermore, the app can raise consumer consciousness approximately the bad impact of hate speech with the aid of imparting educational assets and records on a way to correctly reply to hate speech. The hate speech detection chatbot mobile app can help to create a greater respectful and equitable on-line community for all customers via selling social justice and inclusivity.

b. What is the product?

i. Why is this product required?

ii. Any real – world implications?

The product is a hate speech detection chatbot mobile app that detects hate speech when users write or upload comments on online platforms.

- Because of the increasing incidence of hate speech in social media and online groups, the product is required. Hate speech may have severe results, inclusive of cyberbullying, discrimination, and person division. By detecting and flagging hate speech in actual time, the app pursuits to provide a proactive strategy to this hassle. It offers customers the capability to fight hate speech by using reporting it, blockading or filtering it, and selling more respectful and inclusive on-line surroundings. The product is required to foster social justice, defend vulnerable groups, and promote fine online interactions.

- The hate speech detection chatbot cell app has enormous real-global implications. For starters, by identifying and addressing instances of hate speech, it could assist to create safer and extra inclusive online surroundings for all customers. This can help to mitigate the poor outcomes of hate speech, including emotional distress, discrimination, and department within on-line groups. Second, the app can boost attention and educate users about the bad effect of hate speech, as well as provide sources and statistics on a way to correctly reply. This can assist to elevate consciousness and understanding of hate speech even as additionally selling recognize and inclusivity among users. Finally, the app may additionally have prison and ethical implications because it encourages customers to record hate speech, which can result in accountability and results for hate speech perpetrators, in addition to sell compliance with hate speech legal guidelines and rules in on line platforms.

1.2 Survey of existing solutions

Perspective API by Jigsaw

Perspective API uses ML models to identify hate speech. Based on the perceived impact a text/phrase can have in a conversation, the models score/rate the text/phrase. This works as feedback given to commenters by developers and publishers, help moderators in reviewing comments or help readers filtering out language.

Dataset: The dataset, containing over 17 million comments from various sources like news articles, social media, etc., is used by Jigsaw. This dataset helps in training ML models for Perspective API.

Dataset Collection: Using Web Scraping, comments were retrieved from publicly available online sources to form the dataset. The comments collected were of multiple languages and covered a wide range of topics and themes.

Methodology: A machine learning approach was used to develop Perspective API. Through the dataset, ML models including Supervised and Unsupervised learning, were used for identifying patterns of the potentially toxic or abusive content. The models were trained based on Text Content, User Behaviour and Contextual Information.

Usability Evaluation: Usability evaluations were conducted with developers, content moderators and online community managers to gather feedback on the performance, accuracy and ease of use. Using this feedback, improvements were made.

User Experience Evaluation: User Experience evaluations were also conducted to identify potential issues while using the tool. For this process, feedback was gathered through Surveys, Interviews and Usability Testing sessions.

Limitations: One limitation was the possibility of false positives or false negatives in identifying toxic or abusive content. The tool may not be able to accurately identify context dependent language and can have biases. Other limitation could be that the performance can vary across languages.

HateOmeter:

HateOmeter is a hate speech detection tool which uses Machine learning algorithms for the analysing texts. It uses an combination of rule and Machine Learning approach to detect the hate speech in the posts,comments in the social media apps.

Sift:

Sift is an hate speech detection tool uses the machine learning algorithms.It detects the spam,harassment. It provides the content moderation and filtering to detect and also block hate speech.

Detection and Prevention: It uses ML algorithms to analyse data from user behaviour, transaction data, etc., to detect fraud, abuse or malicious activities. This helps businesses identify and remove suspicious activities, reducing the risk of fraud and abuse.

Dataset: The models are trained on large datasets containing ranges of interaction data and online transaction. These dataset are collected from customers, who give anonymized data to help improve the accuracy.

Dataset Collection: The data is collected from customers, who integrate Sift's API or SDK into their online platforms. The collected data includes transaction details, user interactions, etc.

Methodology: Sift uses Supervised and Unsupervised learning to classify data based on patterns related to fraud, abuse or malicious activities. Models are trained on large datasets using feature engineering, training and validation.

Usability Evaluation: Sift engages with customers, business that use its products. This is done to gather feedback on usability, effectiveness and performance of the tool.

User Experience Evaluation: It also gathers feedbacks from end users, that are protected by Sift's solutions. This includes feedback on the effectiveness of fraud and abuse detection, false positive rates; overall user satisfaction.

Limitations: One limitation includes false positives or false negatives in detecting fraud, biases in training data or model predictions. Another could be the effectiveness of tool can depend on the quality of data provided by customers.

HateSonar:

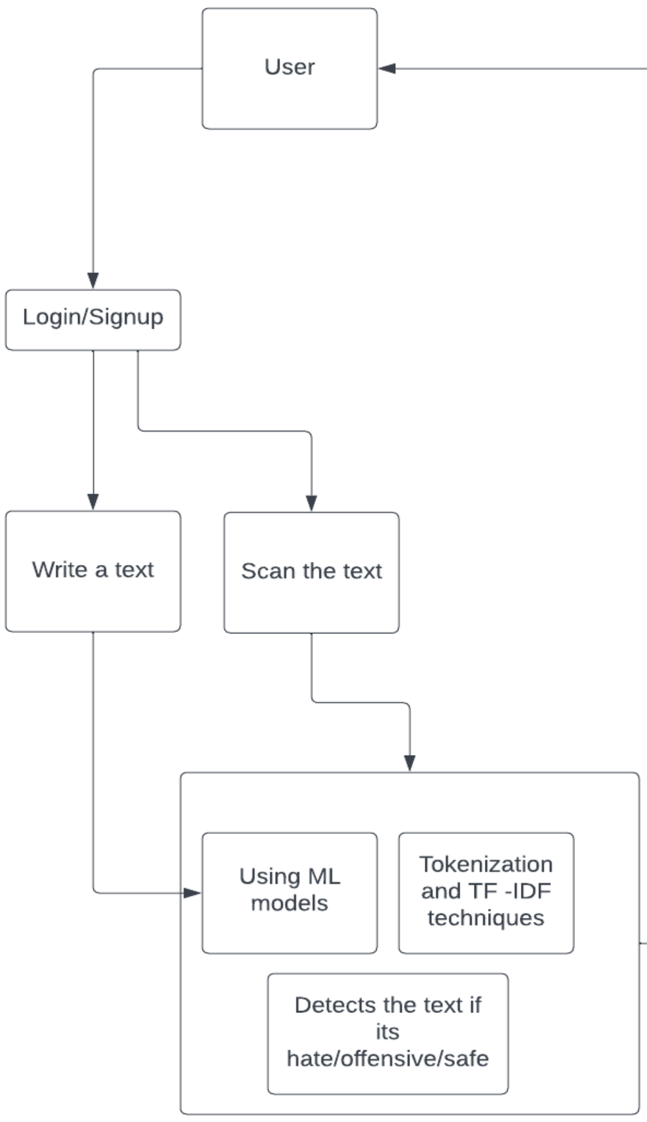
It is a hate speech detection API which was developed by Open Web. It uses machine learning algorithms to analyse text data and detect hate speech in the comments. It gives a toxicity score and classification of comments into categories, including hate speech, to help platforms identify and take action against hate speech.

Hate Speech Blocker:

Hate speech blocker is an browser extension.It was developed by Online Hate Prevention Institute. Here it uses the machine learning algorithms to detect and also block hate speech on the social media platforms.It identifies hate speech in real-time and provides the users an option to report or block hate speech content.

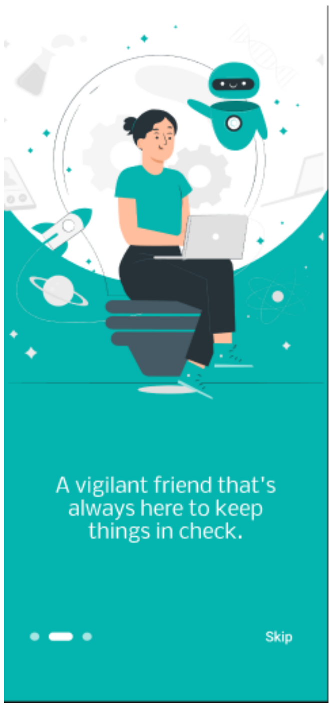
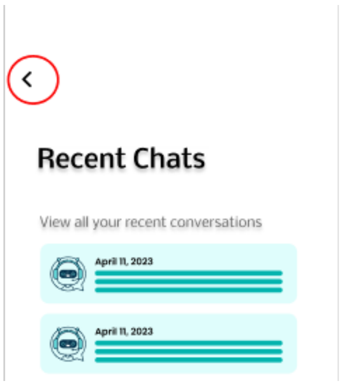
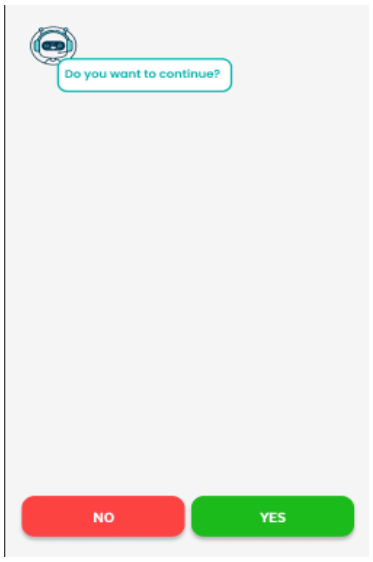
2. System

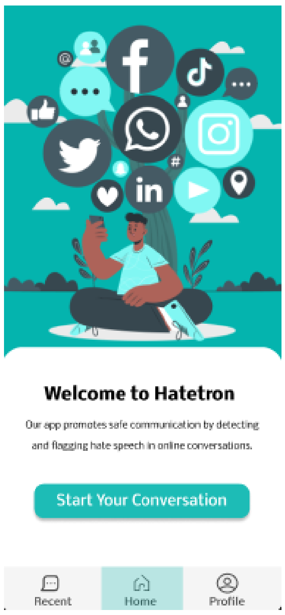
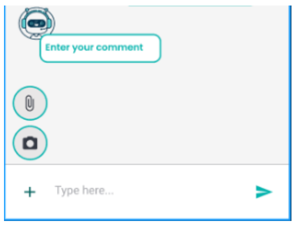
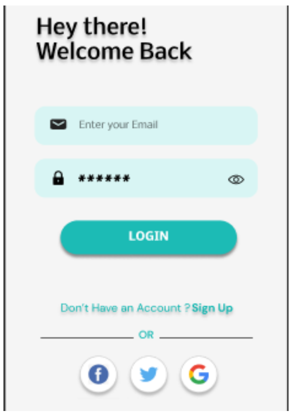
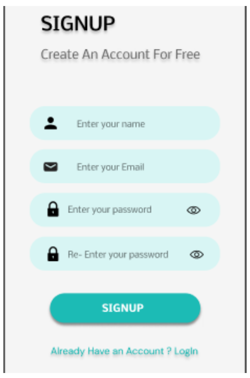
System Architecture

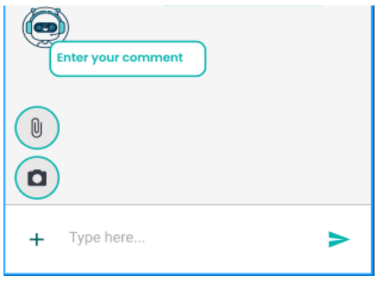
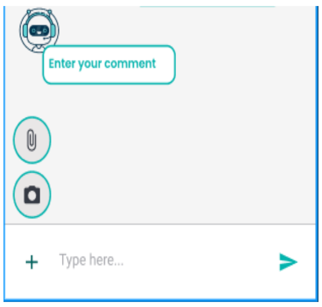
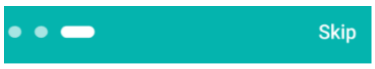
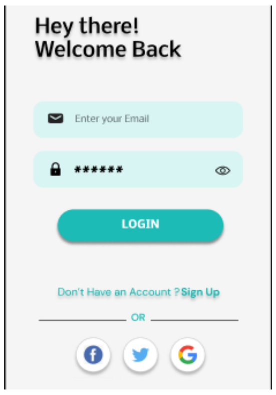
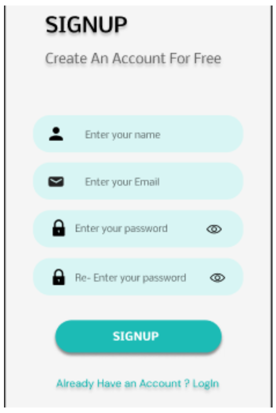


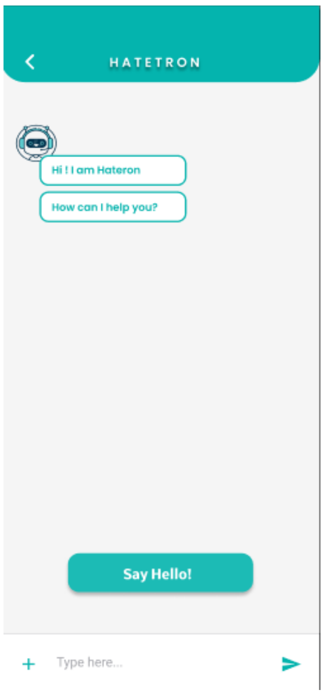
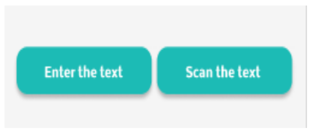
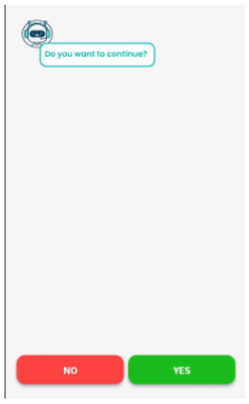
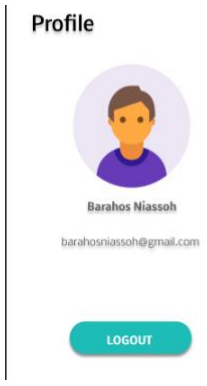
User
Login/Signup
Write a text
Scan the text
Using ML models
Tokenization and TF -IDF techniques
Detects the text if its hate/offensive/safe
Response in HATETRON app
Sends the popup as This is hate speech This is safe speech This is offensive speech

Features in the System and Significance of designing each Feature (Table)

Feature within the application	**Associated Design Principle**	**Significance of the feature**	**Associated Image in the app**
1) User friendly interface	User - Centered design	Aims to increase user satisfaction and management by making sure the app is simple to use, navigate, and understand for users of all ability levels. (Norman, 2013)	 A vigilant friend that's always here to keep things in check.
2) Back option	Affordance, Consistency	When a user navigates backwards using the back button it redirects to the pages that user visited recently.	 Recent Chats View all your recent conversations April 11, 2023 April 11, 2023
3) Yes/No Buttons	Visibility	It allows the persons to actually continue into the app after acquiring some results.	 Do you want to continue?

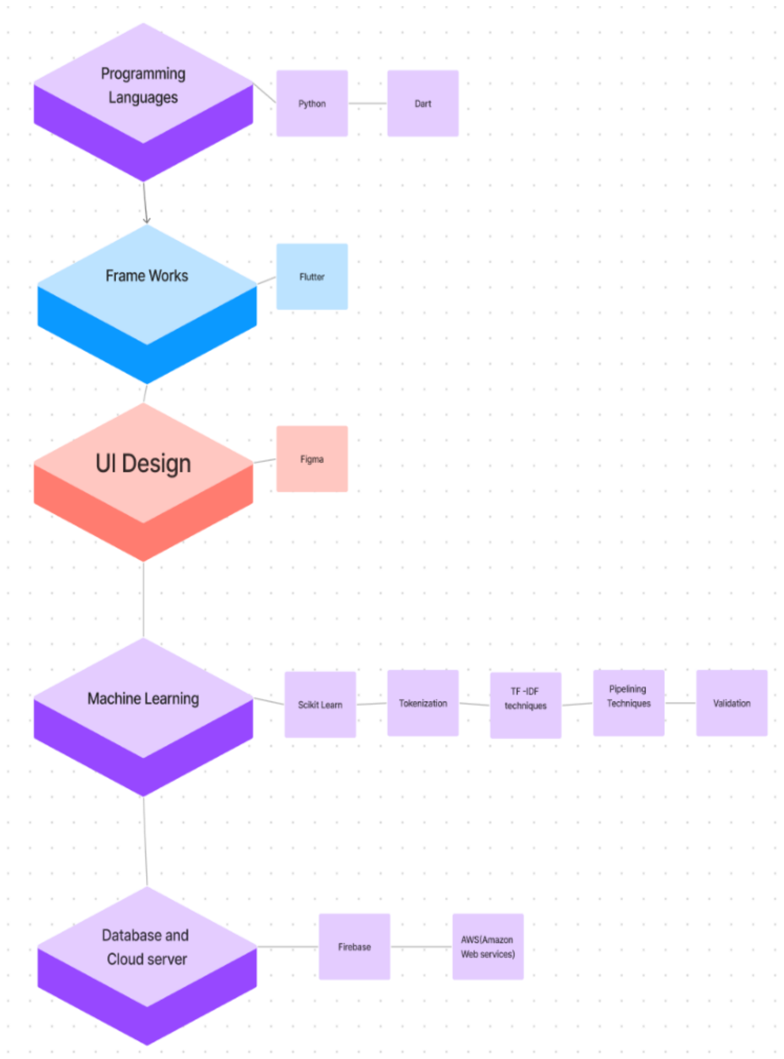
Feature within the application	**Associated Design Principle**	**Significance of the feature**	**Associated Image in the app**
4) Start Your Conversation	Visibility, Affordance	It navigates to the front page of the app interface.	 Welcome to Hatetron Our app promotes safe communication by detecting and flagging hate speech in online conversations.
5) "+" Button for camera and image attached for text detection	Visibility, Affordance, Feedback, Consistency	Gives options to either to capture the image or an attached image.	
6) Login using Facebook, Google, or Twitter.	Affordance, Consistency, Visibility	The user can login in the application through Facebook, Google or Twitter.	 Hey there! Welcome Back Enter your Email ****** LOGIN Don't Have an Account ? Sign Up OR
7) Sign Up	Affordance, Consistency, Visibility & Feedback	It redirects the user to sign in page where user enters the details like email, name and password.	 SIGNUP Create An Account For Free Enter your name Enter your Email Enter your password Re- Enter your password SIGNUP Already Have an Account ? Login

Feature within the application	Associated Design Principle	Significance of the feature	Associated Image in the app
8) Send button	Visibility, Affordance, Feedback & Consistency	This button transmits the input provided in the chat field.	
9) Type Here Chat field	Visibility, Affordance, Consistency & Feedback	The field enables the user to write text to chat with other users.	
10) Skip button on the three pages	Visibility, Affordance, Consistency	The user can skip the pages and directly head to the welcome page.	
11) Login via username and password; Login via Google, Facebook or Twitter.	Constraints	Ensures that the user can login only after providing the correct username and password.	 Hey there! Welcome Back Enter your Email ****** LOGIN Don't Have an Account ? Sign Up OR
12) Creating Usernames and Passwords at the Signup Page	Constraints	Ensures that the username is unique and the password complies with the set constraints; ensures its safe	 SIGNUP Create An Account For Free Enter your name Enter your Email Enter your password Re- Enter your password SIGNUP Already Have an Account ? Login

Feature within the application	**Associated Design Principle**	**Significance of the feature**	**Associated Image in the app**
13) Starting a conversation	Constraints	A set command that is utilized to activate the chatbot.	 HATETRON Hi ! I am Hateron How can I help you? Say Hello! Type here...
14) Providing input via **Scan** or **Type** text feature	Constraints	Ensures the input is in the correct format and helps to provide accurate results.	
15) Choose either the Yes or No option to continue or discontinue using the chatbot.	Constraints	It helps the chatbot know if the user wants to provide more input or not.	
16) Only one Logout option in the Profile Tab	Constraints and User-Cantered Design	Only one logout option makes it easier for users to remember how to logout and tells the app that the user has logged out and needs to login again in the next session.	 Profile Barahos Niassoh barahosniassoh@gmail.com LOGOUT

Technology Stack

- Programming Languages: Python is used to implement Machine Learning model, Dart is used in flutter.
- Framework: The Flutter framework is used to build the user interface of the app.
- UI Design Tools: We used Figma to design the interface of the application.
- Machine Learning: This includes methods such as tokenization, TF-IDF techniques, Pipelining techniques.
- Databases: AWS and Firebase



3. Methodology

ML CRISP – DM

Business Understanding

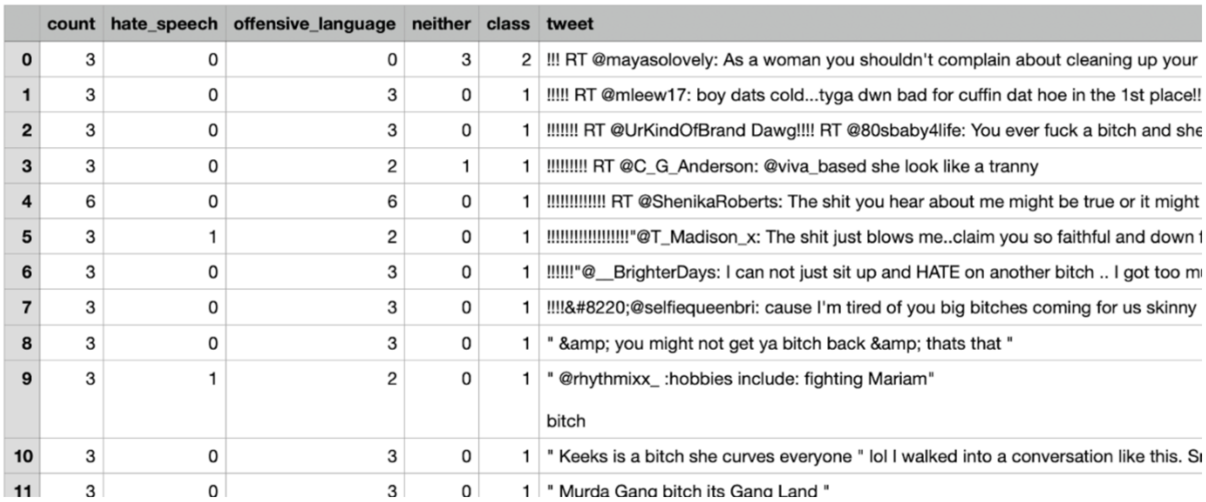
The problem statement draws attention to the problem of harsh language and hate speech on social media sites, which can negatively affect online discussions and obstruct productive communication. A real-time chatbot that can identify hate speech and highlight comments that use derogatory language, slurs, or discriminatory statements is the suggested solution.

This method seeks to enhance online discourse by minimising hate speech and encouraging user respect. The numerous organisations that are engaged in eliminating hate speech on their online platforms make up the target audience for this solution. These organisations include social media corporations, news organisations, schools, government agencies, and non-profits.

The approach can also be helpful for anyone who utilise online platforms and want to monitor and stop hate speech in such areas, such as influencers, students, and teachers. Overall, the creation of a real-time chatbot that can identify hate speech can help to foster a more secure and civil online community.

Data Understanding

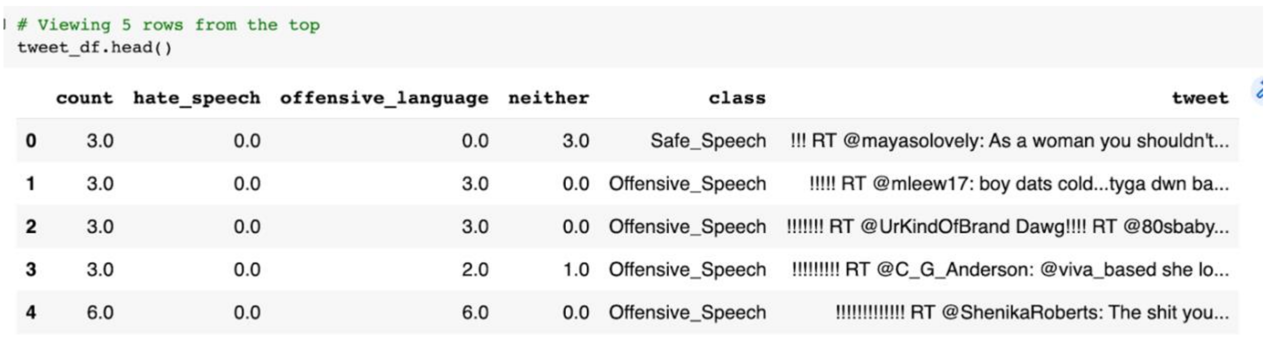
Understanding the available statistics and how it may be utilised to educate the model is essential to create a actual-time chatbot that may become aware of hate speech. The textual facts on social media websites like Twitter, Facebook, and Instagram is a rich source of examples of hate speech. For the purpose of detecting hate speech, gadget studying fashions can be trained the use of this data. We can also do the schooling of hate speech detection models the use of person-generated content, consisting of feedback on articles or weblog posts. These remarks may use derogatory language, racial slurs, or other discriminatory statements that can be known as hate speech. These can be accumulated from form of web sites, blogs, news websites and forums.



	count	hate_speech	offensive_language	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!
2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby4life: You ever fuck a bitch and she
3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she look like a tranny
4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you hear about me might be true or it might
5	3	1	2	0	1	!!!!!!!!!!!!!!!!!!"@T_Madison_x: The shit just blows me..claim you so faithful and down
6	3	0	3	0	1	!!!!!!"@__BrighterDays: I can not just sit up and HATE on another bitch .. I got too
7	3	0	3	0	1	!!!!&#8220;@selfiequeenbri: cause I'm tired of you big bitches coming for us skinny
8	3	0	3	0	1	" &amp; you might not get ya bitch back &amp; thats that "
9	3	1	2	0	1	" @rhythmixx_ :hobbies include: fighting Mariam" bitch
10	3	0	3	0	1	" Keeks is a bitch she curves everyone " lol I walked into a conversation like this.
11	3	0	3	0	1	" Murda Gang bitch its Gang Land "

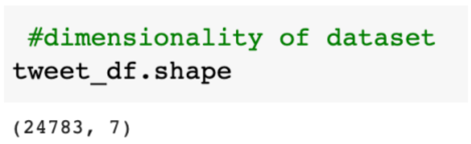
Fig. Snapshot of dataset

Below is a snapshot of the dataset using pandas.



```
# Viewing 5 rows from the top
tweet_df.head()
```

	count	hate_speech	offensive_language	neither	class	tweet
0	3.0	0.0	0.0	3.0	Safe_Speech	!!! RT @mayasolovely: As a woman you shouldn't...
1	3.0	0.0	3.0	0.0	Offensive_Speech	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3.0	0.0	3.0	0.0	Offensive_Speech	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3.0	0.0	2.0	1.0	Offensive_Speech	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6.0	0.0	6.0	0.0	Offensive_Speech	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...



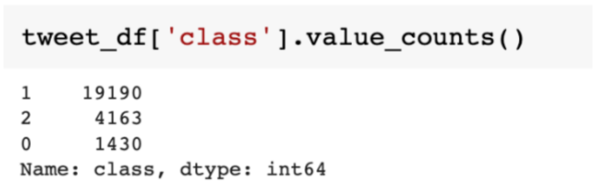
```
#dimensionality of dataset
tweet_df.shape
```

```
(24783, 7)
```

Fig. Dataset in Pandas with dimensionality

The dataset has 24,783 rows and 7 columns: count, hate_speech, offensive_language, neither, class and tweet.

- Index
- Count: range of CrowdFlower customers who coded each tweet (min is 3, occasionally greater customers)
- Hate_speech: quantity of CF users who judged the tweet to be hate speech.
- Offensive_language: number of CF users who judged the tweet to be offensive.
- Neither: quantity of CF users who judged the tweet to be neither offensive nor non-offensive
- Class: elegance label for majority of CF users (0 -> Hate, 1 -> Offensive, 2 -> Safe)
- Tweet: text tweet



```
tweet_df['class'].value_counts()
```

```
1    19190
2     4163
0     1430
Name: class, dtype: int64
```

Fig. Number of rows in each class

The dataset has 19,190 rows of offensive speech, 4,163 rows of safe speech and 1,430 rows of hate speech.

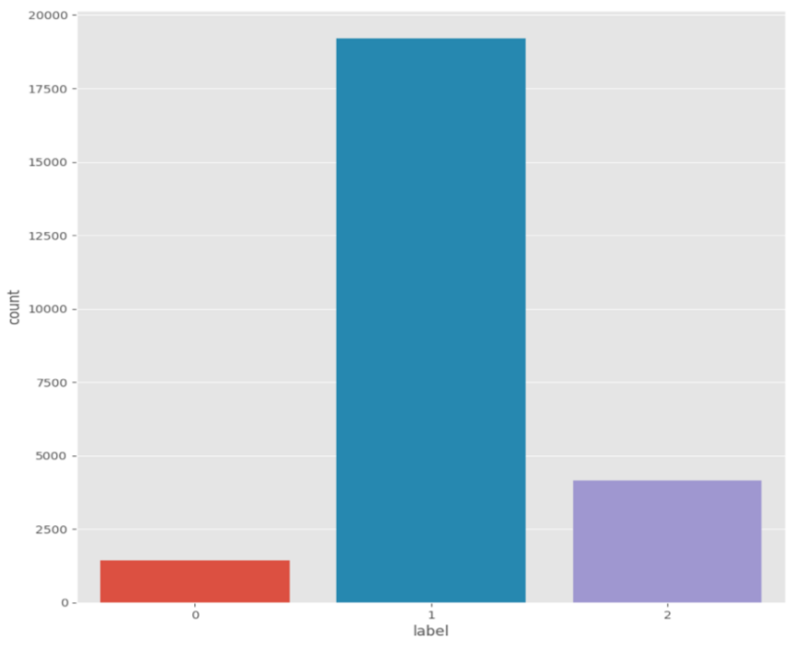


Fig. Countplot of Labels

The dataset has too many rows of “Offensive Language” compared to “Hate Speech” and “Neither”. This can cause poor predictions for “Hate Speech” and “Neither”. (abhishekm, 2022)

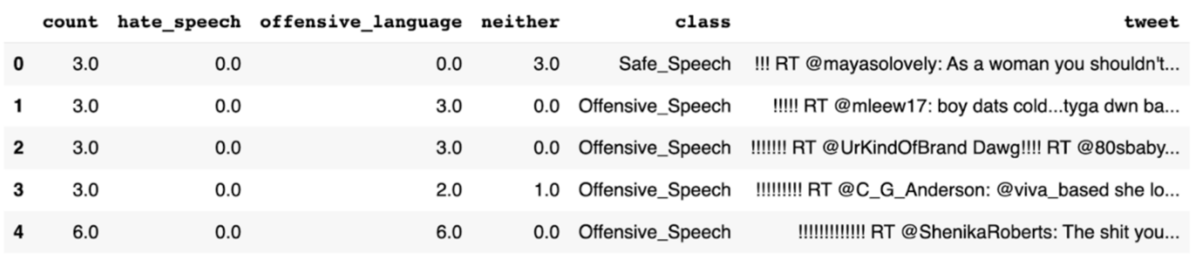
We need to build a classification model for predicting an entered text. The target is “Class” column in the dataset, carrying the labels the model should predict.

We require a labelled dataset to train the classification model. The input column is the “Tweet” column containing the texts the model will utilize to predict the output, “Class” column.

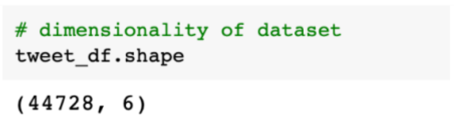
The “Count”, “Hate Speech”, “Offensive Language” and “Neither” columns aren’t necessary for classification. For model training and evaluation, we can ignore these columns and use the “Tweet” and “Class” columns.

Data Processing

We took text inputs from different datasets and merged them to form a balanced dataset to resolve the issue.



	count	hate_speech	offensive_language	neither	class	tweet
0	3.0	0.0	0.0	3.0	Safe_Speech	!!! RT @mayasolovely: As a woman you shouldn't...
1	3.0	0.0	3.0	0.0	Offensive_Speech	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3.0	0.0	3.0	0.0	Offensive_Speech	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3.0	0.0	2.0	1.0	Offensive_Speech	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6.0	0.0	6.0	0.0	Offensive_Speech	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

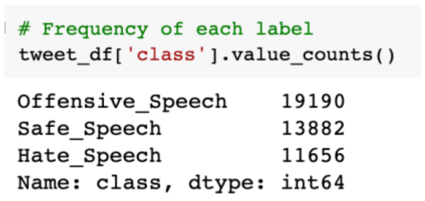


```
# dimensionality of dataset
tweet_df.shape
```

```
(44728, 6)
```

Fig. Updated Dataset in Pandas with dimensionality

The dataset has 44,728 rows and 6 columns.



```
# Frequency of each label
tweet_df['class'].value_counts()
```

```
Offensive_Speech    19190
Safe_Speech         13882
Hate_Speech         11656
Name: class, dtype: int64
```

Fig. Updated number of rows in each class

The dataset has three classes:

19190 is classified as Offensive_Speech
13882 is classified as Safe_Speech
11656 is classified as Hate_Speech

Visualization of the frequency of each label

```
# Count Plot
fig = plt.figure(figsize=(10,10))
sns.countplot(x='label', data = tweet_df)
```

```
<Axes: xlabel='label', ylabel='count'>
```

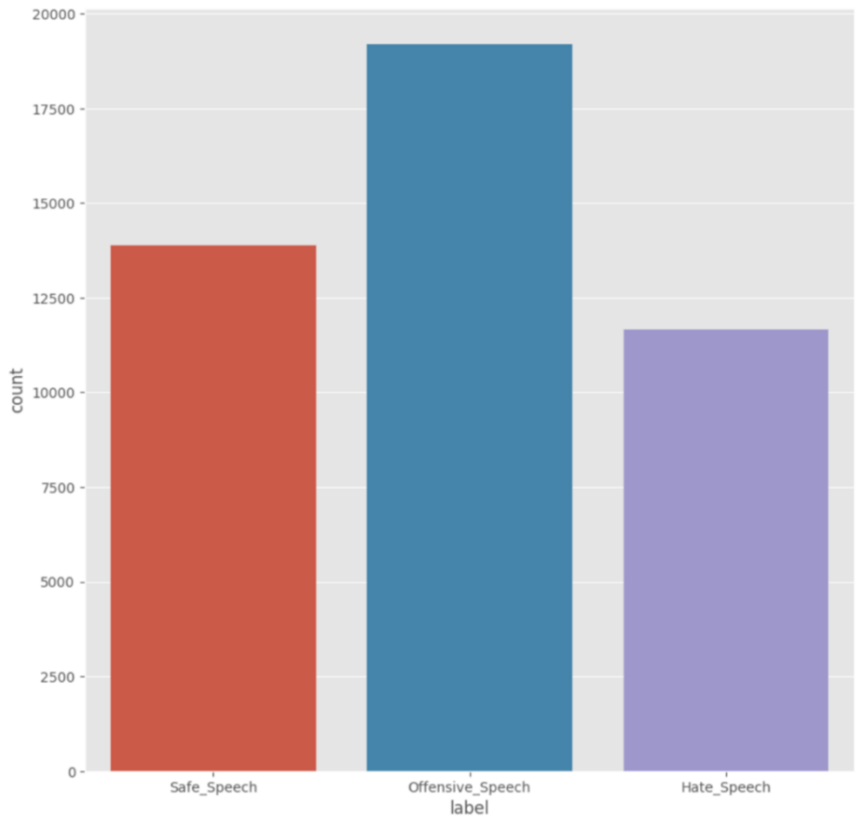
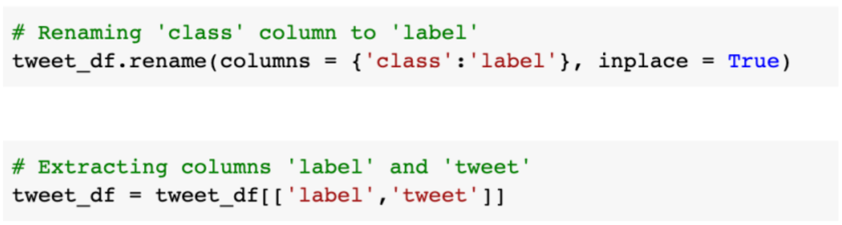


Fig. Updated Countplot of Labels

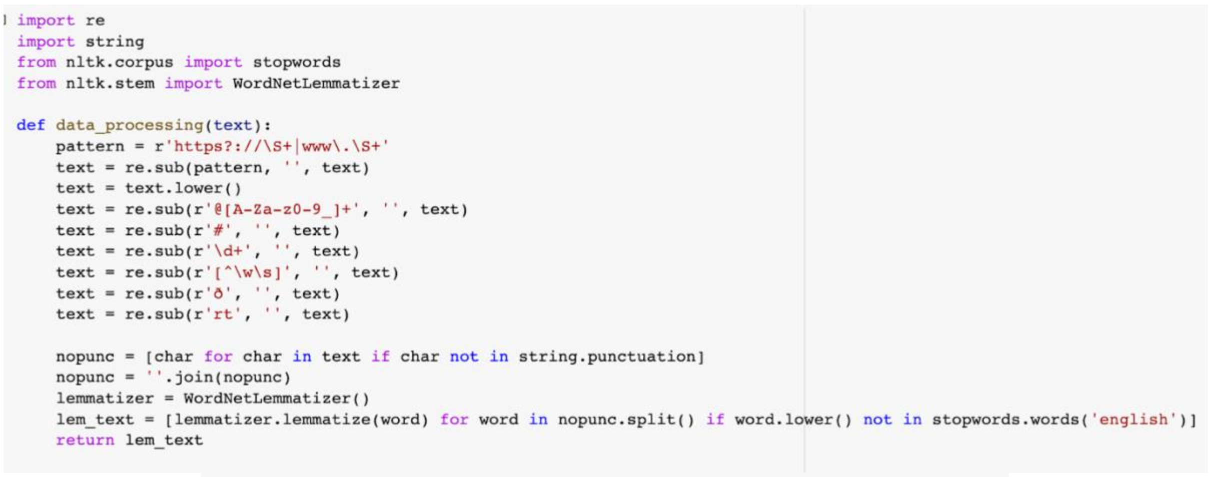


```
# Renaming 'class' column to 'label'
tweet_df.rename(columns = {'class':'label'}, inplace = True)
```

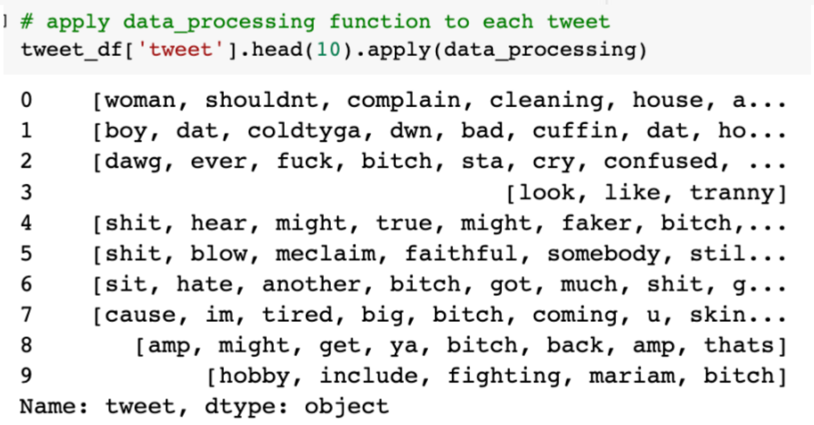
```
# Extracting columns 'label' and 'tweet'
tweet_df = tweet_df[['label','tweet']]
```

Fig. Renaming and Extracting columns

We have to done the data cleaning and pre-processing to get the data ready for training. This involves removing irrelevant information, such as URLs and usernames, and converting the text data into a standardized format. We also performed text normalization by converting all the text to lowercase, removing punctuation and stop words, and lemmatizing the text to reduce the vocabulary size.



```
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def data_processing(text):
    pattern = r'https?://\S+|www\.\S+'
    text = re.sub(pattern, '', text)
    text = text.lower()
    text = re.sub(r'@[A-Za-z0-9_]+', '', text)
    text = re.sub(r'#', '', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'ð', '', text)
    text = re.sub(r'rt', '', text)

    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    lemmatizer = WordNetLemmatizer()
    lem_text = [lemmatizer.lemmatize(word) for word in nopunc.split() if word.lower() not in stopwords.words('english')]
    return lem_text
```



```
# apply data_processing function to each tweet
tweet_df['tweet'].head(10).apply(data_processing)
```

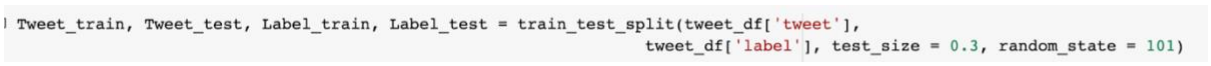
```
0    [woman, shouldnt, complain, cleaning, house, a...
1    [boy, dat, coldtyga, dwn, bad, cuffin, dat, ho...
2    [dawg, ever, fuck, bitch, sta, cry, confused, ...
3                                  [look, like, tranny]
4    [shit, hear, might, true, might, faker, bitch,...
5    [shit, blow, meclaim, faithful, somebody, stil...
6    [sit, hate, another, bitch, got, much, shit, g...
7    [cause, im, tired, big, bitch, coming, u, skin...
8       [amp, might, get, ya, bitch, back, amp, thats]
9            [hobby, include, fighting, mariam, bitch]
Name: tweet, dtype: object
```

Fig. Text Processing function and example to show its working

The tweets column of the dataset has been pre-processed and is now ready for modelling.

Modelling:

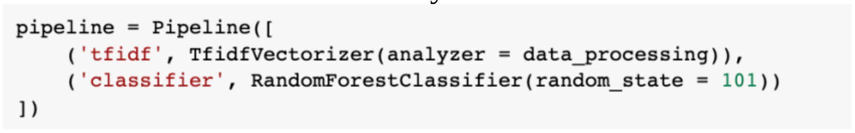
The dataset has been divided into training and testing sets, with test size equal to 0.3 and random state as 101.



```
Tweet_train, Tweet_test, Label_train, Label_test = train_test_split(tweet_df['tweet'],
                                                                    tweet_df['label'], test_size = 0.3, random_state = 101)
```

Fig. Training and Testing the Dataset

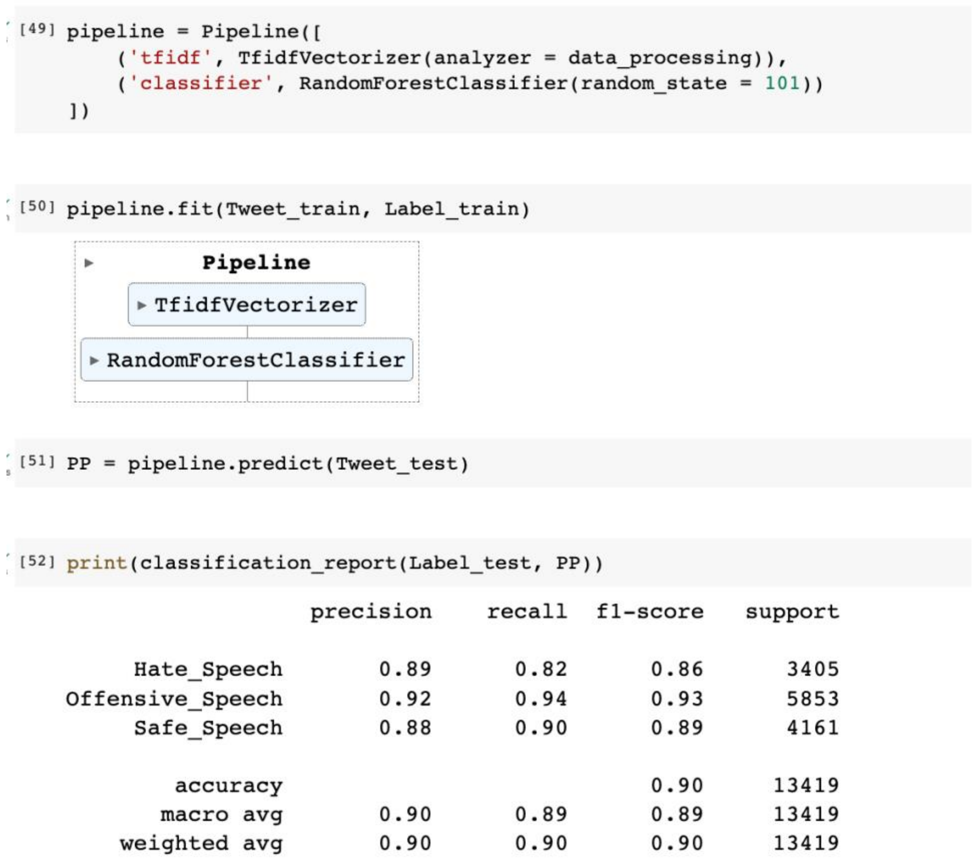
We have used tf-idf for feature extraction and 3 models for classification : Random Forest, Logistic Regression and Multinomial Naive Bayes.



```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(analyzer = data_processing)),
    ('classifier', RandomForestClassifier(random_state = 101))
])
```

Fig. Defining Pipeline (Random Forest)

A pipeline has been created, which uses tfidf for feature extraction and Random Forest as classifier. The Tfidf Vectorizer is a feature extraction method that converts text data into a numerical representation by creating a matrix of TF-IDF (term frequency-inverse document frequency) values for each document. The "analyzer" parameter is set to "data processing," which is used to pre-process and clean the text data before the vectorization step. (Kumar, 2022)
The Random Forest Classifier is a type of machine learning algorithm that is used for classification tasks, with random state equal to 101.



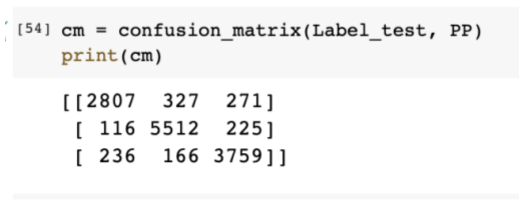
```
[49] pipeline = Pipeline([
         ('tfidf', TfidfVectorizer(analyzer = data_processing)),
         ('classifier', RandomForestClassifier(random_state = 101))
     ])

[50] pipeline.fit(Tweet_train, Label_train)
```

```
[51] PP = pipeline.predict(Tweet_test)

[52] print(classification_report(Label_test, PP))

                  precision    recall  f1-score   support

     Hate_Speech       0.89      0.82      0.86      3405
Offensive_Speech       0.92      0.94      0.93      5853
     Safe_Speech       0.88      0.90      0.89      4161

        accuracy                           0.90     13419
       macro avg       0.90      0.89      0.89     13419
    weighted avg       0.90      0.90      0.90     13419
```

Fig. Fitting and predicting the data through pipeline and Classification Report

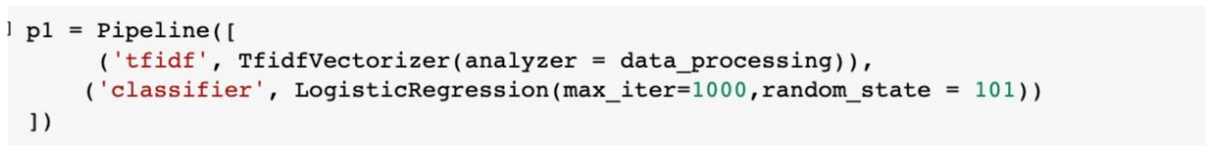
After fitting pipeline, we use that to predict the testing set. Later, we print the classification report, which gives an accuracy of **90%**.

Confusion matrix of the above model:



```
[54] cm = confusion_matrix(Label_test, PP)
     print(cm)

[[2807  327  271]
 [ 116 5512  225]
 [ 236  166 3759]]
```

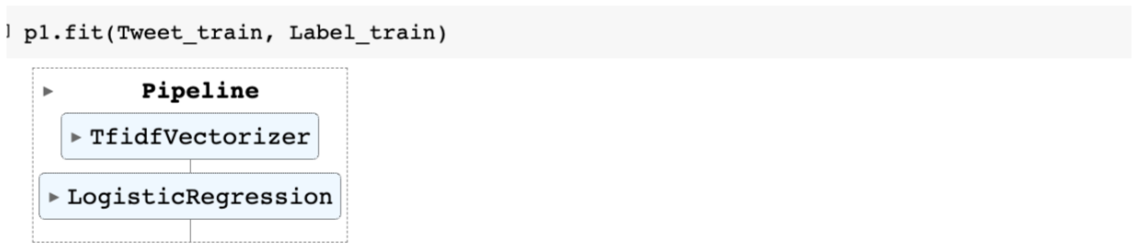
Fig. Confusion Matrix (Random Forest)



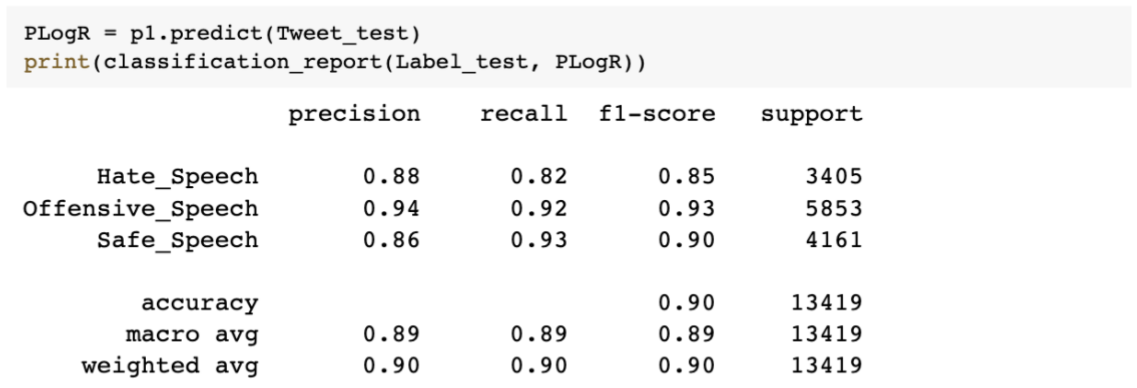
```
p1 = Pipeline([
     ('tfidf', TfidfVectorizer(analyzer = data_processing)),
    ('classifier', LogisticRegression(max_iter=1000,random_state = 101))
 ])
```

Fig. Defining Pipeline (Logistic Regression)

A pipeline has been created, which uses tf idf for feature extraction and Logistic Regression as classifier, with max iterations set to 1000 and random state set to 101.



```
p1.fit(Tweet_train, Label_train)
```

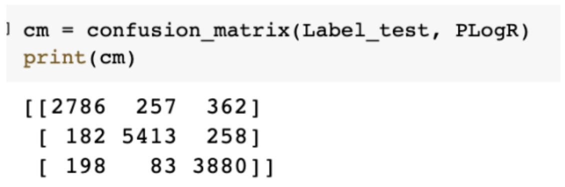


```
PLogR = p1.predict(Tweet_test)
print(classification_report(Label_test, PLogR))

                  precision    recall  f1-score   support

     Hate_Speech       0.88      0.82      0.85      3405
Offensive_Speech       0.94      0.92      0.93      5853
     Safe_Speech       0.86      0.93      0.90      4161

        accuracy                           0.90     13419
       macro avg       0.89      0.89      0.89     13419
    weighted avg       0.90      0.90      0.90     13419
```

Fig. Fitting and predicting the data through pipeline and Classification Report

After fitting the pipeline, we use it to predict the testing set. We then print the classification report, which gives an accuracy of 90%.

Below is the confusion matrix of the model:



```
cm = confusion_matrix(Label_test, PLogR)
print(cm)

[[2786  257  362]
 [ 182 5413  258]
 [ 198   83 3880]]
```

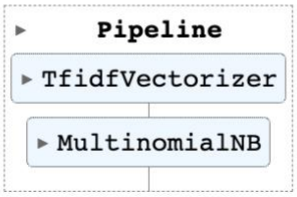
Fig. Confusion Matrix (Logistic Regression)

```
p2 = Pipeline([
     ('tfidf', TfidfVectorizer(analyzer = data_processing)),
    ('classifier', MultinomialNB())
])
```

Fig. Defining Pipeline (Multinomial Naïve Bayes)

A pipeline has been created, which uses tfidf for feature extraction and multinomial naive bayes as classifier.

```
p2.fit(Tweet_train, Label_train)
```



```
PNB = p2.predict(Tweet_test)
print(classification_report(Label_test, PNB))
```

```
                  precision    recall  f1-score   support

      Hate_Speech       0.92      0.76      0.83      3405
 Offensive_Speech       0.75      0.98      0.85      5853
      Safe_Speech       0.93      0.66      0.77      4161

         accuracy                           0.82     13419
        macro avg       0.87      0.80      0.82     13419
     weighted avg       0.85      0.82      0.82     13419
```

Fig. Fitting and predicting the data through pipeline and Classification Report

After fitting the pipeline, we use it to predict the testing set. We then print the classification report, which gives an accuracy of 82%.

Below is the confusion matrix of the model:

```
cm = confusion_matrix(Label_test, PNB)
print(cm)
```

```
[[2587  681  137]
 [  52 5733   68]
 [ 185 1229 2747]]
```

Fig. Confusion Matrix (Multinomial Naïve Bayes)

Data Validation:

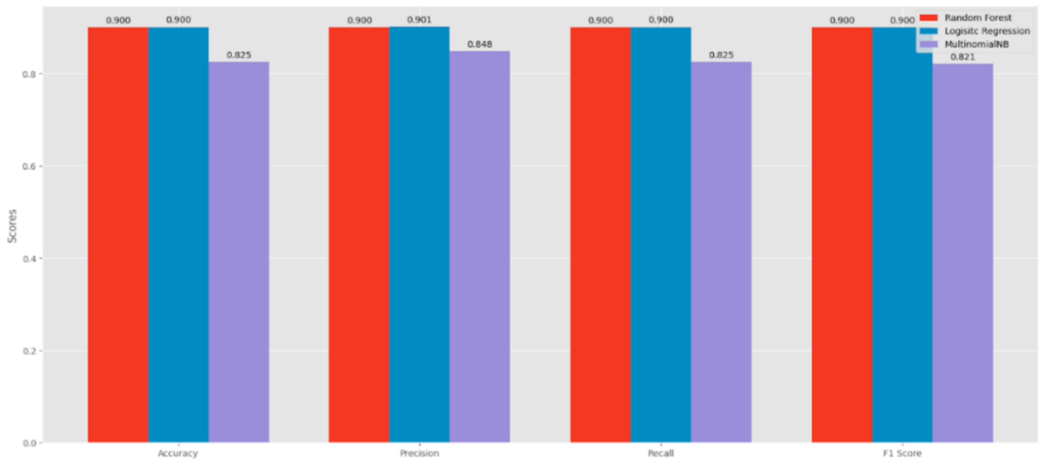


Fig. Comparison between Random Forest, Logistic Regression and Multinomial NaïveBayes

The above graph shows the comparison between Random Forest, Logistic Regression and Multinomial Naive Bayes on the basis of Accuracy, precision, recall and f1 score.

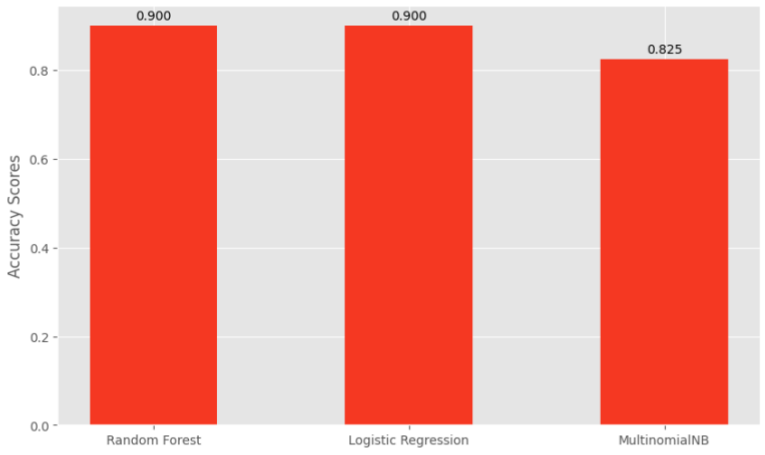


Fig. Accuracy Scores of models

On the basis of accuracy, Random Forest and logistic regression have the same accuracy of 90% while MultinomialNB has 82.5%.

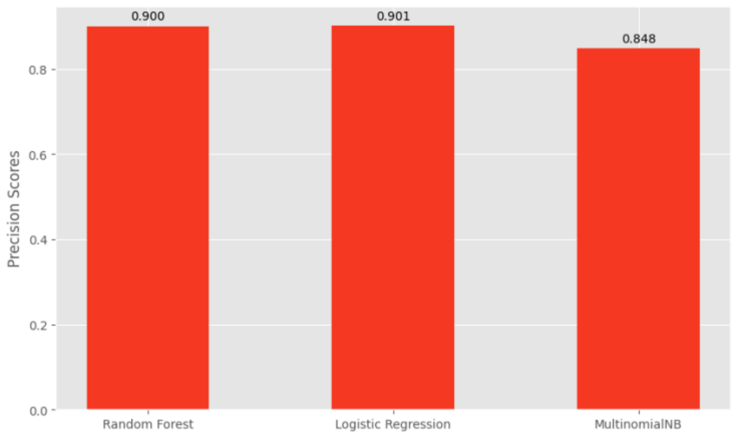


Fig. Precision Scores of models

On the basis of Precision, Logistic Regression has a score of 90.1%, slightly higher than Random Forest which has precision score 90.0%, while MultinomialNB has 84.8%.

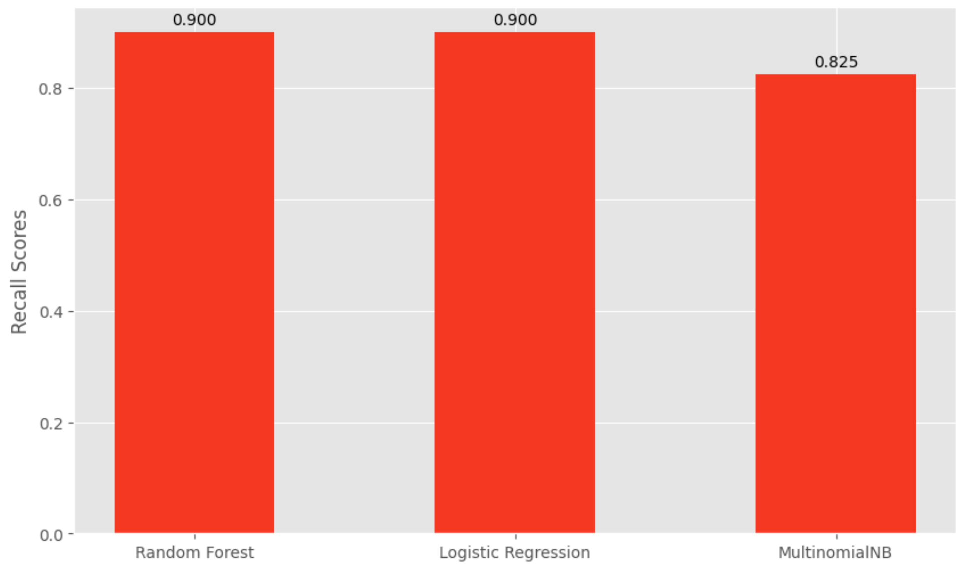


Fig. Recall Scores of models

On the basis of Recall, Random Forest and Logistic Regression both have a score of 90.0% while MultinomialNB has 82.5%

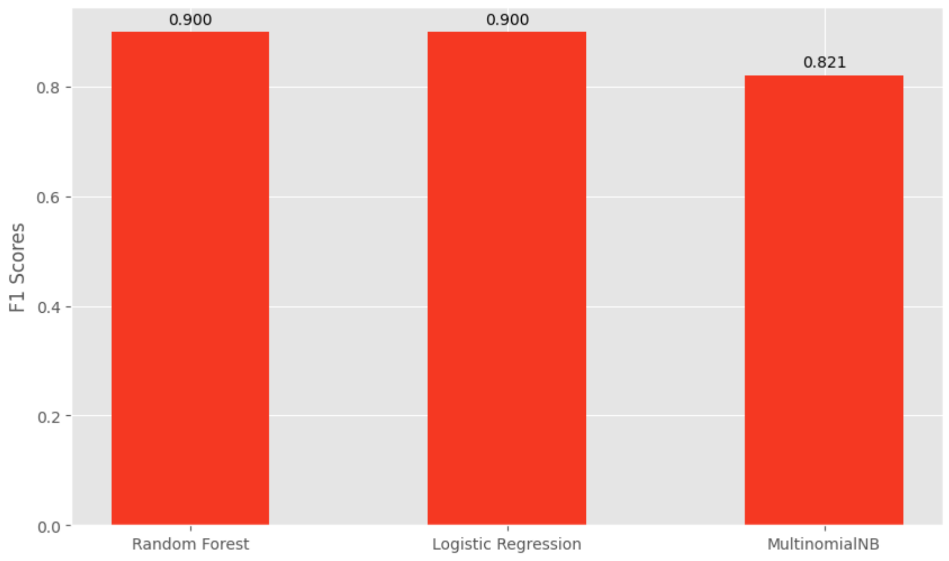


Fig. F1 Scores of models

On the basis of F1 Score, Random Forest and Logistic Regression both have a score of 90.0% while MultinomialNB has 82.1%

Training and Testing Scores:

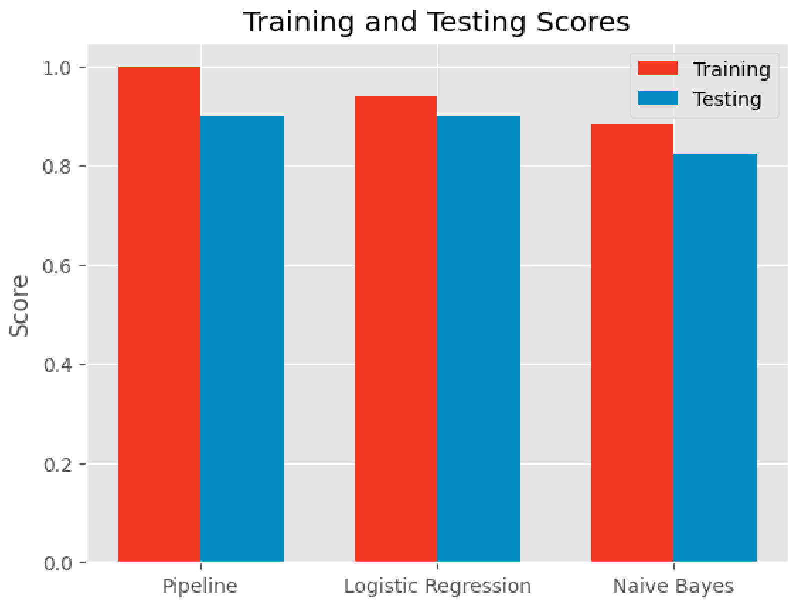
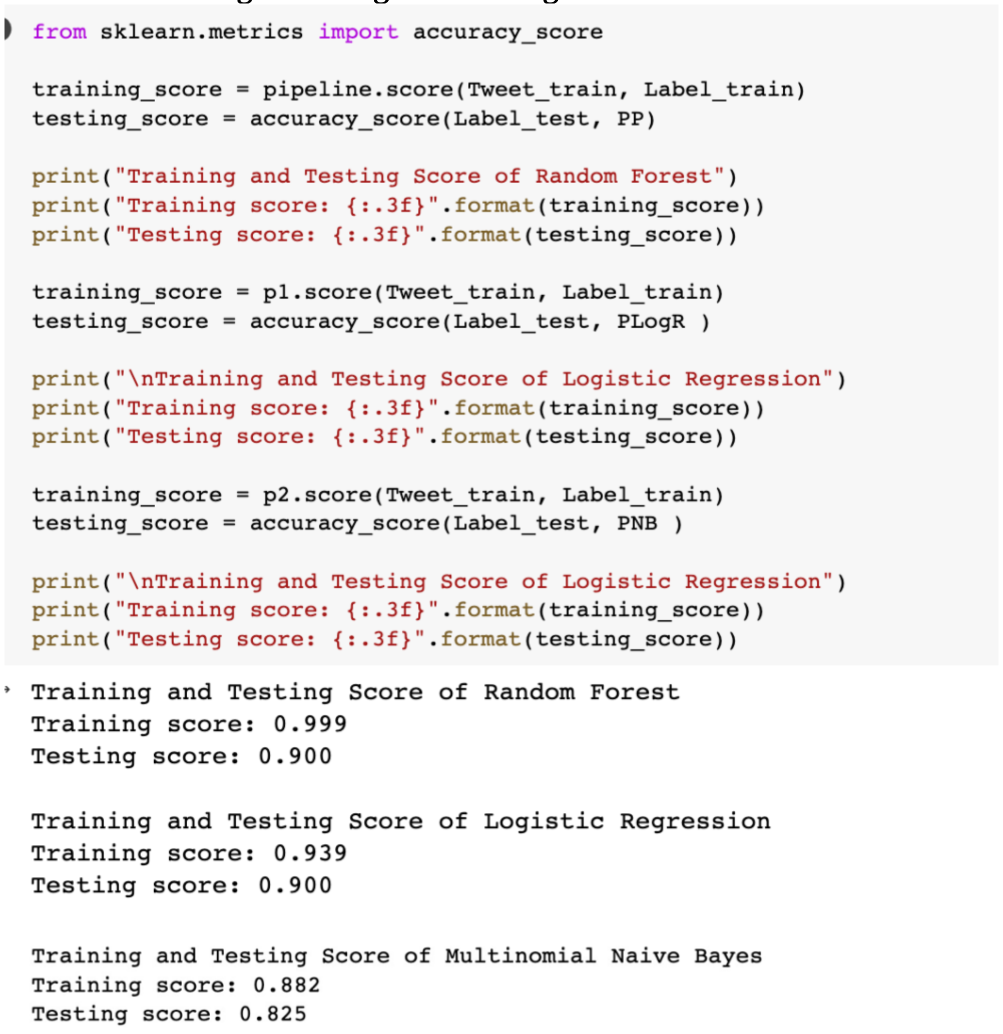


Fig. Training and Testing Scores of models



```
from sklearn.metrics import accuracy_score

training_score = pipeline.score(Tweet_train, Label_train)
testing_score = accuracy_score(Label_test, PP)

print("Training and Testing Score of Random Forest")
print("Training score: {:.3f}".format(training_score))
print("Testing score: {:.3f}".format(testing_score))

training_score = p1.score(Tweet_train, Label_train)
testing_score = accuracy_score(Label_test, PLogR )

print("\nTraining and Testing Score of Logistic Regression")
print("Training score: {:.3f}".format(training_score))
print("Testing score: {:.3f}".format(testing_score))

training_score = p2.score(Tweet_train, Label_train)
testing_score = accuracy_score(Label_test, PNB )

print("\nTraining and Testing Score of Logistic Regression")
print("Training score: {:.3f}".format(training_score))
print("Testing score: {:.3f}".format(testing_score))
```

```
Training and Testing Score of Random Forest
Training score: 0.999
Testing score: 0.900

Training and Testing Score of Logistic Regression
Training score: 0.939
Testing score: 0.900


Training and Testing Score of Multinomial Naive Bayes
Training score: 0.882
Testing score: 0.825
```

Fig. Training and Testing Scores of models

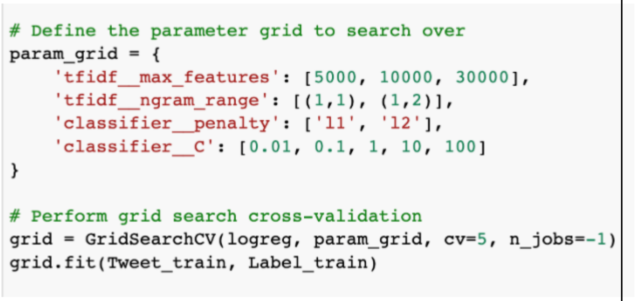
As we can have a look at, the Random Forest model appears to have a completely high training score, while the checking out rating is low, i.e. There may be a long hole between schooling and trying out scores, which means that the version is overfitting the records. On the other hand, the Logistic Regression and MultinomialNB fashions seem to have lower gaps and can be better at generalizing to new records.

Hyperparameter Tuning:

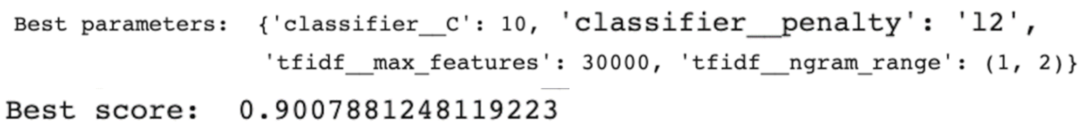
We decided to go with logistic regression, and performed hyperparameter tuning. The below code performs grid search to get the best parameters for the model.

The hyperparameters being searched over are as below:

- The max features set to be either 5000,10000 or 30,000
- The range of n-grams set to be either unigram Or bigram
- Type of regularization used with options L1 and L2
- Regularization strength to be either 0.01,0.1,1 or100.

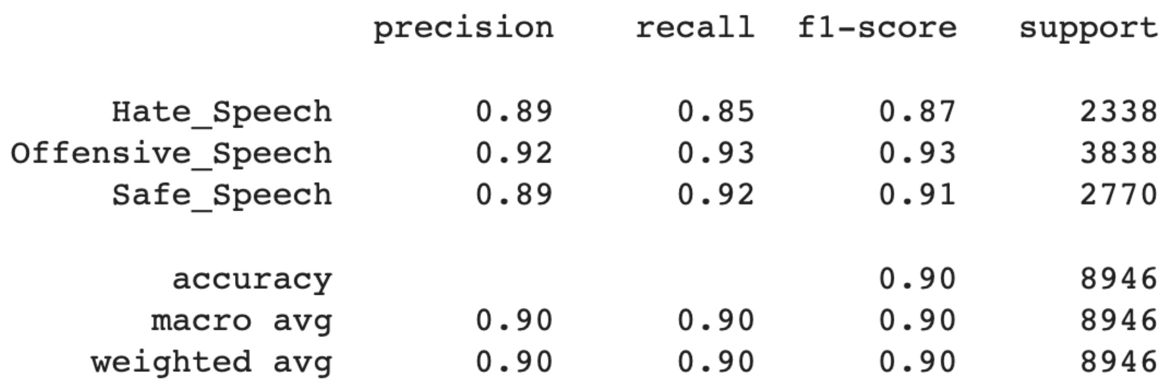


```
# Define the parameter grid to search over
param_grid = {
    'tfidf__max_features': [5000, 10000, 30000],
    'tfidf__ngram_range': [(1,1), (1,2)],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__C': [0.01, 0.1, 1, 10, 100]
}

# Perform grid search cross-validation
grid = GridSearchCV(logreg, param_grid, cv=5, n_jobs=-1)
grid.fit(Tweet_train, Label_train)
```



```
Best parameters:  {'classifier__C': 10, 'classifier__penalty': 'l2',
                   'tfidf__max_features': 30000, 'tfidf__ngram_range': (1, 2)}
Best score:  0.9007881248119223
```

The best parameters selected as shown above are: Regularization strength as 10, L2 as the type of regularization, max features as 30,000 and range of n-grams as bigram.

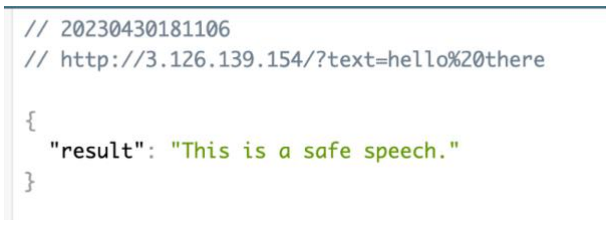


```
                 precision    recall  f1-score   support

    Hate_Speech       0.89      0.85      0.87      2338
Offensive_Speech       0.92      0.93      0.93      3838
    Safe_Speech       0.89      0.92      0.91      2770

       accuracy                           0.90      8946
      macro avg       0.90      0.90      0.90      8946
   weighted avg       0.90      0.90      0.90      8946
```

The precision, recall and f1 score of hate speech has increased by 1% after performing hyperparameter tuning, while the accuracy and the precision, recall and f1 score of the labels are the same.

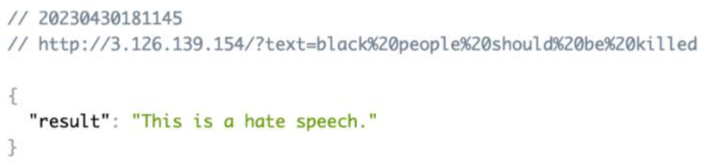
Deployment

The model was converted into an API using Flask, hosted on firebase and AWS, which can be utilized by the HateTron application for text classification.



```
// 20230430181106
// http://3.126.139.154/?text=hello%20there

{
  "result": "This is a safe speech."
}
```

Fig. Screenshot of API while inputting Safe Speech



```
// 20230430181145
// http://3.126.139.154/?text=black%20people%20should%20be%20killed

{
  "result": "This is a hate speech."
}
```

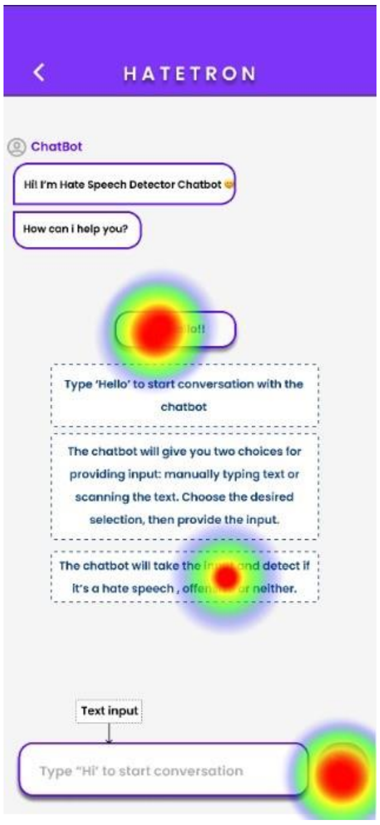
Fig. Screenshot of API while inputting Hate Speech

HCI Experiment Design

We used an iterative design process to create our experiment to test our app.

Iteration 1:
We used the Maze application in the first iteration to conduct a survey in which 12 users were assigned a task to complete and a report was generated based on their success rate. The survey had a 58% success rate. Participants reported difficulty using the interface during usability testing, and some icons were unclear to them. Participants had difficulty navigating the product because they were unsure what certain buttons did. We used both verbal feedback from testing participants and the System Usability Scale (SUS) survey to collect feedback. We analyzed the SUS survey data to identify areas that needed to be improved. We also recorded verbal feedback for later review. The link to the SUS result is Link for SUS Report.

Iteration 2:
We conducted usability testing with students and provided feedback via a UX Heatmap. The navigation difficulties encountered were the usability issue. We simplified the navigation based on feedback from the UX Heatmap to improve the overall usability of the prototype. Screenshots of the user testing and UX heatmap were used to document the findings and track the progress over time.
These are the screenshots of pages where users encountered difficulties, such as navigation.
User1:
We requested User1 to complete a task, and based on the data provided, we analyzed the heatmap and identified navigation issues on two specific pages.



Hi! I'm Hate Speech Detector Chatbot

How can i help you?

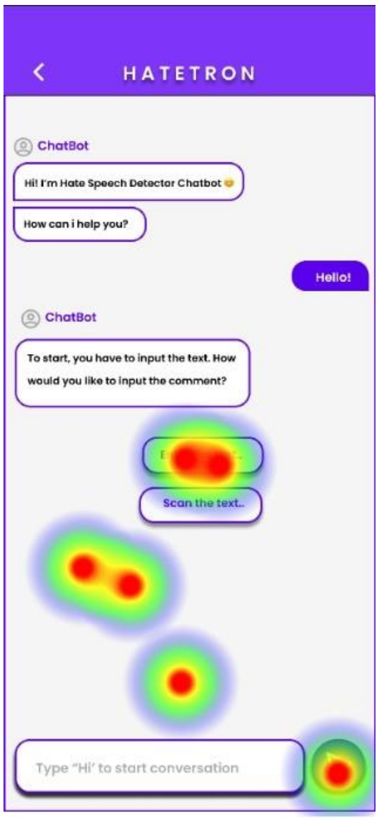
lo!!

Type 'Hello' to start conversation with the chatbot

The chatbot will give you two choices for providing input: manually typing text or scanning the text. Choose the desired selection, then provide the input.

The chatbot will take the in nd detect if it's a hate speech, offen r neither.

Type "Hi" to start conversation



Hi! I'm Hate Speech Detector Chatbot

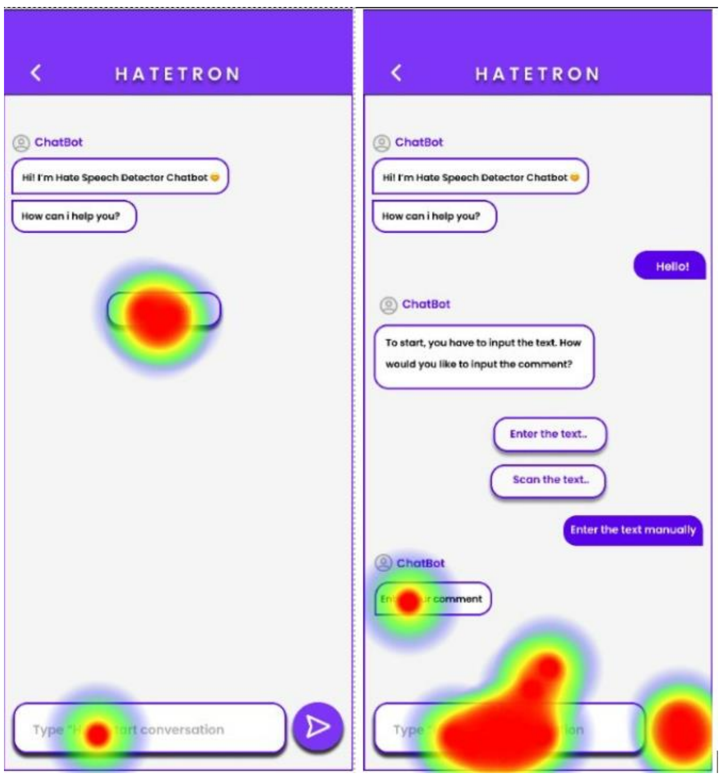
How can i help you?

Hello!

To start, you have to input the text. How would you like to input the comment?

Type "Hi" to start conversation

User2:



Hi! I'm Hate Speech Detector Chatbot 😊

How can i help you?

Hi! I'm Hate Speech Detector Chatbot 😊

How can i help you?

Hello!

To start, you have to input the text. How would you like to input the comment?

Enter the text..

Scan the text..

Enter the text manually

We found that some users were facing difficulty in navigating through our design based on user feedback. Some users were puzzled by the instructions, and others were unsure how certain buttons and features were supposed to work. New designs and interaction models were created to improve the user navigation.
The new design comes up with clear instructions, button layouts, and cues which are visually effective. Usability testing confirmed their effectiveness, demonstrating that users now can navigate the product more easily and quickly understand its functions.

Iteration 3:

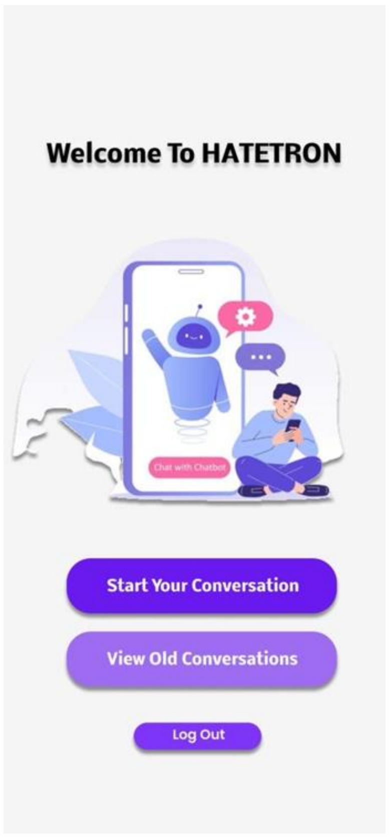
Every user expressed problems and frustration with the terminology mostly focussing on the interface. They felt that certain terms in the HATETRON bot were confusing, making it difficult to navigate and use. Furthermore, some users complained about the typography, which made it difficult to scan and read information quickly.

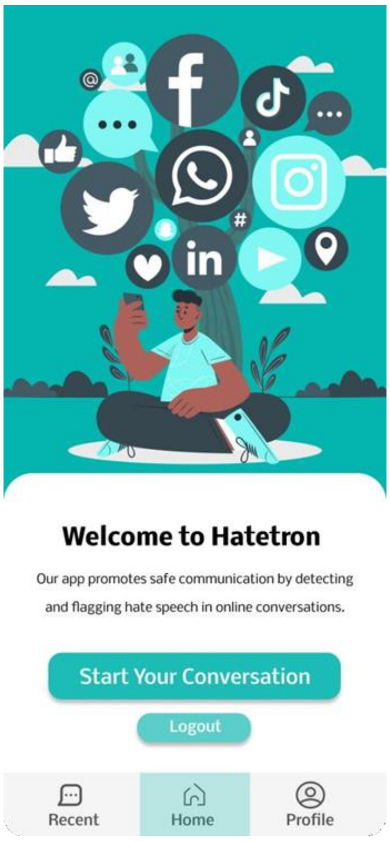
We made the terminology this time simple which was used throughout the interface and ensured that typography was consistent across all pages based on feedback from the focus group. We had also changed the font size and spacing to make the text more readable.

Iteration 4:

After reviewing the prototype, the common issue found through the survey was the confusing feedback in app. The app’s messaging and UI design needed to revise to give users a clear and intuitive feedback. This would improve User Experience and increase effectiveness while detecting Hate Speech. We addressed this issue by providing more contextual information. We included feedback from a HCI research, which indicated clear and concise feedback helps users to interact and understand a product.

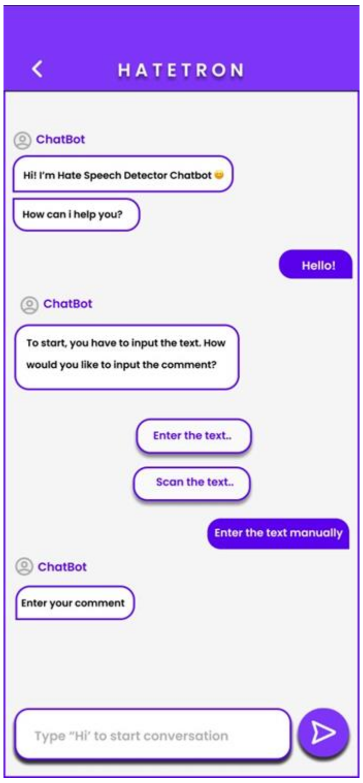
Iteration 5:
We asked for an expert review from two design professionals for the prototype. The prototype had poor colour contrast and design, which impacted the product’s usability. We improved the colour contrast and overall design for correcting this issue.





Welcome to Hatetron

Our app promotes safe communication by detecting and flagging hate speech in online conversations.



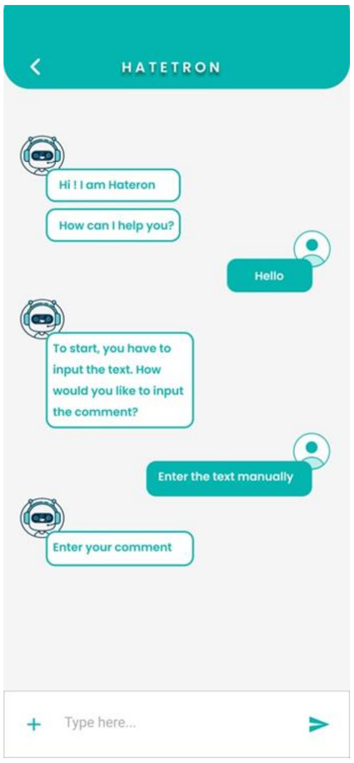
ChatBot
Hi! I’m Hate Speech Detector Chatbot 😊
How can i help you?

Hello!

ChatBot
To start, you have to input the text. How would you like to input the comment?

Enter the text manually

ChatBot
Enter your comment



Hi ! I am Hateron
How can I help you?

Hello

To start, you have to input the text. How would you like to input the comment?

Enter the text manually

Enter your comment

Additional feedback was taken after improving the prototype and the results came out to be positive.

Iteration 6:
We had taken feedback from users, where they mentioned that the prototype lacked some features for detecting and flagging hate or offensive speech. They mentioned that they were dissatisfied with the lack of key features, mainly when it comes to scanning the text for detecting the problem. (Kanade, 2022) This problem was discovered through observations of user behaviour and verbal feedback in a focus group. through the feedback we received from the users we added a button called “Scan the text”.

Iteration 7:
With the help of Maze app, we conducted a survey with **11 students,** and they were given tasks in the figma which was provided to them. Based on the survey results we found out that 73% of users completed the task successfully, and remaining faced some problems, which shows that the prototype had some usability issues.

Based on the SUS report, we found out that the length and complexity of the given tasks which caused issues and frustration for users. Later, we simplified the navigation, and we reduced the number of pages/steps required to complete the given task which can improve the user experience. We had done the above changes with the help of human computer interaction principles where we prioritized the ease of use and to minimize the cognitive load. This is the link to the SUS report: Link for SUS Report.

Iteration 8:
To gather feedback on the hate speech detector chatbot, we conducted a Google survey with 16 users from various backgrounds. We assigned tasks to users, and their responses generated a report based on how well they performed. We discovered that users had difficulty accessing and viewing previous records within the app. This problem was discovered because of user responses to survey questions.

Based on survey feedback and human-computer interaction principles, we improved the interface by adding a search function and a clear history button, as well as simplifying the language and providing more contextual information.
This is the link to the google response: Link to Google Forms.

MAD

Login Page:

- The code uses a 'StatefulWidget' called 'LoginPage' that has a 'State' called '_LoginPageState'. Two properties of the "LoginPage" widget are "key" and "showRegisterPage," a "VoidCallback" function that, when called, displays the registration page.
- A 'Scaffold' widget with a 'SafeArea' as its child is the result of the 'build' method of the '_LoginPageState' widget. The "SafeArea" widget has a "Centre" widget that is a "SingleChildScrollView" widget's child. This scroll view has a number of widgets as its children, including "Text," "Container," "Padding," and "TextField." (Flutter, 2023)
- The login page's title and subtitle are displayed in the 'Text' widgets. The 'TextField' widgets that request user input for password and email are enclosed by 'Container' widgets. The 'Container' widget, which has the 'Login' button that activates the'signIn' function when tapped, is encased in the 'GestureDetector' widget.

Register Page:

On the user registration page, personal details such as first and last names, age, email, password, and confirm password can be submitted by the user. When the register button is clicked, a new account with the given email and password is created via Firebase Authentication. The user's information is also stored in the Firestore database under the 'users' collection. The app's components consist of 'RegisterPage' and '_RegisterPageState'. '_RegisterPageState' is generated from the stateful widget 'RegisterPage', where the build method for the registration page's UI layout is included in '_RegisterPageState'. To manage the input field's values, 'TextEditingController' is employed within '_RegisterPageState'. When the user taps on the register button, the 'signUp()' function is invoked, generating a new user account if the password is confirmed. Finally, the 'addUserDetails()' method is utilised to include the user's data in the Firestore database.

Forgot Password:

Flutter and Firebase authentication's "Forgot Password" function for mobile applications. In the event that a client fails to remember their secret phrase, they can enter their email address and get a secret key reset connect through email. A form for changing the user's password is created by the 'ForgotPasswordPage' class, a 'StatefulWidget'.
The code includes the following features:

- ‘AppBar’: A standard Material AppBar widget that displays a colored background and no elevation.
- ‘Column’: A widget that arranges its children in a vertical column.
- ‘Padding’: A widget that adds padding to its child.
- ‘stateful’: A widget which displays password reset form.
- ‘Text’: A widget that displays text.
- ‘SizedBox’: A widget that creates an empty box with a specified height.
- ‘Container’: A widget that provides a rectangular visual element. It has a background color, a border, and border radius.
- ‘TextField’: A widget that allows the user to input text.
- ‘MaterialButton’: A widget that displays a clickable button with a specified color and text and calls the ‘passwordReset()’ function which sends a password reset email to the user’s email.
- ‘AlertDialog’ function which displays a message of forgot password status.

Home Page
The code uses a "homepage" StatefulWidget. It brings in three files:

- Access to the Firebase Cloud Firestore database is provided by the cloud_firestore.dart file.
- Firebase Authentication may be accessed through firebase_auth.dart.
- Flutter/Material.dart: Offers tools and widgets for creating the user interface.

The "user" variable in the "homepage" widget's state, which reflects the currently authorized user, is present. Additionally, it has a list called "docId" where document IDs obtained from the Firestore database will be kept.
The code creates a scaffold widget in the "build" method that has an AppBar and a body with a centred Column widget. The Column widget includes:
- A widget for images that shows an image.
- An email address text widget that shows the user's email address.
- Two MaterialButton widgets that let users sign out of the app or go to another page.

Chat area:
The code uses the DialogFlowtter package to create a chatbot in Flutter. MyApp is the main class that creates the MaterialApp and initializes an instance of Homee for the home screen, and Homee is the secondary class that initializes an instance of MyApp for the app's menu.
The home screen state is defined by the Homee class, a StatefulWidget class. A TextEditingController for user input, an instance of the DialogFlowtter class for handling chatbot functionality, and a list of messages to display on the screen are all included. The code additionally incorporates different UI parts like an AppBar and buttons for client association.
Additionally, the code makes use of an API that incorporates the ML model to obtain predictions and display warnings based on those predictions. The code uses the API to look at user messages and make predictions. Based on the API predictions and the number of messages in the messages list, the code uses conditional rendering to display specific UI elements. On the off chance that a message is anticipated to be hazardous or hurtful, an admonition message is shown to the client.

Recent Conversations:
The "RecentConversationsScreen" stateless widget can be found on the page. This gadget shows a rundown of ongoing discussions with different clients in the application. The screen incorporates an application bar with a custom title and a back button to explore to the landing page. Each conversation item includes the text input, date, and class it was classified into, and the list of conversations is displayed in a container that can be scrolled. A variety of fonts, colors, and sizes are used to style the conversation items. The widget takes the user to view the entire conversation history, when a conversation item is tapped.

Logout:
The page has a stateless widget referred to as “LogoutScreen”. This widget shows a affirmation conversation to the user once they pick out to log out of the app. The dialog consists of a message asking the user if they're certain they need to log off and two buttons, one to affirm the sign off action and one to cancel and live logged in. The verify button is coloured in green even as the cancel button is pink, to prevent confusion and errors. When the person taps the “Confirm” button, the widget navigates to the login screen, wherein the consumer can log in again with their credentials. The “LogoutScreen" widget is used to address logout moves during the app.

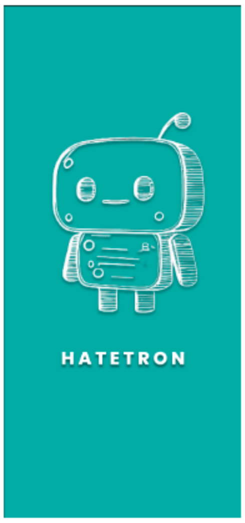
4. Results

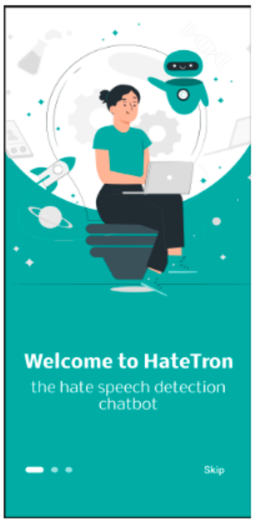
ML Results

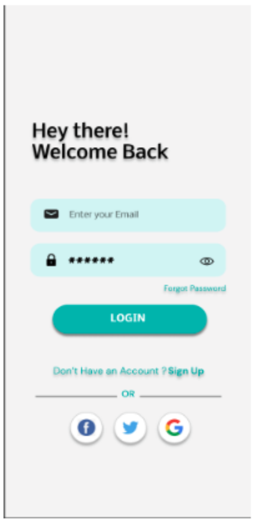
Model.	Accuracy.	Precision	Recall	F1 Score	Overfitting?
Random Forest Classifier	90.0%	90.0%	90.0%	90.0%	YES
Logistic Regression	90.0%	90.1%	90.0%	90.0%	NO
Multinomial Naive Bayes	82.5%	84.8%	82.5%	82.1%	NO

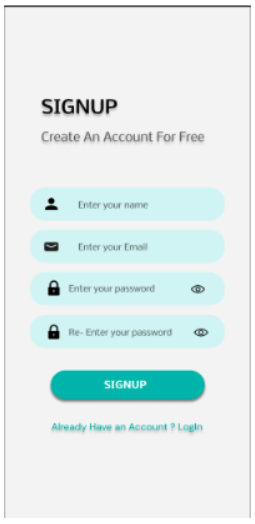
From the table we can conclude that Random Forest Classifier and Logistic Regression both have the highest Accuracy, precision, recall and f1 score. However, Random Forest Classifier is overfitting the data. Therefore, the Logistic Regression model has been selected to be deployed.

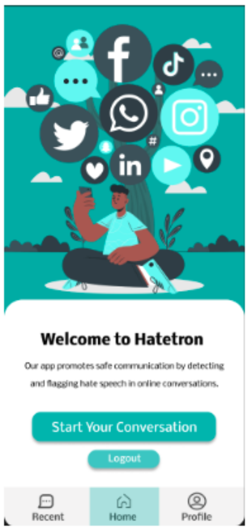
UI Screenshots

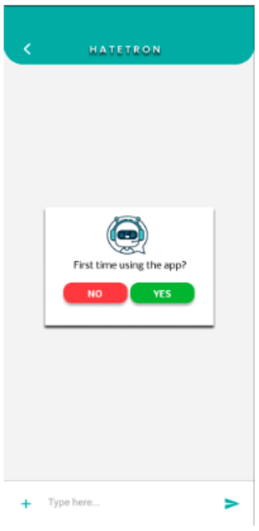


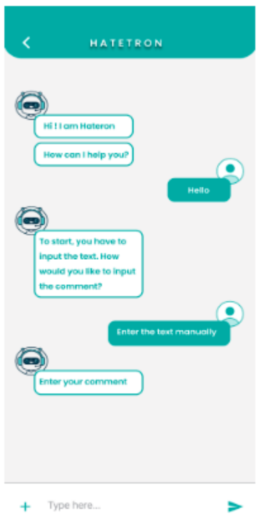


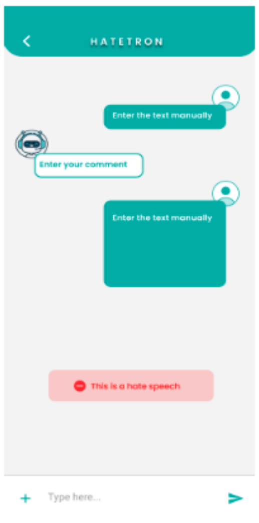


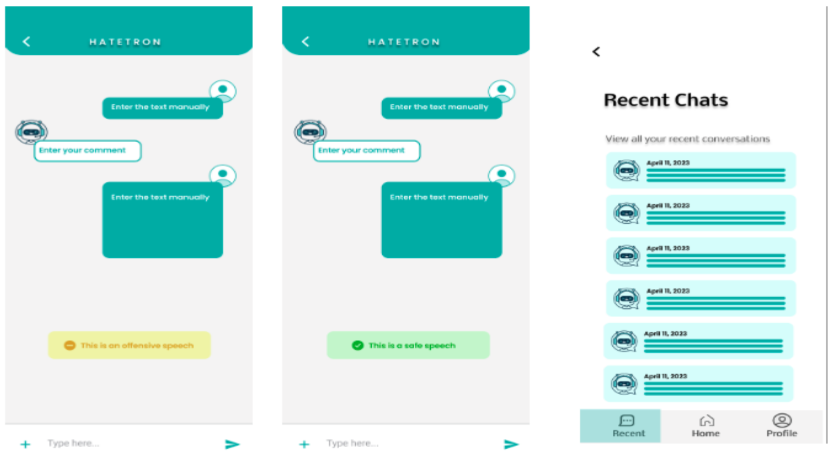



Welcome to Hatetron
Our app promotes safe communication by detecting and flagging hate speech in online conversations.
Start Your Conversation
Logout
Recent
Home
Profile







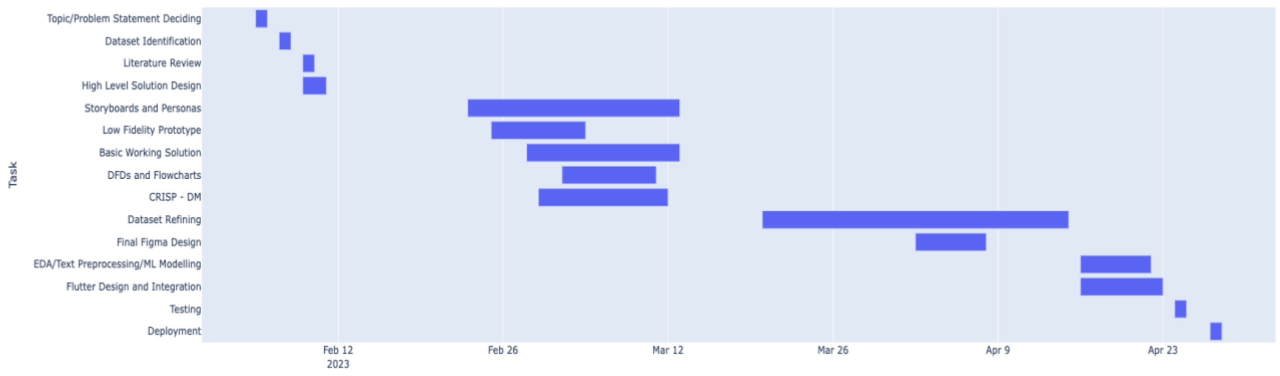


Usability evaluation results:

Iteration	Date/ Month	Duration	Number of Participants	Type of Participants	Method of Feedback	Type of usability issue encountered	Improvement from previous iteration	Link to artifacts collected
1	2nd March	1 week	12	Students	Usability Testing and SUS	Difficulty in using the interface and unclear icons	N/A - First iteration	Link for SUS Result
2	10th March	3 days	4	Students	UX Heatmap	Navigation difficulties	Simplified interface design, redesigned icons	Screenshots of user testing UX Heatmap
3	13th March	1 day	5	General Public	Focus Groups	Poor terminology and poor typography	Simplified navigation	N/A
4	15th March	2 days	10	Students	Survey	Confusing feedback	Simplified terminology with consistent typography	Link to Google Forms
5	18th March	1 day	2	Design professionals	Expert Review	Poor color contrast and design	More intuitive feedback	Annotated Screenshots of design changes
6	20th March	1 week	6	All users	Focus Groups	Lack of features to complete key tasks	Improved color contrast and design	Feedback mentioned below
7	28th March	1 day	11	Students	SUS Report	Bad user experience due to long steps for completing key tasks	Improved by adding scan the text option	Link for SUS Report
8	1st April	3 days	16	All	Survey	difficulty in accessing/viewing previous records.	Improved user experience	Link to Google Forms

5. Project Management

5.1 Gantt Chart



5.2 Through the Gantt chart, you need to explain how you made progress in the project over the course.

Our challenge arrangements started out on February five, 2023. We spent two days (February 5, 2023 - February 6, 2023) exploring and finalizing our subject matter/hassle statement. We wanted another two days (Feb 7, 2023 – Feb eight, 2023) to discover a dataset for our undertaking. In the following days (Feb nine, 2023 - Feb 10, 2023), we reviewed four papers and began working on excessive-degree answer design (Feb nine, 2023 - Feb 11, 2023). Then, in nearly 1.5 weeks (February 25, 2023 - March five, 2023), we created a low-constancy prototype on Figma. We also labored at the Basic Working Solution in almost 1.Five weeks (Feb 28, 2023 - Mar 13, 2023).
The dataset changed into delicate over the subsequent 3 weeks (March 20, 2023 - April 15, 2023). Meanwhile, we finished the very last Figma design in six days (April 2, 2023 - April 8, 2023). My team members have been taken 6 days (April 16, 2023 - April 22, 2023) for completing the ML code (EDA/Text Pre-processing/ML Modelling) and for the undertaking, as well as one week (April 16, 2023 - April 23, 2023) designing and integrating Flutter. Testing passed off (April 24, 2023 - April 25, 2023) and deployment befell (April 27, 2023 - April 28, 2023).

5.3 How did you split up project amongst yourself? How were the tasks distributed amongst yourself?

The group decided to work on the project in accordance with the evaluations, honing our skills as we went, attempting to perfect each component. For the initial evaluation, all members attempted to identify the problem statement and dataset, review various papers, and design a high-level solution. For the second evaluation, the group was divided into three subgroups of three members each. Each subgroup was assigned specific tasks to complete.

- Subgroup 1 had the task of completing DFDs and Flowcharts.
- Subgroup 2 had the task of completing Storyboards, Low Fidelity Prototype and researching various MAD and HCl techniques to implement in the project.
- Subgroup 3 had the task of researching various ML models that could be implemented in our project and completing CRISP - DM.
- All 9 nine members contributed equally for designing a basic working solution.

The institution turned into then divided once more, this time into elements, one with 3 participants and the opposite with six. The 3 members had been tasked with refining the dataset and finalizing the ML models for use in the code. The very last Figma layout became created with the aid of one of the three. The different six targeting Flutter Design, Integration, and Deployment.

5.4 What went well? each member to give self-assessment and peer evaluation, grade for each person's participation between 0-10 and give justification for each member's grade.

S.no	**Name**	**Registration No.**	**Avg. Rating each member received.**
1.	Anish Borkar	210C2030036	10
2.	Anoushka Srivastava	210C2030048	10
3.	Ankur Kaushal	210C2030218	10
4.	Dyuti Dasmahapatra	210C2030142	10
5.	Godavarthi Sai Nikhil	210C2030030	10
6.	Nichenametla Karthik Raja	210C2030113	10
7.	Vandamasu Sai Sumanth	210C2030060	10
8.	D Veera Harsha Vardhan Reddy	210C2030061	10
9.	Himanshu Sharma	210C2030207	10

6. Discussion Section

Key Learnings

We were able to learn a variety of techniques for developing and testing our project. On the technical side, we investigated various ML models, and learning about each helped us apply our knowledge to the project. We learned about concepts such as NLP, SMOTE, and others. We tested nearly four machine learning models and learned when to use each one. We learned how to find datasets based on problem statements, clean and pre-process them, and perform EDA.

We were able to create a functional chatbot using Flutter. We were able to learn about the various widgets and components that assisted us in building our project. We learned about Firebase and its database applications. We were able to save login/signup information as well as old conversation details using it. We were able to deploy it as an API, which we used to make predictions in the app.

We were capable of check our app using HCI assessment strategies such as Usability Testing, UX Heatmap, Survey, and so forth. We were able to companion Design Features together with Affordance, Consistency, Feedback, User-centred Design, Visibility, and Constraints with the numerous components of our app.

On the non-technical aspect, we have been capable of paintings well collectively. We understood teamwork, effective communication, and time management. We were a hit in motivating every member of our institution to finish his or her tasks correctly. Overall, we have been successful in know-how Project Management.

Limitations:
When text is passed through our model, our app/model is currently unable to correctly classify texts that contain short forms of profanities/expletives. It can only predict text written in English.

Challenges:
We acquired an imbalanced dataset wherein 19000 rows were of the “Offensive_Speech” elegance, 1013 rows had been of the “Hate_Speech” class, and 4000 rows had been of the “Safe_Speech” class. When we skilled and tested our model on these records, we were given the predicted accuracy, however the values of precision, do not forget, and f1 rating had been now not what we expected. For example, the values of precision, don't forget, and f1 score of “Hate_Speech” and “Safe_Speech” had been very low in assessment to the “Offensive_Speech” elegance.

Future Scope:
The project's future scope could include implementing the built model on a much larger dataset or real-time data using the Twitter API. The model's text processing function can be improved. To improve the results, new and better ML techniques can be used. The app could be expanded to detect Hate Speech in other languages like Hindi, Arabic, German, etc. It could also be developed so that the app makes suggestions to correct the text entered.

Abstract

This study introduces HateTron, a chatbot for detecting hate speech that achieves 90% accuracy by using logistic regression. The study's background is the growing demand for efficient automated systems to find/identify and report/address hate-speech on internet. The goal is to create a chatbot that interacts with users and can quickly identify Hate Speech. Several models have been trained on a tweets dataset that have been annotated as Hate Speech, Offensive Language, and Safe Speech as part of the methodology used in this study, and the model with the highest accuracy is chosen. After that, the classifier is included into a chatbot that can communicate with people and identify hate speech and other inappropriate language in their communications.

In order to evaluate the chatbot, a sample of tweets with offensive, safe, and hate speech was used. The evaluation's findings demonstrated that the chatbot could distinguish between safe and objectionable speech with 90% accuracy. In conclusion, the chatbot created in this work to detect hate speech is a potentially useful tool for preventing hate speech instances anywhere.

References

abhishekm. (2022, November 11). *geeksforgeeks*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/hate-speech-detection-using-deep-learning/

Farzindar Atefeh, W. K. (2013, September 04). *Wiley Online Library*. Retrieved from onlinelibrary.wiley.com: https://onlinelibrary.wiley.com/doi/10.1111/coin.12017

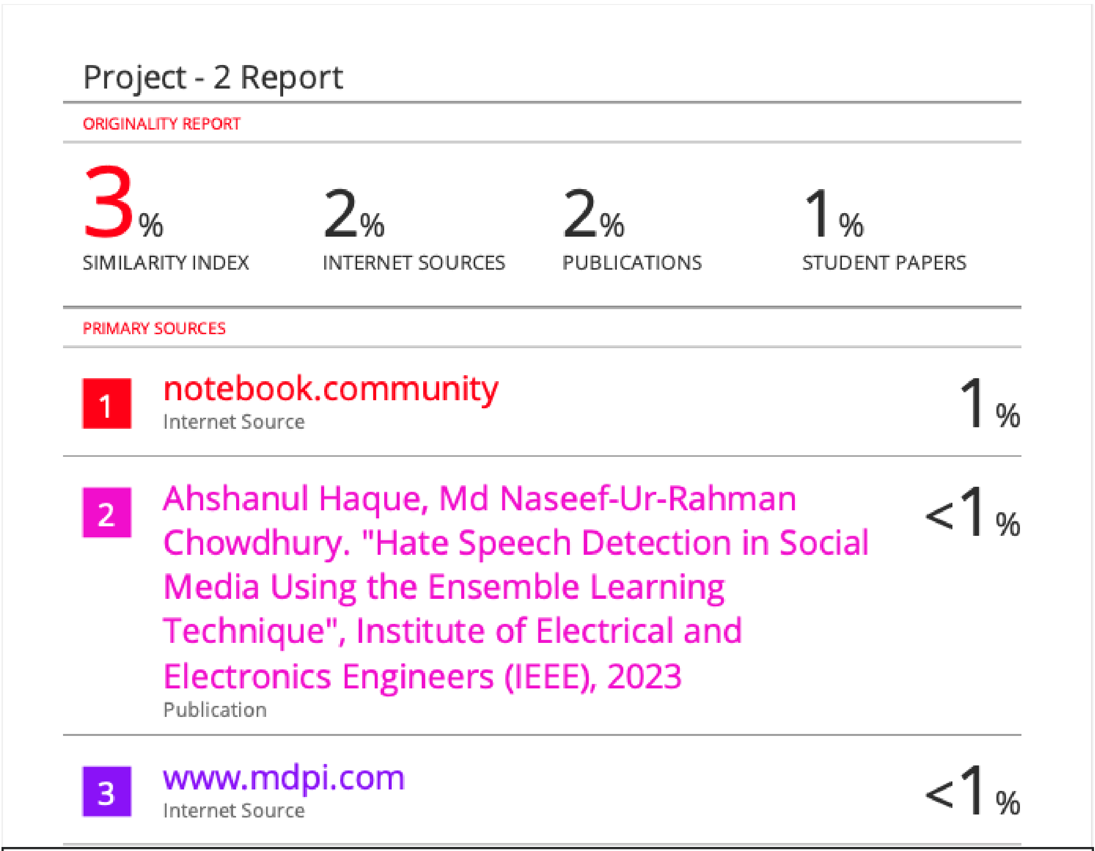
Flutter. (2023). *Flutter Documentation*. Retrieved from Flutter: https://docs.flutter.dev/ui/layout

Kanade, V. (2022, July 22). *spiceworks*. Retrieved from spiceworks: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-hci/

Kumar, A. (2022, March 20). *Data Analytics*. Retrieved from https://vitalflux.com: https://vitalflux.com/hate-speech-detection-using-machine-learning/#:~:text=Machine%20learning%20algorithms%20can%20be,disguised%20as%20jokes%20or%20sarcasm.

Norman. (2013). *The Design of Everyday Things.* Basic books.

Plagiarism Report



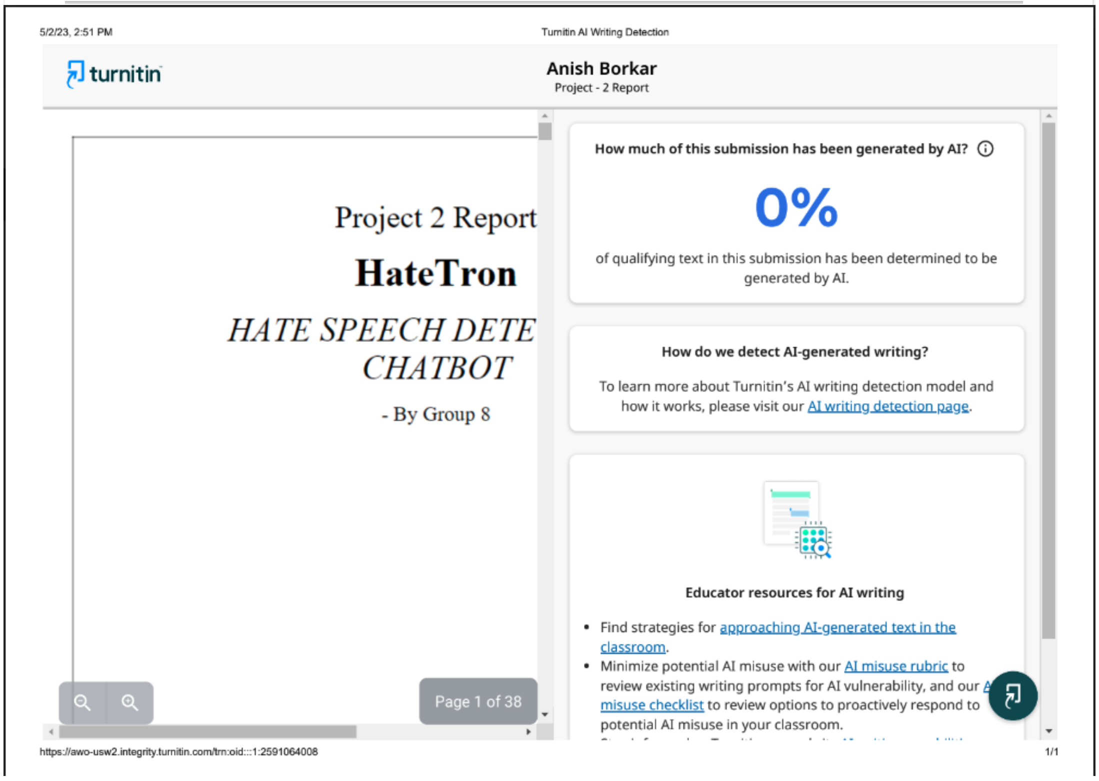
Project - 2 Report

ORIGINALITY REPORT

3%	2%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	notebook.community Internet Source	1%
2	Ahshanul Haque, Md Naseef-Ur-Rahman Chowdhury. "Hate Speech Detection in Social Media Using the Ensemble Learning Technique", Institute of Electrical and Electronics Engineers (IEEE), 2023 Publication	<1%
3	www.mdpi.com Internet Source	<1%



5/2/23, 2:51 PM Turnitin AI Writing Detection

turnitin

Anish Borkar
Project - 2 Report

Project 2 Report
HateTron
HATE SPEECH DETE
CHATBOT
- By Group 8

Page 1 of 38

How much of this submission has been generated by AI?

0%

of qualifying text in this submission has been determined to be generated by AI.

How do we detect AI-generated writing?

To learn more about Turnitin's AI writing detection model and how it works, please visit our AI writing detection page.

Educator resources for AI writing

- Find strategies for approaching AI-generated text in the classroom.
- Minimize potential AI misuse with our AI misuse rubric to review existing writing prompts for AI vulnerability, and our A misuse checklist to review options to proactively respond to potential AI misuse in your classroom.

https://awo-usw2.integrity.turnitin.com/trn:oid:::1:2591064008 1/1