PROJECT REPORT

Introduction to AIML

CSL236

Faculty name: Dr. Nidhi Malik

Student name: Shivam , Gaurav

Roll No.: 21csu325 , 21csu316

Section: DS-V-B

Submission Date: 20-11-2023

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2023-24

CONTENTS

Project Title:

Hate Speech Recognition

Creating a hate speech recognition system is driven by social responsibility, enhancing user safety, and regulatory compliance. Utilizing AI algorithms, the project seeks to mitigate harm by identifying and moderating hate speech, fostering a more positive online environment. Emphasizing ethical AI use, developers aim to minimize biases in algorithms and contribute to a safer, inclusive user experience. The initiative extends beyond moderation, advancing research in natural language processing and sentiment analysis. Collaboration with communities and advocacy groups is integral, ensuring diverse perspectives are considered in addressing the evolving challenges of online content moderation. Ultimately, the project aligns with a broader commitment to fostering respectful communication and contributing to the ongoing improvement of digital spaces.

About The Project:

Hate Speech Recognition

Problem Statement: The problem revolves around classifying tweets into different sentiment categories, specifically identifying hate speech, offensive language, or non-offensive content. The dataset used for this project contains tweets labeled by CrowdFlower users, with each tweet categorized into one of the three classes: hate speech, offensive language, or neither.

Approach: The approach to solving this problem involves several key steps:

1.Data Loading:

- Load the labeled dataset containing information about the tweets, including the text content and corresponding labels.

2.Data Cleaning and Preprocessing:

- Clean and preprocess the text data to remove noise, irrelevant information, and standardize the format.
- Steps include converting to lowercase, removing URLs, special characters, and numbers, tokenization, removing stopwords, and lemmatization.

3.Text Vectorization:

- Convert the preprocessed text data into a format suitable for machine learning models.
- Two common vectorization techniques used are Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

4.Model Training:

- Split the dataset into training and testing sets.
- Train machine learning models on the vectorized text data. Models used include Support Vector Machine (SVM), Random Forest, and Naive Bayes.

5.Model Evaluation:

- Evaluate the trained models using metrics such as accuracy, classification report, and confusion matrix. This step assesses how well the models generalize to unseen data.

6.User Input Classification:

- Allow users to input text, preprocess the input using the same preprocessing steps, vectorize it using the trained vectorizer, and classify it using the trained model.

7.Model Saving and Loading:

- Save the trained Random Forest model and the BoW vectorizer for future use.
- Load the saved models for making predictions on new data.

8.Visualization:

- Visualize the distribution of sentiments in the dataset using bar charts.
- Create a word cloud to visually represent the most used words in the dataset

Machine learning model

Here's an explanation of each model and the potential reasons for using them in the context of hate speech recognition:

Support Vector Machine (SVM):

Type: Supervised Learning (both for classification and regression)

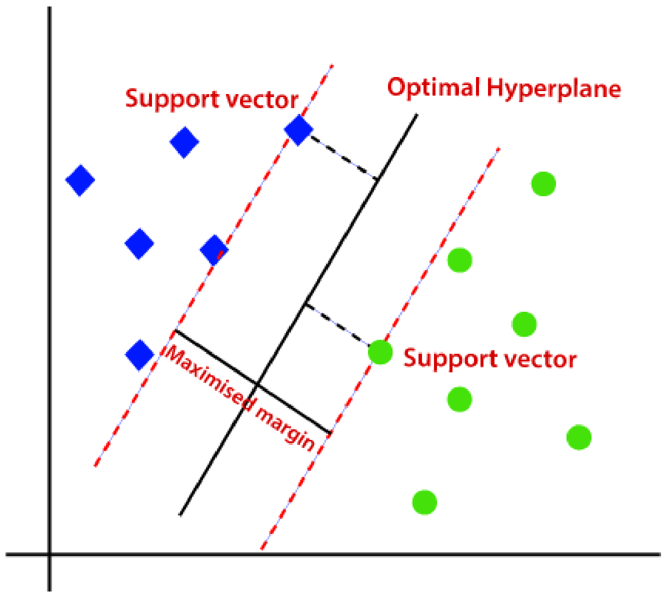
Use Case: SVM is widely used for classification tasks.

Working Principle: SVM works by finding a hyperplane in a high-dimensional space that best separates the data into different classes. It aims to maximize the margin between the classes.

Key Features:

Effective in high-dimensional spaces.

Versatile - various kernel functions can be used to handle different types of data.



Random Forest:

Type: Ensemble Learning (specifically, Bagging)

Use Case: Random Forest is often used for classification and regression tasks.

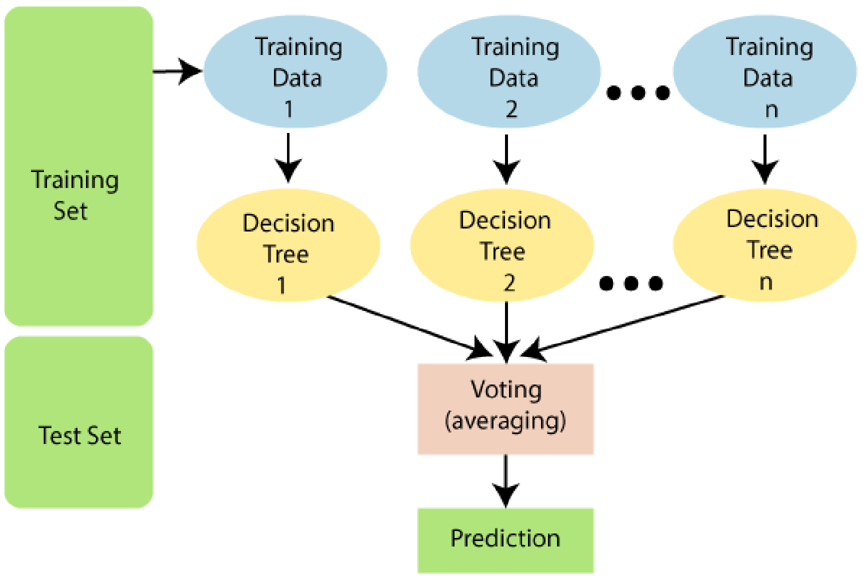
Working Principle: Random Forest builds multiple decision trees during training and merges them together to get a more accurate and stable prediction. It introduces randomness in the tree-building process by considering random subsets of features and data points.

Key Features:

Robust and less prone to overfitting.

Can handle large datasets with high dimensionality.

Provides feature importance.



Naive Bayes:

Type: Probabilistic, Bayesian

Use Case: Commonly used for classification tasks, especially in text classification (e.g., spam detection).

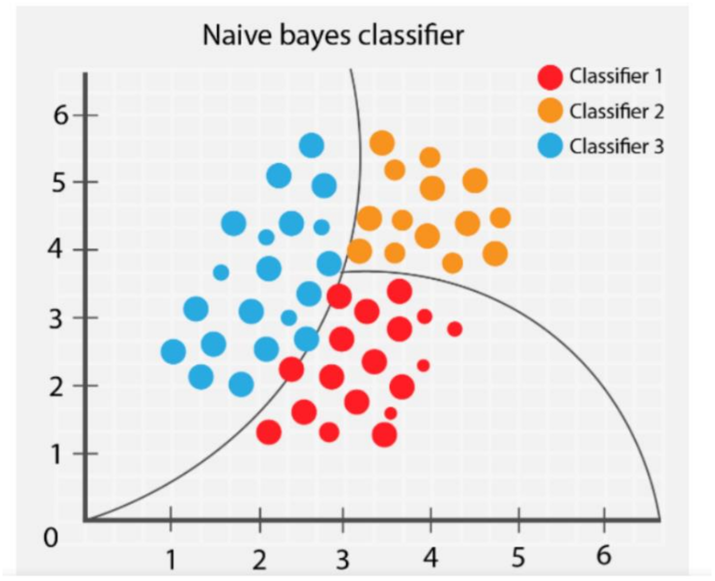
Working Principle: Naive Bayes is based on Bayes' theorem and assumes that features are conditionally independent given the class. Despite its "naive" assumption, it often performs well, especially in text-related tasks.

Key Features:

Simple and computationally efficient.

Works well with high-dimensional data.

Particularly effective in scenarios with relatively simple relationships between features.



Formula :

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Logistic Regression:

Type: Supervised Learning (Classification)

Use Case: Logistic Regression is widely used for binary classification problems, making it suitable for tasks where the goal is to categorize instances into two classes, such as hate speech and non-hate speech.

Reasons for Using Logistic Regression in Hate Speech Recognition:

1. Interpretability: Logistic Regression provides coefficients for each feature, allowing for easy interpretation of the impact of individual features on the classification decision. This can be crucial in understanding which words or patterns contribute to identifying hate speech.

2. Efficiency with Linear Relationships: Hate speech detection may exhibit linear relationships between certain features and the likelihood of the content being hateful. Logistic Regression is efficient in capturing such linear relationships.

3. Probabilistic Output: Logistic Regression outputs probabilities, indicating the likelihood of a tweet being hate speech. This probabilistic nature allows for a nuanced understanding of the model's confidence in its predictions.

4. Baseline Model: Logistic Regression serves as a good baseline model due to its simplicity. It helps establish a performance benchmark against more complex models, providing insights into whether the problem requires more sophisticated approaches.

5. **Ease of Implementation:**Logistic Regression is relatively easy to implement and computationally efficient. It can be a practical choice when working with large datasets or in scenarios where model interpretability is essential.

k-Nearest Neighbors (KNN):

Type: Supervised Learning (Classification or Regression)

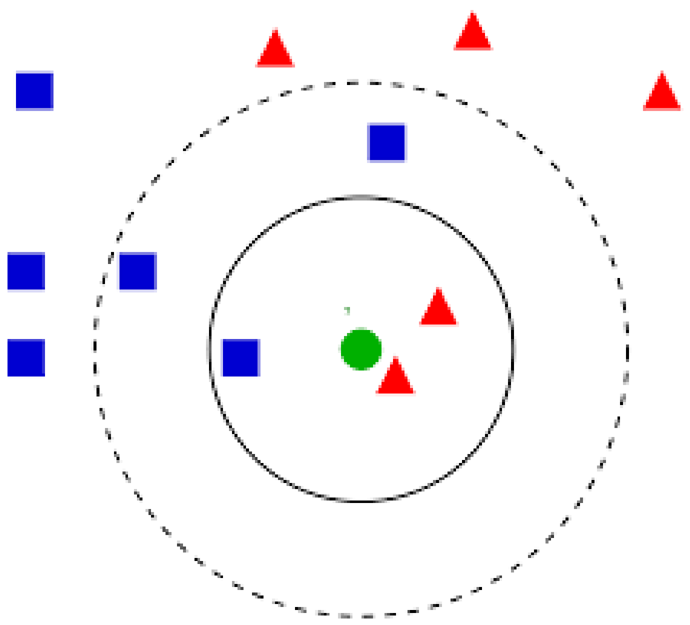
Use Case:KNN is a versatile algorithm used for both classification and regression tasks. It is particularly effective when the decision boundary is complex or nonlinear.

Working Principle: Given a new data point, KNN classifies or predicts its label based on the labels of its k-nearest neighbors in the feature space.
For classification, the majority class among the neighbors is assigned to the new data point. For regression, the average of the neighbors' values is used.

Hyperparameter:The crucial hyperparameter is 'k,' the number of neighbors to consider. The choice of 'k' affects the model's sensitivity to noise and its ability to capture patterns.

Distance Metric:The choice of distance metric (Euclidean, Manhattan, etc.) influences how the algorithm measures the proximity between data points.

Decision Boundary: The decision boundary is non-linear and can adapt to the shape of the data.



Reasons for Using Multiple Models:

Diversity in Approaches:

Each model makes different assumptions and uses distinct mechanisms for classification. Using multiple models provides a diverse set of approaches to capturing patterns in the data.

Ensemble Learning:

Combining the predictions of multiple models can result in a more robust and accurate overall model. Ensemble methods like Random Forest naturally leverage this principle.

Model Comparison:

By implementing different models, you can compare their performances using metrics like accuracy, precision, recall, and F1-score. This helps identify the most suitable model for your specific hate speech recognition task.

Handling Complex Data:

Hate speech data can be complex, and different models may excel in capturing different aspects of this complexity. Employing multiple models allows you to address the diversity of hate speech content.
In summary, the use of SVM, Random Forest, and Naive Bayes in your hate speech recognition project reflects a thoughtful approach to model selection. These models, with their distinct characteristics, provide a comprehensive strategy for addressing the challenges inherent in identifying and classifying hate speech. The final choice may depend on empirical performance results and considerations specific to your project requirements.

Libraries Used In Project:

1.Pandas (import pandas as pd):

•Used for data manipulation and analysis. It provides data structures like dataframes for handling structured data.

2.Regular Expressions (import re):

•Useful for pattern matching and text manipulation. This can be especially handy for preprocessing text data.

3.NLTK (Natural Language Toolkit):

•from nltk.corpus import stopwords: Provides a list of common English stop words.

•from nltk.tokenize import word_tokenize: Tokenizes sentences into words.

•from nltk.stem import WordNetLemmatizer: Performs lemmatization, reducing words to their base or root form.

•Used for natural language processing tasks.

4.Scikit-learn:

•from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer: Used for converting text data into numerical features (Bag of Words or TF-IDF representation).

•from sklearn.model_selection import train_test_split: Splits data into training and testing sets.

•from sklearn.svm import SVC: Support Vector Machine classifier.

•from sklearn.ensemble import RandomForestClassifier: Random Forest classifier.

5.Seaborn (import seaborn as sns) and Matplotlib (import matplotlib.pyplot as plt):

•Used for data visualization, particularly for creating plots and graphs.

6.Joblib (import joblib):

•Used for saving and loading machine learning models.

7.String (import string):

•Provides a collection of string constants (e.g., ASCII letters, digits, punctuation) that can be useful intext processing.

8.WordCloud (from wordcloud import WordCloud):

•Used for creating word clouds, which visually represent the most frequent words in a text corpus.

Dataset Overview:

The dataset consists of the following columns:

1.'Unnamed: 0': An index or identifier for each row.

2.'count': Represents the count of a certain feature or occurrences, though the specific context is not explicitly mentioned.

3.'hate_speech': Indicates the count or presence of instances labeled as hate speech.

4.'offensive_language': Indicates the count or presence of instances labeled as offensive language.

5.'neither': Indicates the count or presence of instances that are neither categorized as hate speech nor offensive language.

6.'class': Represents the class label associated with each row, categorizing the type of content in the 'tweet' column.

7.'tweet': Contains the actual text content of the tweet.

Dataset Characteristics:

•The dataset is free of any null values in any row, ensuring completeness for analysis.

Interpretation and Context:

•The dataset appears to be related to text classification, potentially for hate speech or offensive language detection.

•The 'class' column likely provides categorical labels for the type of content present in the tweets.

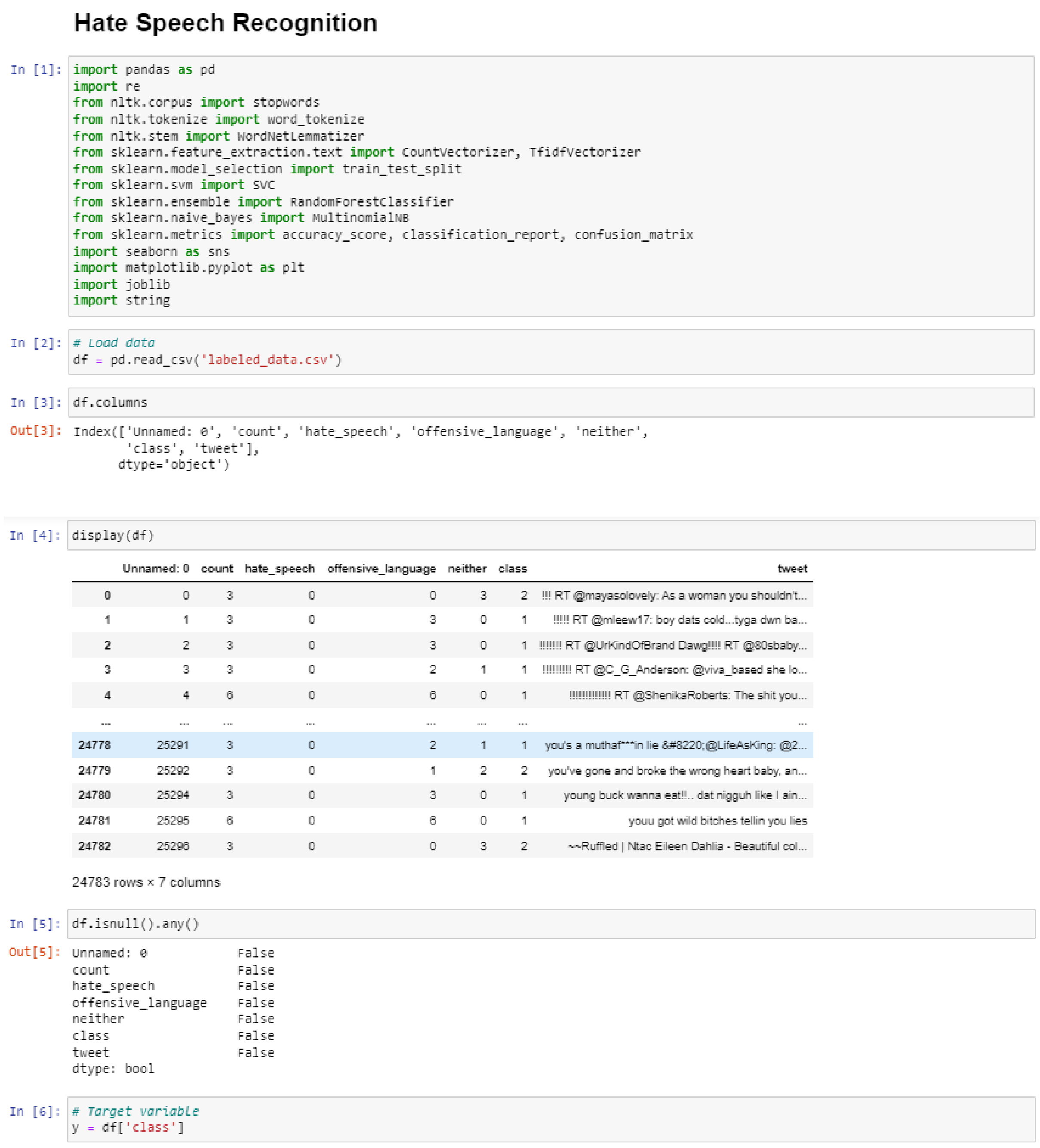
Recommendations for Analysis:

•Consider using the 'tweet' column as the input feature and the 'class' column as the target variable for text classification tasks.

•Explore the distribution of samples across different classes in the 'class' column.

•Preprocess the 'tweet' column, which may involve text cleaning and tokenization, before building machine learning models.

Screenshots:



Hate Speech Recognition

```
In [1]: import pandas as pd
        import re
        from nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize
        from nltk.stem import WordNetLemmatizer
        from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
        from sklearn.model_selection import train_test_split
        from sklearn.svm import SVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
        import seaborn as sns
        import matplotlib.pyplot as plt
        import joblib
        import string
```

```
In [2]: # Load data
        df = pd.read_csv('labeled_data.csv')
```

```
In [3]: df.columns

Out[3]: Index(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither',
               'class', 'tweet'],
              dtype='object')
```

```
In [4]: display(df)
```

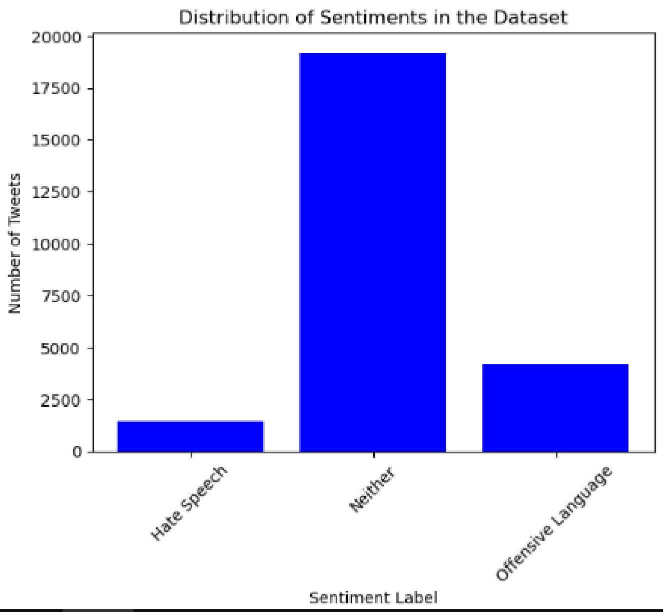
	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
...	...	...	...	...	...	...	...
24778	25291	3	0	2	1	1	you's a muthaf***in lie &#8220;@LifeAsKing: @2...
24779	25292	3	0	1	2	2	you've gone and broke the wrong heart baby, an...
24780	25294	3	0	3	0	1	young buck wanna eat!!.. dat nigguh like I ain...
24781	25295	6	0	6	0	1	youu got wild bitches tellin you lies
24782	25296	3	0	0	3	2	~~Ruffled | Ntac Eileen Dahlia - Beautiful col...

24783 rows × 7 columns

```
In [5]: df.isnull().any()

Out[5]: Unnamed: 0            False
        count                 False
        hate_speech           False
        offensive_language    False
        neither               False
        class                 False
        tweet                 False
        dtype: bool
```

```
In [6]: # Target variable
        y = df['class']
```

```
In [7]: import matplotlib.pyplot as plt

        # Assuming 'class' is the column containing the class labels (0, 1, 2)
        sentiment_distribution = df['class'].value_counts()


        order = [2, 1, 0]
        # Plotting the bar chart
        plt.bar(sentiment_distribution.index, sentiment_distribution.values, color='blue')
        plt.xlabel('Sentiment Label')
        plt.ylabel('Number of Tweets')
        plt.title('Distribution of Sentiments in the Dataset')
        plt.xticks(sentiment_distribution.index, ['Neither', 'Offensive Language', 'Hate Speech'], rotation=45)
        plt.show()
```



```
In [8]: def clean_and_preprocess(text):
            text = str(text).lower()
            text = re.sub('\[.*?\]', '', text)
            text = re.sub('https?://\S+|www\.\S+', '', text)
            text = re.sub('<.*?>+', '', text)
            text = re.sub(r"\@w+|\#",'',text)
            text = re.sub(r"[^\w\s]",'',text)
            text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
            text = re.sub('\n', '', text)
            text = re.sub('\w*\d\w*', '', text)

            # Tokenization
            tokens = word_tokenize(text)

            # Remove stopwords
            stop_words = set(stopwords.words('english'))
            tokens = [token for token in tokens if token not in stop_words]

            # Lemmatization
            lemmatizer = WordNetLemmatizer()
            tokens = [lemmatizer.lemmatize(token) for token in tokens]

            # Join the tokens back into a single string
            preprocessed_text = ' '.join(tokens)

            return preprocessed_text
```

```
In [9]: def vectorize_text(corpus, vectorizer):
            X = vectorizer.fit_transform(corpus)
            print(f"{vectorizer.__class__.__name__} Matrix:")
            print(X.toarray())
            return X
```

In [10]:
```python
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model Accuracy: {accuracy}")

    classification_rep = classification_report(y_test, y_pred)
    print("\nClassification Report:\n", classification_rep)

    conf_matrix = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:\n", conf_matrix)

    # Plotting the Confusion Matrix Heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Neither', 'Offensive Language', 'Hate Speech'],
            yticklabels=['Neither', 'Offensive Language', 'Hate Speech'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

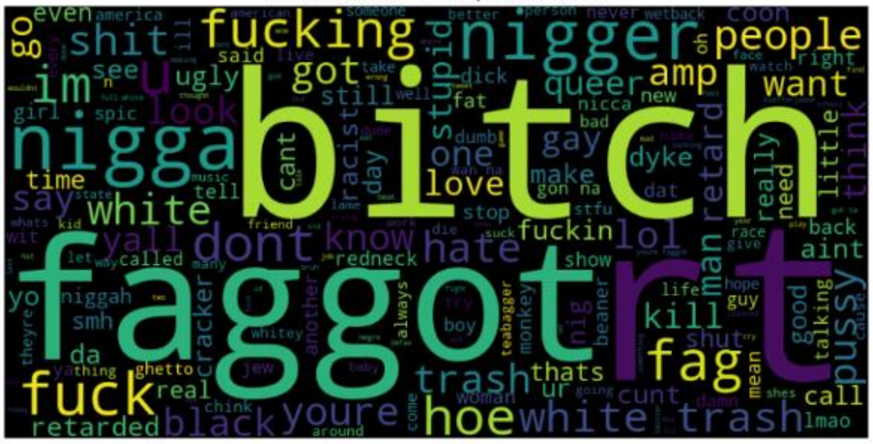
In [11]:
```python
# Clean and preprocess text
df['processed_tweet'] = df['tweet'].apply(clean_and_preprocess)
```

In [12]:
```python
# Vectorization
corpus = df['processed_tweet'].tolist()
```

In [13]:
```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Print column names to check for the correct column name
print(df.columns)

# Update the column name based on your DataFrame
hate_speech_tweets = ' '.join(df[df['class'] == 0]['processed_tweet'])

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='black').generate(hate_speech_tweets)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Hate Speech Tweets')
plt.show()
```

```
Index(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither',
       'class', 'tweet', 'processed_tweet'],
      dtype='object')
```

Word Cloud for Hate Speech Tweets



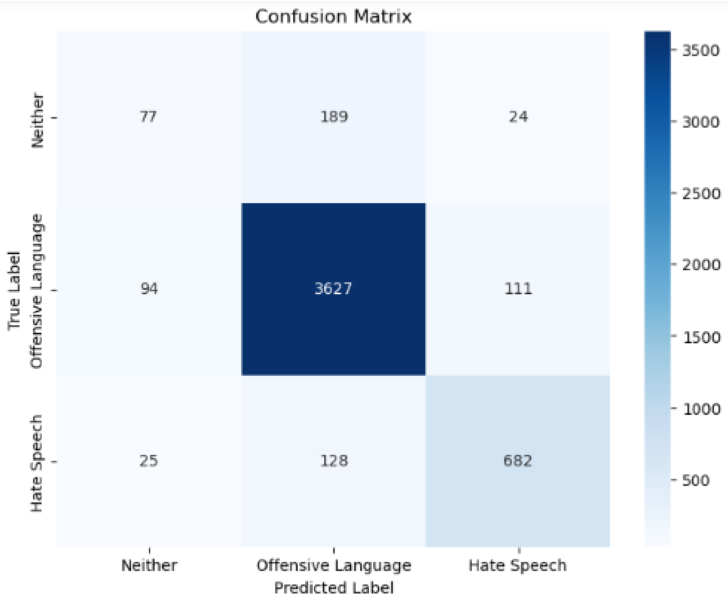
In [14]:
```python
# Bag-of-Words (BoW) Representation
vectorizer_bow = CountVectorizer(max_features=5000)  # Adjust max_features as needed
X_bow = vectorize_text(corpus, vectorizer_bow)
```

```
CountVectorizer Matrix:
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

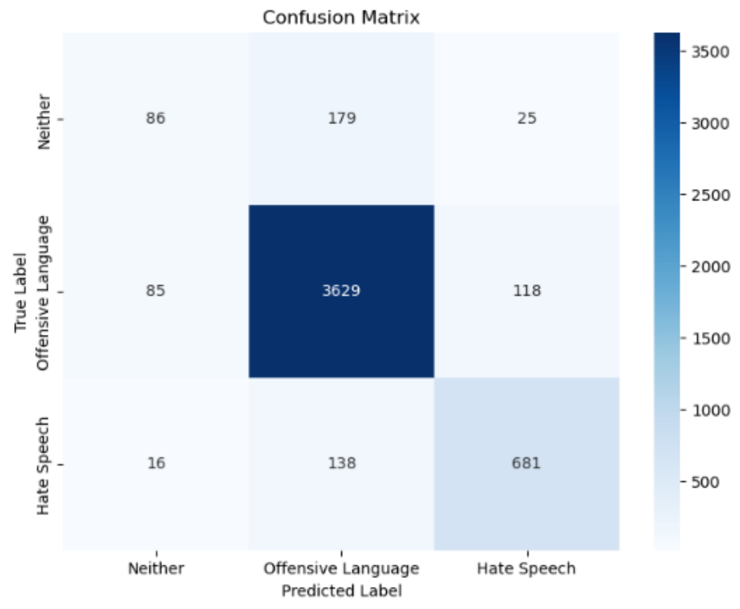
```
In [15]: # TF-IDF Representation
         vectorizer_tfidf = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
         X_tfidf = vectorize_text(corpus, vectorizer_tfidf)

         TfidfVectorizer Matrix:
         [[0. 0. 0. ... 0. 0. 0.]
          [0. 0. 0. ... 0. 0. 0.]
          [0. 0. 0. ... 0. 0. 0.]
          ...
          [0. 0. 0. ... 0. 0. 0.]
          [0. 0. 0. ... 0. 0. 0.]
          [0. 0. 0. ... 0. 0. 0.]]
```

```
In [16]: # Split the data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X_bow, y, test_size=0.2, random_state=42)
```

```
In [17]: # SVM Model
         svm_model = SVC(kernel='linear', random_state=42)
         train_and_evaluate_model(svm_model, X_train, y_train, X_test, y_test)

         Model Accuracy: 0.8848093605003026

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.39      0.27      0.32       290
                    1       0.92      0.95      0.93      3832
                    2       0.83      0.82      0.83       835

             accuracy                           0.88      4957
            macro avg       0.72      0.68      0.69      4957
         weighted avg       0.87      0.88      0.88      4957

         Confusion Matrix:
          [[  77  189   24]
          [  94 3627  111]
          [  25  128  682]]
```

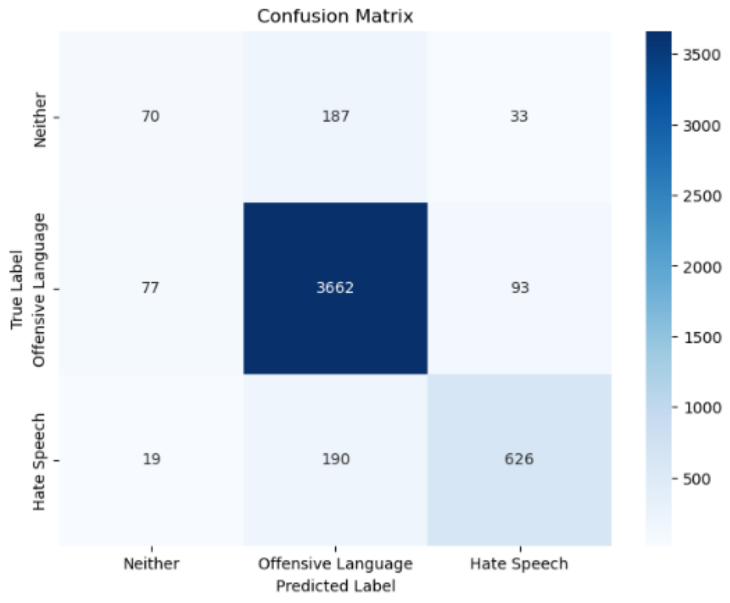


```
In [18]: # Random Forest Model
         rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
         train_and_evaluate_model(rf_model, X_train, y_train, X_test, y_test)

         Model Accuracy: 0.8868267097034497

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.46      0.30      0.36       290
                    1       0.92      0.95      0.93      3832
                    2       0.83      0.82      0.82       835

             accuracy                           0.89      4957
            macro avg       0.74      0.69      0.70      4957
         weighted avg       0.88      0.89      0.88      4957

         Confusion Matrix:
          [[  86  179   25]
          [  85 3629  118]
          [  16  138  681]]
```



```
In [19]: # Naive Bayes Model
         nb_model = MultinomialNB()
         train_and_evaluate_model(nb_model, X_train, y_train, X_test, y_test)
```

```
Model Accuracy: 0.8791607827314908

Classification Report:
               precision    recall  f1-score   support

           0       0.42      0.24      0.31       290
           1       0.91      0.96      0.93      3832
           2       0.83      0.75      0.79       835

    accuracy                           0.88      4957
   macro avg       0.72      0.65      0.68      4957
weighted avg       0.87      0.88      0.87      4957

Confusion Matrix:
 [[  70  187   33]
 [  77 3662   93]
 [  19  190  626]]
```



Recognizing and classifying the user input

In [20]:
```python
# Save the Random Forest model
model_save_path = '/Users/rav/Codes/AI project/model.pkl'
joblib.dump(rf_model, model_save_path)
print(f"Random Forest model saved to {model_save_path}")

# Save the BoW vectorizer
vectorizer_save_path = '/Users/rav/Codes/AI project/vectorizer_bow.pkl'
joblib.dump(vectorizer_bow, vectorizer_save_path)
print(f"Vectorizer saved to {vectorizer_save_path}")

# Load the Random Forest model and the saved vectorizer
loaded_rf_model = joblib.load(model_save_path)
loaded_vectorizer_bow = joblib.load(vectorizer_save_path)
print("Model and vectorizer loaded successfully.")

# User input
user_input = input("Enter the text to classify: ")

# Preprocess and vectorize the input
cleaned_input = clean_and_preprocess(user_input)
vectorized_input = loaded_vectorizer_bow.transform([cleaned_input])

# Make predictions
prediction = loaded_rf_model.predict(vectorized_input)

# Interpret the result
if prediction == 0:
    result = "Hate Speech"
elif prediction == 1:
    result = "Offensive Language"
else:
    result = "Non-Offensive"

print(f"The input text is classified as: {result}")
```

```
Random Forest model saved to /Users/rav/Codes/AI project/model.pkl
Vectorizer saved to /Users/rav/Codes/AI project/vectorizer_bow.pkl
Model and vectorizer loaded successfully.
Enter the text to classify: Its warm today
The input text is classified as: Non-Offensive
```

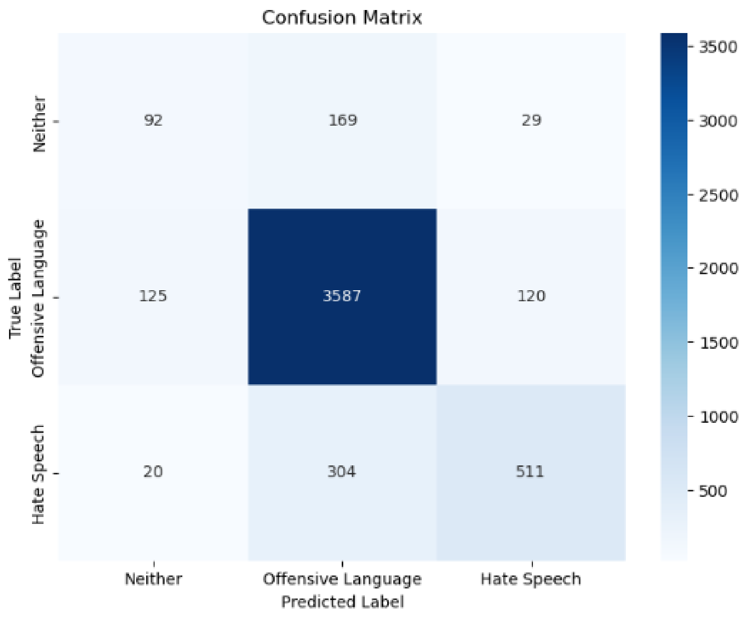
In [24]:
```python
# Save the Logistic Regression model
logreg_model_save_path = '/Users/rav/Codes/AI project/logreg_model.pkl'
joblib.dump(logreg_model, logreg_model_save_path)
print(f"Logistic Regression model saved to {logreg_model_save_path}")

# Save the BoW vectorizer (assuming vectorizer_bow is already loaded)
vectorizer_save_path = '/Users/rav/Codes/AI project/vectorizer_bow.pkl'
joblib.dump(vectorizer_bow, vectorizer_save_path)
print(f"Vectorizer saved to {vectorizer_save_path}")

# Load the Logistic Regression model and the saved vectorizer
loaded_logreg_model = joblib.load(logreg_model_save_path)
loaded_vectorizer_bow = joblib.load(vectorizer_save_path)
print("Model and vectorizer loaded successfully.")

# User input
user_input = input("Enter the text to classify: ")

# Preprocess and vectorize the input
cleaned_input = clean_and_preprocess(user_input)
vectorized_input = loaded_vectorizer_bow.transform([cleaned_input])

# Make predictions
prediction = loaded_logreg_model.predict(vectorized_input)

# Interpret the result
if prediction == 0:
    result = "Hate Speech"
elif prediction == 1:
    result = "Offensive Language"
else:
    result = "Non-Offensive"

print(f"The input text is classified as: {result}")
```

```
Logistic Regression model saved to /Users/rav/Codes/AI project/logreg_model.pkl
Vectorizer saved to /Users/rav/Codes/AI project/vectorizer_bow.pkl
Model and vectorizer loaded successfully.
Enter the text to classify: India is a great nation
The input text is classified as: Non-Offensive
```

```
In [25]: from sklearn.neighbors import KNeighborsClassifier

         # Example:
         knn_model = KNeighborsClassifier()
         train_and_evaluate_model(knn_model, X_train, y_train, X_test, y_test)
```

```
Model Accuracy: 0.8452693161186201

Classification Report:
              precision    recall  f1-score   support

           0       0.39      0.32      0.35       290
           1       0.88      0.94      0.91      3832
           2       0.77      0.61      0.68       835

    accuracy                           0.85      4957
   macro avg       0.68      0.62      0.65      4957
weighted avg       0.84      0.85      0.84      4957

Confusion Matrix:
 [[  92  169   29]
 [ 125 3587  120]
 [  20  304  511]]
```

```
/Users/rav/Applications/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:237: FutureWarning: Unlike o
ther reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statis
tic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this w
arning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



References:

Kaggle Dataset : Heart Failure Prediction

Google : https://www.google.com/

Towards Data Science : https://www.kaggle.com/datasets/mrmorj/hate-speech-and-offensive-language-dataset

GeeksforGeeks (To learn about ML Algorithm Models) : https://www.geeksforgeeks.org