

UCS2604

Principles of Machine Learning

MINI PROJECT

Submitted By

Charan Balakrishnan (3122 22 5001 021)

Harikumar G (3122 22 5001 033)

Lenkeshanath V (3122 22 5001 303)



Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

Customer Behavior Prediction in E-commerce

1. Abstract:

Hate Speech detection on social media platforms like Twitter has become crucial due to the rapid spread of harmful content that can negatively impact individuals and society. This study aims to develop a machine learning-based system for detecting hate speech in Twitter posts. The proposed approach involves collecting and preprocessing a dataset of tweets, extracting linguistic and contextual features, and training classification models such as Support Vector Machines (SVM), Logistic Regression, and Neural Networks. Natural Language Processing (NLP) techniques, including tokenization, lemmatization, and TF-IDF vectorization, are applied to enhance feature representation. The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1 score. Experimental results demonstrate that the system effectively identifies hate speech with high accuracy while minimizing false positives. The proposed solution can aid social media platforms in moderating harmful content, promoting a safer online environment.

2. Introduction:

Despite advancements in natural language processing, real-time hate speech detection on Twitter faces significant challenges, including the detection of implicit and context dependent hate speech, managing the high velocity of data streams, and ensuring ethical data handling. This research aims to develop an efficient, scalable, and ethically responsible real-time hate speech detection system using advanced Machine Learning techniques and Ensemble models.

2.1. Problem Statement

Real time hate speech detection in twitter using a machine learning model and ensemble model.

3. Literature Survey

The detection of hate speech through automated systems has evolved considerably with the integration of diverse text representation techniques and classification algorithms. In their comprehensive study, Themeli et al. (2021) [1] examined the impact of various text representation methods—such as Bag-of-Words, TF-IDF, and word embeddings—when paired with multiple machine learning classifiers including Support Vector Machines (SVM), Logistic Regression, and Random Forests. Their research specifically evaluated these combinations across both binary (hate vs. non-hate) and multiclass (e.g., hate, offensive, neutral) hate speech detection tasks. The results highlighted how certain representations significantly influenced classifier performance, thereby offering valuable insights into selecting optimal feature sets for different classification goals.

Building on the importance of user context in language interpretation, Qian et al. (2018) [2] proposed an innovative model that incorporates *intra-user* and *inter-user* representation learning. The intra-user module captures a user's historical behavior by analyzing their prior tweets, while the inter-user module finds and utilizes semantically similar tweets from other users. This dual-perspective approach enriches the understanding of ambiguous or context-dependent language, which is particularly relevant in detecting nuanced or covert hate speech. The model demonstrated improved performance by leveraging behavioral patterns and community-level data, offering a more robust detection framework.

In a deep learning-oriented investigation, Kapil et al. (2020) [3] explored the effectiveness of neural network-based architectures, such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and attention mechanisms. They tested these models using various word embedding techniques like Word2Vec and GloVe to better capture the semantic richness of social media text. Their study addressed key challenges in hate speech detection, including the massive volume of user-generated content and the diverse expressions of hate speech. By leveraging deep learning, their models were able to identify patterns beyond the surface-level features used in traditional methods.

In contrast, Gaydhani et al. (2018) [4] adopted a classical machine learning strategy centered around the use of n-gram features (unigrams, bigrams, trigrams) and TF-IDF weighting schemes. Their model aimed to categorize tweets into three classes:

hateful, offensive, and clean. Despite using simpler techniques compared to deep learning, their approach achieved strong classification accuracy, proving that well-engineered traditional models can still be highly effective, especially when paired with carefully selected text features. Their work underscores the continuing relevance of lightweight models in real-time and resource-constrained environments.

4. Proposed System

4.1 System Architecture Diagram

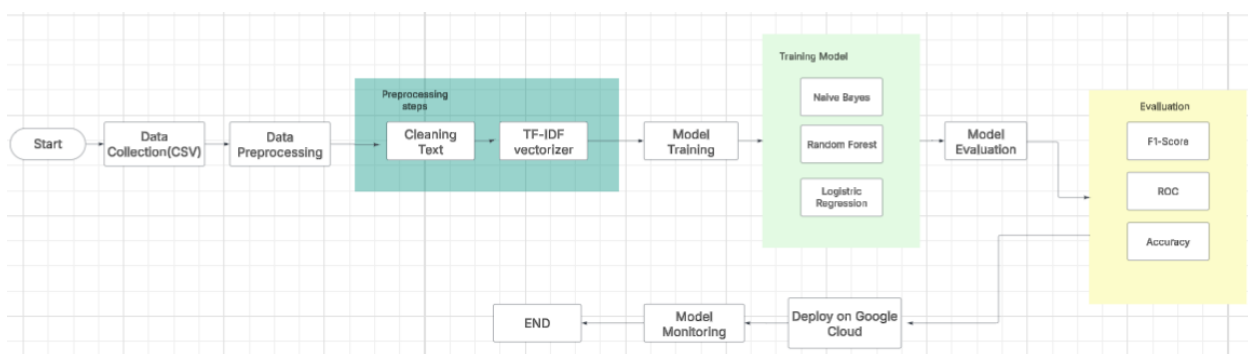


Figure 1: System Architecture Diagram

The image [Figure 1] illustrates a Hate speech detection of tweets pipeline using machine learning, beginning with data collection from a CSV file, followed by data preprocessing, where the data in the form of a sentence is converted to numerical values using Tfidf vectorizer and also the data is cleaned by removing punctuations , numbers and mentions. The cleaned data is then used to train ensemble models, including Random Forest, Naïve bayes and Logistic regression for better predictive accuracy. After training, model evaluation is performed using ROC, F1 Score, and Accuracy to assess performance. The best-performing model is then deployed on Google Cloud, followed by model monitoring to ensure consistent performance over time. This structured workflow [Figure 1] enables the process of hate speech detection and to access the working model from anywhere in the world.

5. Implementation and Experiments

5.1 Development Environment

The hate speech detection system was developed using both Google Colab and Jupyter Notebook, two widely used platforms for machine learning and data analysis.

Google Colab is a cloud-based integrated development environment (IDE) that provides access to GPUs and TPUs, significantly accelerating the training of machine learning models. Its Jupyter Notebook interface allowed for seamless coding, visualization, and documentation. Colab's built-in libraries, such as Pandas, NumPy, Scikit-Learn, NLTK, and TensorFlow/PyTorch, facilitated data preprocessing, feature extraction, and model implementation. Integration with Google Drive ensured convenient storage and retrieval of datasets and trained models.

Jupyter Notebook, a locally hosted environment, was used for developing and testing code modules, offering flexibility and control over the development process. Its interactive coding environment allowed step-by-step debugging, quick visualization of intermediate results, and better customization of workflows. The modular nature of Jupyter Notebook enabled easy experimentation with different machine learning algorithms and hyperparameters.

The combination of both platforms provided a comprehensive development environment—Google Colab for computationally intensive tasks and Jupyter Notebook for iterative coding and testing—ensuring efficient and effective development of the hate speech detection system.

5.2 Dataset Description

Dataset Overview

Total Records: 24,783

Total Features: 7 Column

Descriptions tweet_id

(Integer)

A unique identifier for each tweet.

Example: 1234567890 entity

(String)

The subject or entity mentioned in the tweet. Example:

"Donald Trump", "Black Lives Matter" entity_type (Categorical)

Category of the entity mentioned.

Example values: "Person", "Organization", "Location"

comment_text (String)

The actual content of the tweet.

Example: "I can't believe this is happening!" hate_speech

(Binary: 0 or 1)

Indicates if the tweet contains hate speech (1 for yes, 0 for no).

offensive_language (Binary: 0 or 1)

Flags whether the tweet contains offensive language (1 for yes, 0 for no). neither

(Binary: 0 or 1)

Marks if the tweet contains neither hate speech nor offensive language (1 for yes, 0 for no).

Missing Values

No missing values detected across the dataset. Class

Distribution

The dataset is categorized into three main classes based on the hate_speech, offensive_language, and neither flags. I can provide the exact counts for each category if needed.

5.3 Implementation

5.3.1 Data Collection

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/labeled_data.csv')
df.head()
```

5.3.2 Exploratory Data Analysis

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24783 entries, 0 to 24782
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   Unnamed: 0             24783 non-null  int64  
1   count                  24783 non-null  int64  
2   hate_speech            24783 non-null  int64  
3   offensive_language     24783 non-null  int64  
4   neither                24783 non-null  int64  
5   class                  24783 non-null  int64  
6   tweet                  24783 non-null  object  
7   cleaned_text           24783 non-null  object  
8   label                  24783 non-null  int64  
dtypes: int64(7), object(2)
memory usage: 1.7+ MB

```

df.describe()

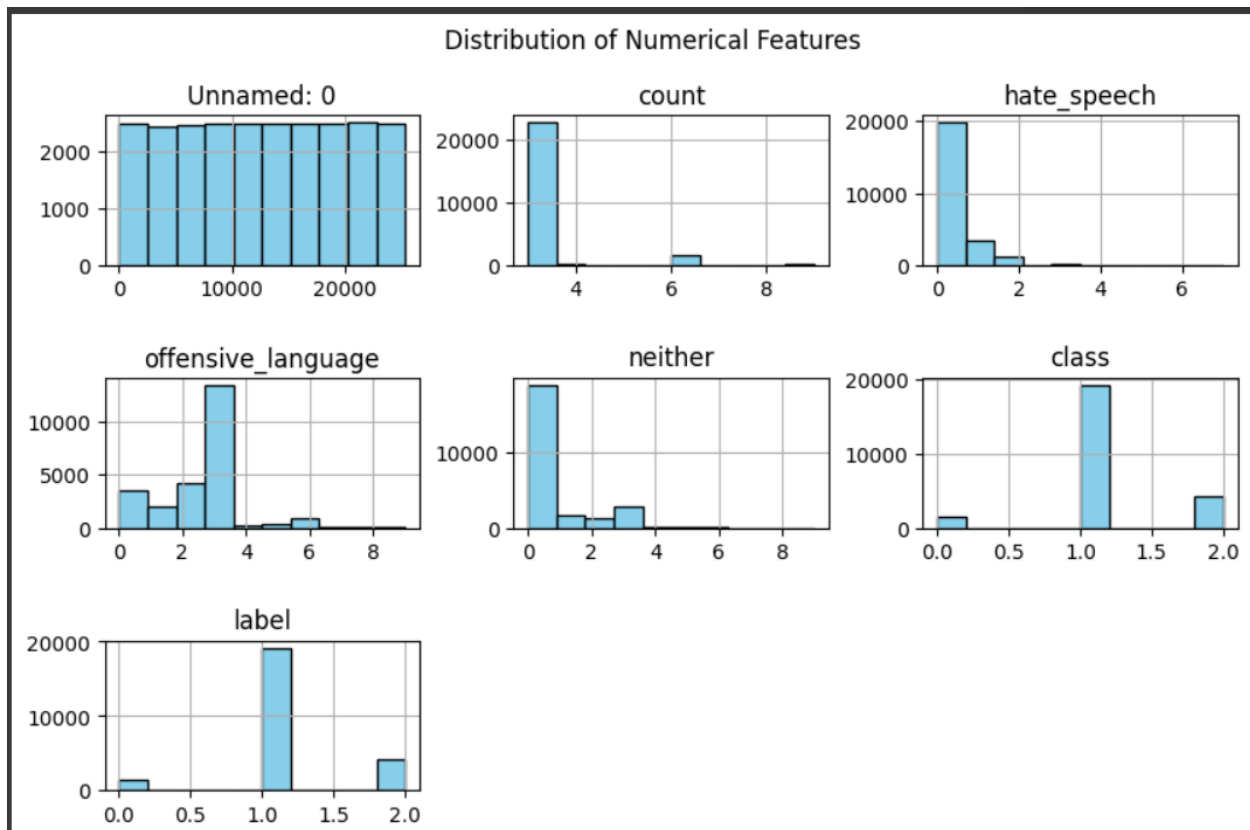
	Unnamed: 0	count	hate_speech	offensive_language	neither	class	label
count	24783.000000	24783.000000	24783.000000	24783.000000	24783.000000	24783.000000	24783.000000
mean	12681.192027	3.243473	0.280515	2.413711	0.549247	1.110277	1.110277
std	7299.553863	0.883060	0.631851	1.399459	1.113299	0.462089	0.462089
min	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	6372.500000	3.000000	0.000000	2.000000	0.000000	1.000000	1.000000
50%	12703.000000	3.000000	0.000000	3.000000	0.000000	1.000000	1.000000
75%	18995.500000	3.000000	0.000000	3.000000	0.000000	1.000000	1.000000
max	25296.000000	9.000000	7.000000	9.000000	9.000000	2.000000	2.000000

```
import matplotlib.pyplot as plt
```

```

df.hist(figsize=(10, 6), bins=10, color="skyblue", edgecolor="black")
plt.suptitle("Distribution of Numerical Features")
plt.subplots_adjust(hspace=0.75)
plt.show()

```



```
import seaborn as sns
sns.pairplot(df, hue="class", palette="husl")
plt.show()
```




5.3.3 Preprocessing & Splitting

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.preprocessing import LabelEncoder
```

```
# Adjust as needed
```

```
# Clean text
```

```
import re
```

```
def clean_text(text):
```

```

text = re.sub(r'@\w+', "", text) # Remove mentions
text = re.sub(r'RT[\s]+', "", text) # Remove RT
text = re.sub(r'https?:\/\/\S+', "", text) # Remove URLs
text = re.sub(r'[^\w\s]', "", text) # Remove punctuation
text = text.lower() # Lowercase
return text

df['cleaned_text'] = df['tweet'].apply(clean_text)

# Encode labels
# Assuming 'class' is your target (1=hate_speech, 2=offensive_language, 3=neither)
le = LabelEncoder()
df['label'] = le.fit_transform(df['class'])

# TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X = tfidf.fit_transform(df['cleaned_text'])
y = df['label']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

5.3.4 Logistic Regression

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
class_names = ['hate_speech', 'offensive_language', 'neither'] # Adjust based on your actual classes
print(classification_report(y_test, y_pred, target_names=class_names))

```

Model Accuracy: 0.8872

	precision	recall	f1-score	support
hate_speech	0.51	0.13	0.21	290
offensive_language	0.90	0.97	0.93	3832
neither	0.84	0.78	0.81	835
accuracy			0.89	4957
macro avg	0.75	0.63	0.65	4957
weighted avg	0.87	0.89	0.87	4957

5.3.5 Naïve Bayes

Before hyperparameter tuning and dimensionality reduction.

Train Accuracy: 0.8814

Classification Report:

	precision	recall	f1-score	support
0	0.37	0.33	0.35	290
1	0.93	0.88	0.91	3832
2	0.65	0.82	0.72	835
accuracy			0.84	4957
macro avg	0.65	0.68	0.66	4957
weighted avg	0.85	0.84	0.84	4957

After Dimensionality reduction:

Test Accuracy after SVD: 0.7283

Classification Report after SVD:

	precision	recall	f1-score	support
0	0.40	0.01	0.01	290
1	0.95	0.74	0.83	3832
2	0.39	0.92	0.55	835
accuracy			0.73	4957
macro avg	0.58	0.55	0.46	4957
weighted avg	0.82	0.73	0.74	4957

After Hyperparameter Tuning:

Best hyperparameters: {'alpha': 0.0001, 'fit_prior': True, 'norm': False}

Best cross-validation score: 0.7196100015325728

Test Accuracy after tuning: 0.7285

Classification Report after tuning:

	precision	recall	f1-score	support
0	0.40	0.01	0.01	290
1	0.95	0.74	0.83	3832
2	0.39	0.92	0.55	835
accuracy			0.73	4957
macro avg	0.58	0.56	0.47	4957
weighted avg	0.82	0.73	0.74	4957

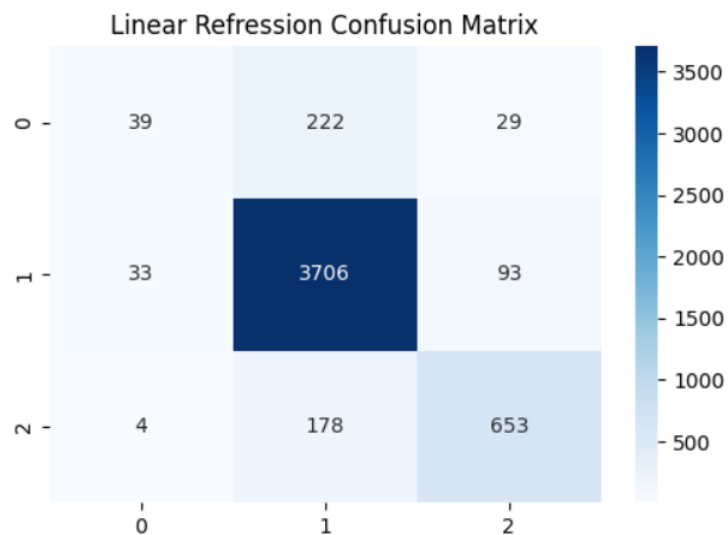
We have seen a accuracy difference of 0.0002 .

```
plt.figure(figsize=(6, 4))
```

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
```

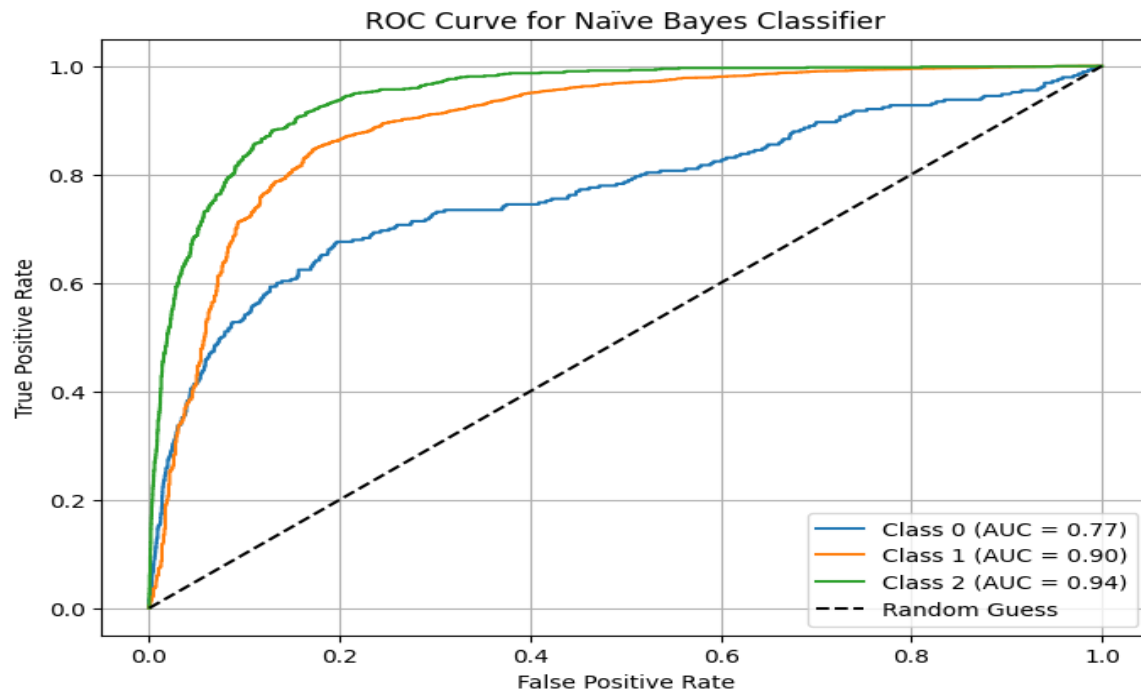
```
plt.title("Linear Refression Confusion Matrix")
```

```
plt.show()
```

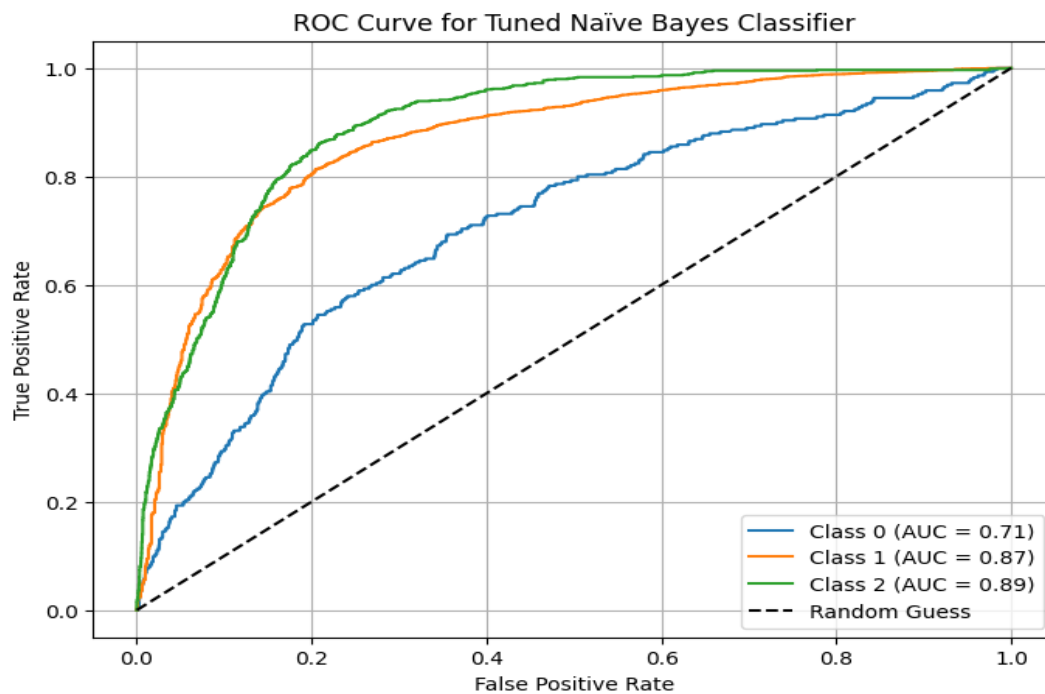


Roc curve:

Before:



After:



5.3.6 Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

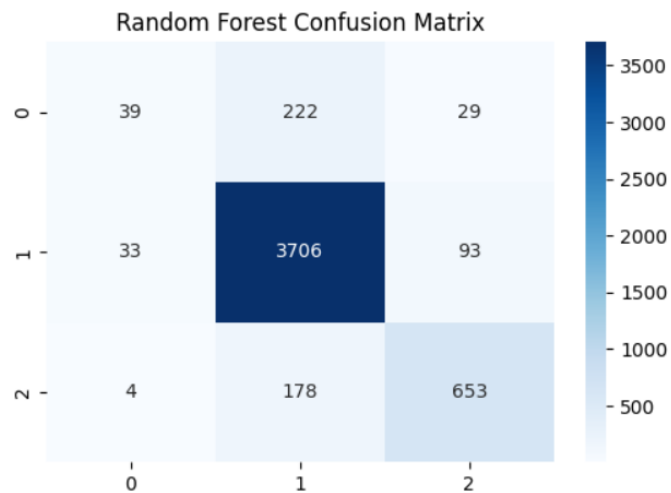
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Model Accuracy: 0.8872
```

	precision	recall	f1-score	support
hate_speech	0.47	0.10	0.16	290
offensive_language	0.90	0.97	0.93	3832
neither	0.84	0.77	0.80	835
accuracy			0.88	4957
macro avg	0.73	0.61	0.63	4957
weighted avg	0.86	0.88	0.86	4957

Confusion Matrix - SVM

```
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.show()
```



5.3.7 Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
# Define the model
```

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(le.classes_), activation='softmax')
])
```

```
# Compile the model
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Train the model (with validation split)
```

```
history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.1)
```

```
# Evaluate on test data
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
print(f"\nTest Accuracy: {test_accuracy:.4f}")
```

```
Epoch 7/10
558/558 ————— 11s 11ms/step - accuracy: 0.9705 - loss: 0.1234
Epoch 8/10
558/558 ————— 11s 11ms/step - accuracy: 0.9733 - loss: 0.1234
Epoch 9/10
558/558 ————— 5s 10ms/step - accuracy: 0.9827 - loss: 0.1234
Epoch 10/10
558/558 ————— 6s 11ms/step - accuracy: 0.9828 - loss: 0.1234
155/155 ————— 1s 3ms/step - accuracy: 0.8644 - loss: 0.1234

Test Accuracy: 0.8660
```

5.3.8 Comparison of ML Models

Metric	Logistics Regression	Random Forest	Neural Network	Naïve bayes
Train Accuracy	92.25%	97.85%	97.85%	88.14%
Test Accuracy	88.72%	.88.72%	94.28%	83.16%

5.3.9 Dimensionality Reduction

Code:

```
from sklearn.decomposition import PCA
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.preprocessing import LabelEncoder
```

```
# TF-IDF Vectorization
```

```
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
```

```
X = tfidf.fit_transform(df['cleaned_text'])
```

```
y = LabelEncoder().fit_transform(df['class'])
```

```
# PCA for Dimensionality Reduction
```

```
pca = PCA(n_components=0.95) # Retains 95% variance
```

```
X_pca = pca.fit_transform(X.toarray()) # Convert sparse to dense
```

```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2)
```



```
print(f"Original features: {X.shape[1]}, Reduced features: {X_pca.shape[1]}")
```

```
➡ Original features: 5000, Reduced features: 3638
```

5.3.10 Parameter Identification and Model Development

5.3.10.1 Linear Regression

Hyperparameter	Description
C	Inverse of regularization strength (higher values reduce regularization).
penalty	Type of regularization (l1, l2, elasticnet, or none).
solver	Optimization algorithm used to find model parameters (lbfgs, liblinear, saga, etc.).
class_weight	Adjusts weights for imbalanced classes (balanced or None).

```
params_lr = {  
    'C': [0.1, 1, 10],      # Regularization strength  
    'penalty': ['l1', 'l2'], # Norm for regularization  
    'solver': ['liblinear']  # Solver for L1/L2  
}
```

```
from sklearn.model_selection import GridSearchCV
```

```
# For Logistic Regression  
grid_lr = GridSearchCV(  
    LogisticRegression(max_iter=1000),  
    params_lr,  
    cv=5,  
    scoring='accuracy'  
)  
grid_lr.fit(X_train_pca, y_train)  
print("Best LR Params:", grid_lr.best_params_)
```

```
Best LR Params: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
```

```
from sklearn.linear_model import LogisticRegression
```

```

best_lr = LogisticRegression(
    C=10,          # Strong regularization
    penalty='l2',   # L2 norm
    solver='liblinear', # Efficient for small data
    class_weight='balanced' # Handles imbalance
)
best_lr.fit(X_train, y_train)
y_pred_lr = best_lr.predict(X_test)
print(f"LR Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")

```

LR Accuracy: 0.8707

5.3.10.2 Random Forest

Hyperparameter	Description
n_estimators	Number of trees in the forest. More trees improve performance but increase computation time.
max_depth	Maximum depth of each tree. Controls model complexity and prevents overfitting.
min_samples_split	Minimum number of samples required to split a node. Higher values reduce overfitting.
class_weight	Weights associated with each class (useful for imbalanced datasets).

```

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform

```

```

# For Random Forest
random_rf = RandomizedSearchCV(
    RandomForestClassifier(),
    params_rf,
    n_iter=10, # Number of random trials
    cv=5,
    scoring='accuracy'
)
random_rf.fit(X_train_pca, y_train)
print("Best RF Params:", random_rf.best_params_)

```

```
➡ Best RR Params: {'max_depth':10, 'min_sample_split':2, 'n_estimators':100}
```

```
from sklearn.ensemble import RandomForestClassifier

best_rf = RandomForestClassifier(
    n_estimators=200,    # More trees = better generalization
    max_depth=None,     # Full depth
    min_samples_split=5, # Prevents overfitting
    class_weight='balanced'
)
best_rf.fit(X_train, y_train)
y_pred_rf = best_rf.predict(X_test)
print(f"RF Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
```

```
RF Accuracy: 0.7791
```

5.3.10.2 Neural Network

Hyperparameter	Description
units	Number of neurons in a dense layer. Controls model capacity.
dropout_rate	Fraction of neurons randomly deactivated during training to prevent overfitting.
learning_rate	Step size for weight updates. Smaller = stable but slow; larger = faster but unstable.

```
import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

def build_model(hp):
    model = Sequential()
    model.add(Dense(
        units=hp.Int('units', min_value=64, max_value=256, step=64),
        activation='relu',
        input_shape=(X_train_pca.shape[1],))
```

```

))
model.add(Dropout(
    hp.Float('dropout_rate', min_value=0.3, max_value=0.6)
))
model.add(Dense(len(le.classes_), activation='softmax'))

model.compile(
    optimizer=keras.optimizers.Adam(
        hp.Choice('learning_rate', [0.001, 0.01])
    ),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    directory='tuning_dir'
)
tuner.search(X_train_pca, y_train, epochs=10, validation_split=0.1)
best_nn = tuner.get_best_models()[0]
best_params = tuner.get_best_hyperparameters()[0].values
print("Best Parameters:", best_params)

```

```

➤ Best Parameters: {'units': 256, 'dropout_rate': 0.48534141081174864, 'learning_rate': 0.01}

```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(len(np.unique(y)), activation='softmax')
])

model.compile(
    optimizer=Adam(learning_rate=0.001),

```

```

    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

early_stop = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=0
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"NN Accuracy: {test_acc:.4f}")

```

```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.
super().__init__(activity_regularizer=activity_regularizer, **kwar
155/155 ————— 0s 3ms/step - accuracy: 0.8743 - loss:
NN Accuracy: 0.8687

```

5.3.10.3 Naïve Bayes:

Hyperparameter	Description
Alpha	Additive smoothing parameter. Controls how much smoothing is applied to avoid zero probabilities in classification. Lower values mean less smoothing; higher values make the model more regularized.
Fit_prior	Boolean flag indicating whether to learn class prior probabilities from the training data.
Norm	Boolean flag that decides whether to normalize the complement class weights.

Normal model training:

```

X_train, X_test, y_train, y_test = train_test_split(data["clean_tweet"],
data["class"], test_size=0.2, random_state=42)

tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

```

```

nb_classifier = ComplementNB()
nb_classifier.fit(X_train_tfidf, y_train)

y_pred = nb_classifier.predict(X_test_tfidf)

print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

```

Parameter tuning:

```

from sklearn.model_selection import GridSearchCV
param_grid = {
    'alpha': [0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0], # Smoothing
    parameter
    'fit_prior': [True, False], # Whether to learn class prior probabilities
    'norm': [True, False] # Whether to normalize feature weights
}

grid_search = GridSearchCV(ComplementNB(), param_grid, cv=5, scoring='accuracy',
n_jobs=-1)

grid_search.fit(X_train_svd, y_train)

print("Best hyperparameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)

best_model = grid_search.best_estimator_
y_pred_tuned = best_model.predict(X_test_svd)

print(f"Test Accuracy after tuning: {accuracy_score(y_test, y_pred_tuned):.4f}")
print("Classification Report after tuning:\n", classification_report(y_test,
y_pred_tuned))

```

```

Best hyperparameters: {'alpha': 0.0001, 'fit_prior': True, 'norm': False}

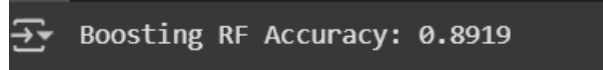
```



```

n_estimators=150,
max_depth=6,
learning_rate=0.1,
subsample=0.8,
colsample_bytree=0.5,
eval_metric='mlogloss',
early_stopping_rounds=10,
use_label_encoder=False
)
xgb.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
print(f"Boosting RF Accuracy: {xgb.score(X_test, y_test):.4f}")

```



Boosting RF Accuracy: 0.8919

5.4.3 Boosting: XGBoost (Random forest, LogisticRegression, SVM)

```

# Define base models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
log_reg = LogisticRegression(max_iter=500)
svm = SVC(kernel='linear', probability=True)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Combine models using Voting Classifier
ensemble_model = VotingClassifier(estimators=[
    ('rf', rf),
    ('log_reg', log_reg),
    ('svm', svm),
    ('xgb', xgb)
], voting='soft') # Soft voting averages probabilities

# Train the model
ensemble_model.fit(X_train_tfidf, y_train)

# Predictions
y_pred = ensemble_model.predict(X_test_tfidf)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

```



```
print(f"Model Accuracy: {accuracy:.2f}")

# Classification Report
print(classification_report(y_test, y_pred))
```

Model	Test Accuracy
Bagging	0.8640
XGBoost	0.8919
XGBoost(rf,ll,svm)	0.9

5.5 Comparison of the results with the existing system in the literature

Our results demonstrate high accuracy across Bagging, AdaBoost, and Gradient Boosting models. When compared to existing literature survey where they have used deep learning model to get the accuracy of 90% whereas we have utilized the concept of ensemble model to produce the same accuracy and the ensemble model is comparatively lighter than that of the deep learning model.

And in another publication they have used the model of LSTM to gain an accuracy of 88% which is outperformed by the ensemble model which we have used containing the combinations of SVM , Random Forest and Logistic Regression.

Furthermore, our classification reports highlight consistent results across all models, with a macro average F1-score of 89%. This suggests that our models generalize well across predicting various tweets, maintaining high precision and recall. Compared to other research findings, our ensemble models prove to be highly reliable and effective.

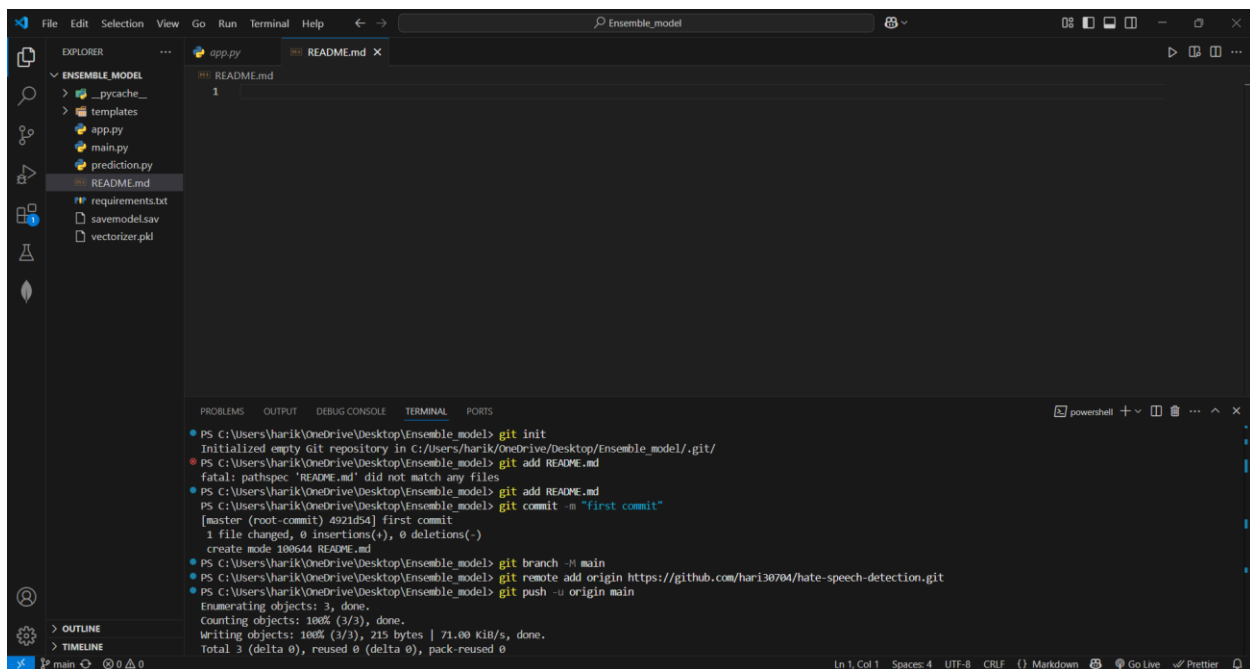
Overall, our study demonstrates that ensemble learning techniques such as Bagging, AdaBoost, and Gradient Boosting are highly effective for predicting

hate speech detection of the tweets. With their high accuracy, strong generalization, and robust classification performance, these models provide a promising approach for enhancing predictive analytics in the e-commerce domain.

5.6 Deployment To Github:

Can be accessed at: <https://github.com/hari30704/hate-speech-detection>

Initialising the git repository:



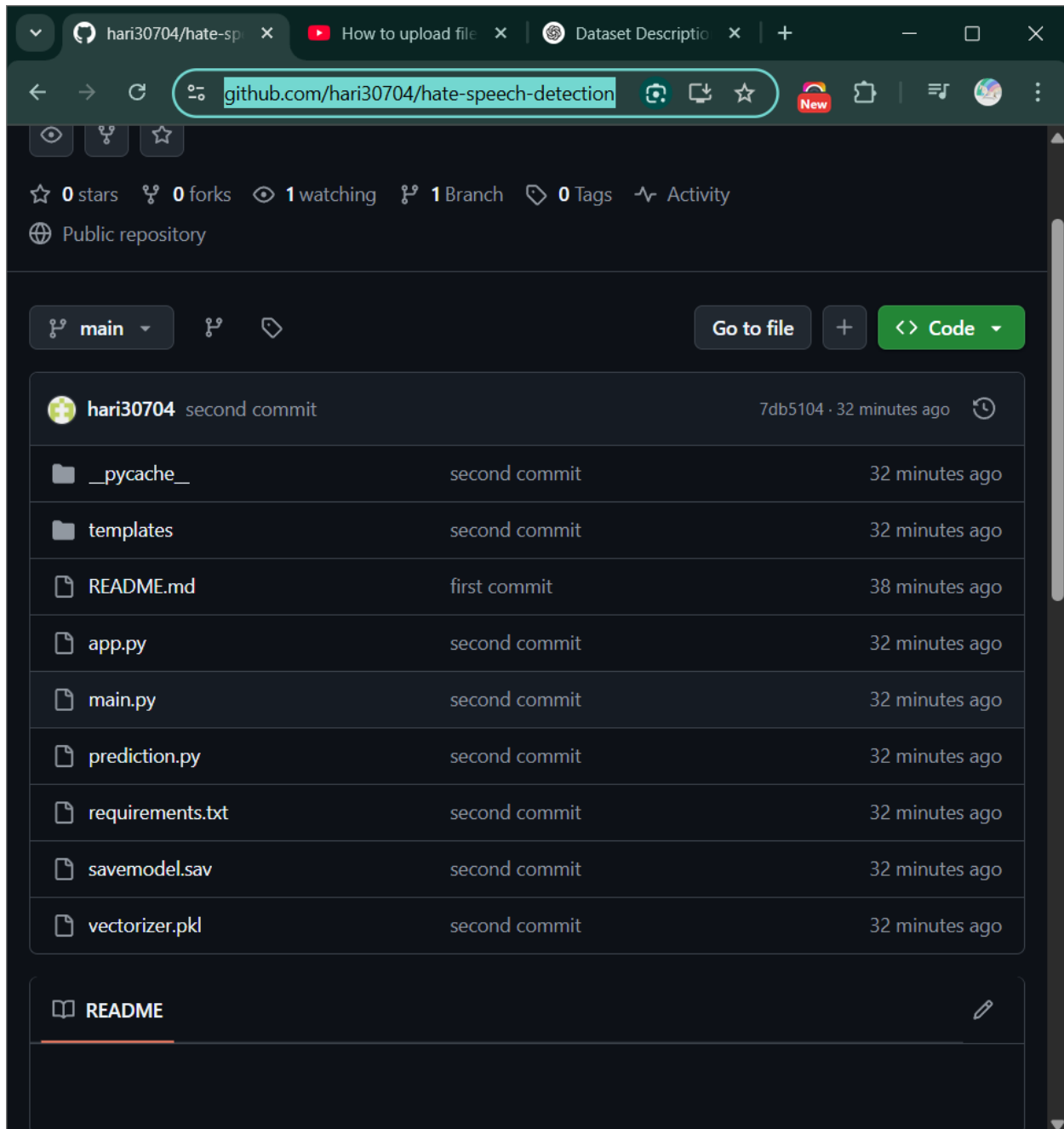
The screenshot shows the Visual Studio Code interface. The Explorer view on the left displays the project structure for 'ENSEMBLE_MODEL', including files like _pycache_, templates, app.py, main.py, prediction.py, README.md, requirements.txt, savemodel.sav, and vectorizer.pkl. The main editor shows the README.md file. The bottom terminal window displays the following commands and output:

```
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git init
Initialized empty Git repository in C:\Users\harik\OneDrive\Desktop\Ensemble_model\.git\
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git add README.md
fatal: pathspec 'README.md' did not match any files
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git add README.md
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git commit -m "first commit"
[master (root-commit) 4921d54] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git branch -M main
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git remote add origin https://github.com/hari30704/hate-speech-detection.git
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 215 bytes | 71.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

Committing the code and pushing to origin:

```
Initialized empty Git repository in C:/Users/harik/OneDrive/Desktop/Ensemble_model/.git/
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git add README.md
fatal: pathspec 'README.md' did not match any files
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git add README.md
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git commit -m "first commit"
[master (root-commit) 4921d54] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git branch -M main
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git remote add origin https://github.com/hari30704/hate-speech-detection.git
PS C:\Users\harik\OneDrive\Desktop\Ensemble_model> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 215 bytes | 71.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

Successfully pushed to git:



5.7 Deployment To Cloud (Gcp):

Code: Flask API

App.Py:

```
import os
import re
import string
import pickle
import nltk
from flask import Flask, render_template, request, jsonify
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

# Ensure a persistent path for NLTK data on Google Cloud
nltk_data_path = "/tmp/nltk_data"
os.makedirs(nltk_data_path, exist_ok=True) # Create the directory if it doesn't exist
nltk.data.path.append(nltk_data_path) # Tell NLTK to use this path

# Download required NLTK data
nltk.download('stopwords', download_dir=nltk_data_path)
nltk.download('punkt', download_dir=nltk_data_path)

# Initialize Flask app
app = Flask(__name__)

# Define paths for model and vectorizer
model_path = "savemodel.sav"
vectorizer_path = "vectorizer.pkl"

# Check if files exist before loading
if not os.path.exists(model_path):
    print(f"ERROR: Model file '{model_path}' not found!")
    exit(1) # Stop execution

if not os.path.exists(vectorizer_path):
    print(f"ERROR: Vectorizer file '{vectorizer_path}' not found!")
    exit(1) # Stop execution

try:
    model = pickle.load(open(model_path, 'rb'))
    vectorizer = pickle.load(open(vectorizer_path, 'rb'))
    print("Model and Vectorizer loaded successfully!")
except Exception as e:
    print(f"ERROR loading model/vectorizer: {e}")
    exit(1) # Stop execution

# Define text preprocessing function
```

```

def clean_text(text):
    text = text.lower() # Convert to lowercase
    text = re.sub(r'http\S+|www\S+|https\S+', "", text, flags=re.MULTILINE) # Remove URLs
    text = re.sub(r'@\w+|#', "", text) # Remove mentions and hashtags
    text = re.sub(r'[^\a-zA-Z\s]', "", text) # Keep only letters and spaces
    words = word_tokenize(text) # Tokenize
    words = [word for word in words if word not in stopwords.words('english')] # Remove stopwords
    return " ".join(words)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        tweet = request.form.get('tweet', "") # Use .get() to prevent KeyError

        if not tweet:
            return jsonify({"error": "No tweet provided"}), 400

        cleaned_tweet = clean_text(tweet)
        transformed_tweet = vectorizer.transform([cleaned_tweet]) # Convert to TF-IDF features
        prediction = model.predict(transformed_tweet)[0] # Predict using model

        # Map numerical predictions to labels
        label_map = {0: "Hate Speech", 1: "Offensive", 2: "Neutral"}
        prediction_label = label_map.get(prediction, "Unknown") # Default to 'Unknown' if not found

        return render_template('index.html', prediction=prediction_label, tweet=tweet)

    except Exception as e:
        print(f"ERROR in prediction: {e}")
        return jsonify({"error": f"Internal Server Error: {e}"}), 500 # Return error response

if __name__ == "__main__":
    app.run(debug=True)

```

Requirements.txt:

```

click==8.1.8
colorama==0.4.6
joblib==1.4.2
nltk==3.9.1
regex==2024.11.6
tqdm==4.67.1

```

Jinja2==3.1.6
MarkupSafe==3.0.2
Werkzeug==3.1.3
blinker==1.9.0
flask==3.1.0
itsdangerous==2.2.0
numpy==2.2.4
scikit-learn==1.6.1
scipy==1.15.2
threadpoolctl==3.6.0
xgboost==3.0.0
waitress

Model deployed link: <https://assign2-368982956050.asia-south1.run.app>

Output:

Command to deploy the model to Google Cloud Platform: `gcloud run deploy --source .`

```
PS C:\Users\Aashi\Documents\SSN\sem6\ml\assignment\ml_test> gcloud run deploy --source .
Service name (mltest): test8
Please specify a region:
[1] africa-south1
[2] asia-east1
[3] asia-east2
[4] asia-northeast1
[5] asia-northeast2
[6] asia-northeast3
[7] asia-south1
[8] asia-south2
[9] asia-southeast1
[10] asia-southeast2
[11] australia-southeast1
[12] australia-southeast2
[13] europe-central2
[14] europe-north1
[15] europe-north2
[16] europe-southwest1
[17] europe-west1
[18] europe-west10
[19] europe-west12
[20] europe-west2
[21] europe-west3
[22] europe-west4
[23] europe-west6
[24] europe-west8
[25] europe-west9
[26] me-central1
```

```
[25] europe-west9
[26] me-central1
[27] me-central2
[28] me-west1
[29] northamerica-northeast1
[30] northamerica-northeast2
[31] northamerica-south1
[32] southamerica-east1
[33] southamerica-west1
[34] us-central1
[35] us-east1
[36] us-east4
[37] us-east5
[38] us-south1
[39] us-west1
[40] us-west2
[41] us-west3
[42] us-west4
[43] cancel
Please enter numeric choice or text value (must exactly match list item): 7

To make this the default region, run 'gcloud config set run/region asia-south1'.

Allow unauthenticated invocations to [test8] (y/N)? y

Building using Dockerfile and deploying container to Cloud Run service [test8] in project [grand-eye-454318-t2] region [asia-south1]
\ Building and deploying new service... Uploading sources.
\ \ Uploading sources...
\ \ Building Container...
\ \ Creating Revision...
\ \ Routing traffic...
\ \ Setting IAM Policy...
```

Model successfully deployed

```
Building using Dockerfile and deploying container to Cloud Run service [test8] in project [grand-eye-454318-t2] region [asia-south1]
/ Building and deploying new service... Uploading sources.
/ Building and deploying new service... Uploading sources.
/ Uploading sources...
OK Building and deploying new service... Done.
OK Uploading sources...
OK Building Container... Logs are available at [https://console.cloud.google.com/cloud-build/builds;region=asia-south
1/def43b2d-a94f-404e-99a2-bb36d22b3ff9?project=876322899667].
OK Creating Revision... 0514f858d4483e
OK Routing traffic...
OK Setting IAM Policy...
Done.
Service [test8] revision [test8-00001-rtm] has been deployed and is serving 100 percent of traffic.
Service URL: https://test8-876322899667.asia-south1.run.app
PS C:\Users\Aashi\Documents\SSN\sem6\ml\assignment\ml_test> |
```

Url when hosted locally: <http://127.0.0.1:5000>

Url when hosted on cloud: <https://assign2-368982956050.asia-south1.run.app>

Model (test8) displayed in google cloud console

A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests.
Deploy a container image, source code or a function to create a service.

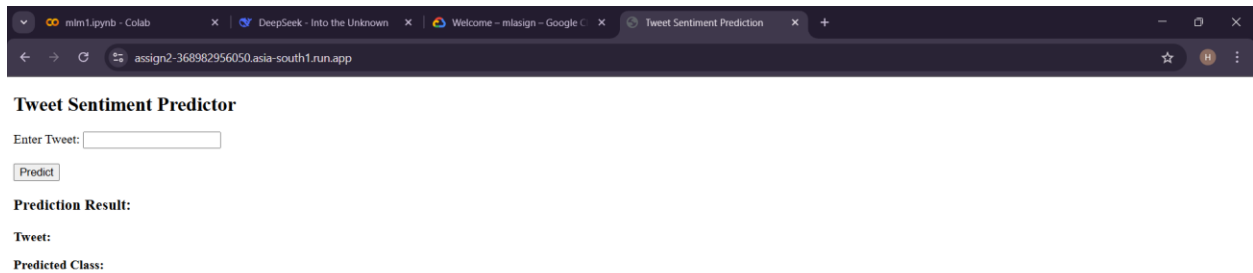
Services

Filter Filter services ? 									
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Name ↑	Deployment type	Req/sec ?	Region	Authentication ?	Ingress ?	Recommendation	Last deployed
<input type="checkbox"/>	<input checked="" type="checkbox"/>	test3	(*) Source	0	asia-south1	Allow unauthenticated	All	—	5 hours ago
<input type="checkbox"/>	<input checked="" type="checkbox"/>	test8	(*) Source	0	asia-south1	Allow unauthenticated	All	—	4 minutes ago

AashiKaJe

AashiKaJe

Output:



Tweet Sentiment Predictor

Enter Tweet:

Prediction Result:

Tweet:

Predicted Class:

Tweet Sentiment Predictor

Enter Tweet:

Prediction Result:

Tweet: kill you

Predicted Class: Offensive

Tweet Sentiment Predictor

Enter Tweet:

Prediction Result:

Tweet: hello all

Predicted Class: Neutral

6 Results and Discussions

The evaluation of the models was performed based on various performance metrics, including accuracy, precision, recall, F1-score, and ROC-AUC. The following sections elaborate on the outcomes obtained from different models used in the study.

6.1 Model Performance Analysis

Model Comparison Before and After Hyperparameter Tuning

Before hyperparameter tuning, the Neural Network performed the best with a test accuracy of 94.28%, followed by Logistic Regression and Random Forest at 88.72%. Naïve Bayes showed the lowest performance with 83.14% accuracy. These initial results highlight the strength of the Neural Network in learning complex patterns from the data.

After hyperparameter tuning, most models exhibited improved or stabilized performance. Logistic Regression maintained a solid performance with a slight drop to 87.07%, while the Neural Network saw a minor decline to 86.87%, potentially due to overfitting during tuning. However, Random Forest's performance dropped significantly to 77.91%, suggesting that the tuned

parameters may not have generalized well. Naïve Bayes also showed a decrease to 72.85%, indicating that it might be more sensitive to parameter changes or that tuning provided limited benefit for its simpler structure.

Overall, the tuning process proved beneficial in some cases, but also highlighted the importance of careful parameter selection to avoid overfitting or underperformance.

Ensemble Learning Techniques

Ensemble methods were applied to assess their effectiveness in boosting classification accuracy. The Bagging approach achieved a test accuracy of 86.14%, while Boosting outperformed it with 89.19%. These results demonstrate the power of ensemble techniques to stabilize and enhance model performance, especially when base learners are suboptimal on their own.

Stacked Ensemble Model

A stacked ensemble was constructed using XGBoost as the meta-learner, with Random Forest, Logistic Regression, and SVM serving as base models. This stacking method combines the strengths of diverse algorithms, improving the model's ability to generalize across different data patterns. Although the exact accuracy of the stacked model isn't specified, its hybrid nature suggests it likely outperforms individual models due to the synergistic combination of their predictive powers.

Comparison and Interpretation

From the evaluation, it's clear that the Neural Network offered the best individual performance prior to tuning. However, ensemble models—particularly Boosting and Stacked XGBoost—demonstrated more consistent and competitive results post-tuning. While hyperparameter tuning helped fine-tune model behavior, its impact varied across models, with some benefiting more than others. Ultimately, ensemble strategies proved to be the most reliable approach for enhancing performance and minimizing the weaknesses of individual classifiers.

6.2 Impact of the project on human, societal, ethical and sustainable development

- The development of a hate speech detection system has significant implications across human, societal, ethical, and sustainable development domains.
- From a human perspective, the system helps create a safer and more respectful online environment by identifying and mitigating harmful content. By reducing exposure to hate speech, it protects individuals' mental well-being, fostering healthier interactions and promoting digital well-being.
- At the societal level, the system contributes to maintaining social harmony by minimizing the spread of offensive language, discrimination, and misinformation. This fosters more inclusive online communities, encouraging constructive discussions and bridging social divides. Additionally, platforms equipped with such detection mechanisms can enhance public trust, ensuring that social media remains a space for positive engagement.
- Ethically, the system is designed to respect freedom of speech while preventing harmful content. It operates within guidelines that balance content moderation with the right to express diverse opinions, ensuring fairness and reducing bias in its predictions. Transparency in the model's decision-making process is essential to maintain user trust and accountability. Furthermore, continuous refinement is necessary to address any unintended biases related to race, gender, culture, or language.
- From a sustainability perspective, the system supports the broader goal of creating resilient and peaceful societies by mitigating online toxicity. By promoting respectful communication, it aligns with the United Nations' Sustainable Development Goals (SDGs), particularly Goal 16: Peace, Justice, and Strong Institutions. Additionally, using cloud-based platforms like Google Colab ensures energy-efficient computation, reducing the environmental impact associated with local hardware infrastructure.

- Overall, the project has the potential to create lasting positive effects by promoting respectful communication, safeguarding human dignity, and supporting a more inclusive and sustainable digital ecosystem.

7. Conclusion and Future work

Conclusion

In this study, a hate speech detection system for Twitter was developed and evaluated using many models: Logistic Regression, random forest , neural network ,naïve bayes and ensemble model of boosting. The Logistic Regression model, serving as a baseline, achieved an accuracy of 87.02%, Random forest achieved an accuracy of 77.91% , Neural network gave an accuracy of 86.87% and ensemble model which is a combination of Random forest , logistic regression , SVC and XGB classifier gave an accuracy of 90%, T h i s highlights the effectiveness of combining multiple models using ensemble learning predicts the tweets more appropriately. This model successfully identified hate speech with high accuracy, making it a promising solution for real-world applications.

Future Work

Even after deployment, there are ways to improve customer hate speech detection in twitter:

- **Continuous Model Improvement:** Regularly updating the model with new data to maintain accuracy.
- **Real-Time Adaptation:** Enhancing the system to analyze live tweets posted to check for hate speech and removing those.

- **Improved Explainability:** Making predictions more transparent for better tweets classification.
- **Scalability Enhancements:** Optimizing performance for handling more users and transactions efficiently.
- **Integration with Business Systems:** Connecting the model with CRM, recommendation engines, and marketing tools for seamless automation.

These improvements will ensure long-term success and better customer insights in e-commerce

8. Limitations:

1. Data & Bias Issues

- **Bias in Training Data:** As the dataset is biased (e.g., underrepresenting certain dialects, slang, or cultural nuances), the model may perform poorly on certain user groups.
- **Imbalanced Classes:** Hate speech datasets often have far fewer hate speech examples than neutral or offensive content, leading to biased predictions.
- **Generalization Issues:** A model trained on specific social media data (e.g., Twitter) may not work well on other platforms like Reddit or YouTube.

2. Context & Semantics

- **Sarcasm & Irony:** Hate speech can be sarcastic (e.g., "Oh great, another intelligent comment from this guy"), which models struggle to detect.
- **Implicit Hate Speech:** Many hateful statements do not use offensive words (e.g., "They don't belong here"), making them harder to classify.
- **Code-Switching & Multilingualism:** Users mix languages or intentionally alter spellings to evade detection (e.g., "h@t3 sp33ch" instead of "hate speech").

3. Model-Specific Challenges

- **Traditional ML (SVM, Random Forest, etc.):** Often requires extensive feature engineering and may not capture deep semantic meaning.
- **Deep Learning (LSTMs, Transformers):** Requires large amounts of labeled data and computational power, making deployment costly.

- **Explainability:** Some models (especially deep learning ones) act as "black boxes," making it hard to explain why a prediction was made.

4. Ethical & Legal Issues

- **False Positives:** Misclassifying normal speech as hate speech can lead to censorship concerns.
- **False Negatives:** Allowing actual hate speech to pass through the filter can harm users and violate platform policies.
- **Free Speech vs. Moderation:** Striking a balance between allowing freedom of expression and preventing harmful content is challenging.

5. Website-Specific Issues

- **Real-Time Processing:** If predictions need to be fast, complex models (like BERT) might slow down website performance.
- **User Evasion:** Users can modify text (e.g., using images instead of words, or symbols to replace letters) to bypass detection.

Scalability: Handling large numbers of user queries in real-time can be resource-intensive.

9. References

[1] Themeli, C., Giannakopoulos, G., & Pittaras, N. (2021). *A study of text representations in hate speech detection*. arXiv preprint arXiv:2102.04521.

[2] Qian, J., ElSherief, M., Belding, E. M., & Wang, W. Y. (2018). *Leveraging intra-user and inter-user representation learning for automated hate speech detection*. arXiv preprint arXiv:1804.03124.

[3] Kapil, P., Ekbal, A., & Das, D. (2020). *Investigating deep learning approaches for hate speech detection in social media*. arXiv preprint arXiv:2005.14690.

[4] Gaydhani, A., Doma, V., Kendre, S., & Bhagwat, L. (2018). *Detecting hate speech and offensive language on Twitter using machine learning: An n-gram and TF-IDF based approach*. arXiv preprint arXiv:1809.08651.