Project: Hate Speech Detection using Transformers

Name: Seoyoung Kim
Email: idellio007@gmail.com
Country: USA
College: University of Michigan
Specialization: NLP
https://github.com/syoungk7/Hate_Speech_Detection

1. Problem description

With the advent of social media, people can express their opinions at any time, regardless of time and space. However, this freedom of expression has made it possible to use derogatory or discriminatory language against individuals or groups. Racial and social discrimination continues online, too. Detecting and mitigating such negative and hate-related speech will help create a safe and inclusive online environment.

2. Project Plan

1. **Understand the problem**: We understand the nature of hate speech, its impact, and the importance of addressing it on online platforms. We define the problem as a sentiment classification task based on labeled Twitter data.
2. **Data cleaning and normalization**: Preprocess the dataset by cleaning and normalizing the text data. This includes handling noise, removing irrelevant information, and ensuring consistency in the representation of tweets.
3. **Expression learning**: We leverage Transformer-based architecture for representation learning. Transformers have achieved significant success in natural language processing tasks and are well-suited to capturing contextual information in text data.

4. **Model building and training**: We develop a deep learning model for hate speech detection using preprocessed data. The model is trained on the training data set to optimize accuracy and minimize false positives and false negatives.
5. **Performance evaluation and reporting**: Evaluate model performance on the test dataset using relevant metrics such as accuracy, precision, recall, and F1 score. Provides a comprehensive report on the strengths and limitations of the model.
6. **Model Deployment**: Deploy the trained model into a production environment, ready to integrate with online platforms to actively detect and filter hate speech.
7. **Model inference**: By implementing the model's inference mechanism, we can classify new tweets in real time to identify and flag potential instances of hate speech.

3. Data Understanding

- What type of data you have got for analysis
    a. The provided dataset for hate speech detection from Twitter consists of two files: test_tweets_anuFYb8.csv and train_E6oV3lV.csv.
    b. The data attributes include a label (0 or 1) and text_format, which contains the original tweets with noise.
- What are the problems in the data (number of NA values, outliers , skewed etc.)
    a. Missing Values (NA): no
    b. Outliers in Text Length: no
    c. Class Imbalance: no
    d. Noise in Text: *need to handle*
- What approaches you are trying to apply on your data set to overcome problems like NA value, outlier etc. and why?
    a. **Text Cleaning and Normalization**: Implement text cleaning techniques to remove noise, special characters, and irrelevant information. Normalize the text by converting to lowercase, handling contractions, and addressing variations in language.

b. **Tokenization and Padding**: Utilize tokenization to convert text into smaller units suitable for input to Transformer models. Apply padding to ensure uniform input lengths, which is crucial for efficient model training.

c. **Exploratory Data Analysis (EDA)**: Conduct a comprehensive exploratory data analysis to understand the distribution of key variables, detect anomalies, and make informed decisions on data preprocessing strategies.

d. **Feature Engineering**: Consider additional feature engineering if necessary, such as extracting information from tweet timestamps, user mentions, or hashtags, to enhance the model's ability to capture relevant patterns.

e. **Validation and Cross-Validation**: Use validation sets and cross-validation techniques to assess the model's performance robustly, ensuring it generalizes well to unseen data.

4. Data Cleansing and Transformation

Cleaning noisy text data is crucial for Hate Speech Detection using Natural Language Processing:

a. Text Cleaning

i. Removing URLs and User Mentions using regex

Usage: URL – 'http\S+', Mentions - '@\w+'

ii. Expending Contractions using python module - contractions

(reference: https://www.geeksforgeeks.org/nlp-expand-contractions-in-text-processing/)

Usage: contractions.fix()

iii. Removing Special Characters and Punctuation using regex

Usage: '[^a-zA-Z0-9\s]'

b. Tokenization

i. Tokenization - from nltk.tokenize import word_tokenize

Usage: word_tokenize(new_text)

ii. Lowercasing – core function

Usage: token.lower()

c. **Removing Stopwords** - from nltk.corpus import stopwords

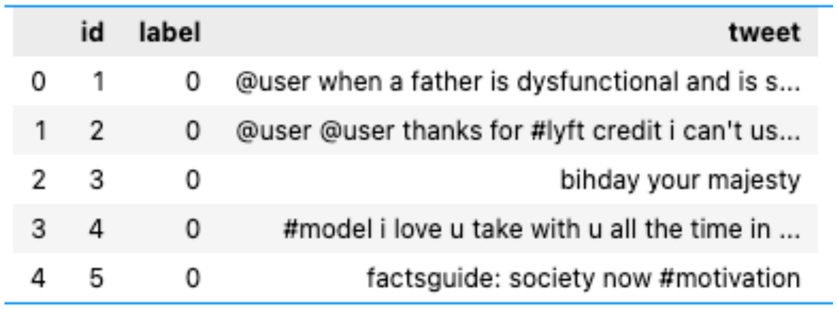
Usage: stop_words = set(stopwords.words('english'))

d. **Lemmatization** - from nltk.stem import WordNetLemmatizer

Usage: lemmatizer = WordNetLemmatizer()

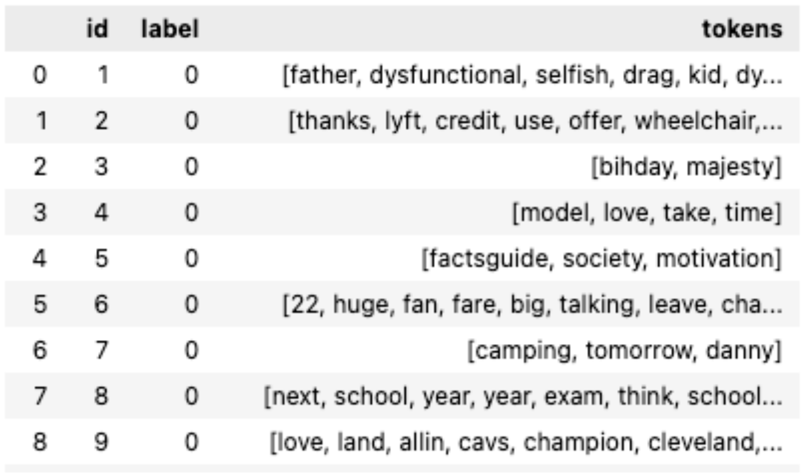
Result

- Raw data



	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation

- Cleaned data



	id	label	tokens
0	1	0	[father, dysfunctional, selfish, drag, kid, dy...
1	2	0	[thanks, lyft, credit, use, offer, wheelchair,...
2	3	0	[bihday, majesty]
3	4	0	[model, love, take, time]
4	5	0	[factsguide, society, motivation]
5	6	0	[22, huge, fan, fare, big, talking, leave, cha...
6	7	0	[camping, tomorrow, danny]
7	8	0	[next, school, year, year, exam, think, school...
8	9	0	[love, land, allin, cavs, champion, cleveland,...

5. Model Selection

- **Model Selection**: Multinomial Naive Bayes (MNB), Random Forest (RF), Decision Tree (DT), Logistic Regression (LR), Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Stochastic Gradient Descent (SGD), XGBoost (XGB) Classifiers
- **Feature Extraction:** TF-IDF

6. Model Building & Training

6.1 Data Preprocessing

1) **Importing Libraries**
   The initial step involves importing necessary libraries for data analysis and preprocessing, including %matplotlib inline for inline plotting in Jupyter notebooks, Pandas for data manipulation, Seaborn for data visualization, and Matplotlib for additional plotting.
2) **Loading Dataset**
   The training dataset (train_E6oV3lV.csv) is loaded using Pandas, and the first five rows are displayed. Basic information about the dataset is provided, including its type, keys, shape, data types, and descriptive statistics.
3) **Check Dataset**
   Data integrity checks are performed, including a pie chart showing the distribution of classes, checking for duplicate rows, and identifying missing or null values.
4) **Text Preprocessing including Data Cleaning**
   Several text preprocessing steps are performed to clean and prepare the text data for modeling. The steps include:
   - Cleaning Text: Removing URLs and mentions.
   - Expanding Contractions: Expanding contractions in the text.
   - Removing Non-Alphabetic Characters: Extracting only alphabets.
   - Removing Stopwords: Eliminating common English stopwords.
   - Lemmatization: Reducing words to their base or root form.

- Tokenization: Breaking text into individual tokens.

5) **Word Frequency Analysis and Generate Word Clouds**

   Word frequency analysis is conducted to understand the distribution of words in the dataset. The most common words in both hate and non-hate tweets are identified. Word clouds are generated to visually represent the most frequent words in both hate and non-hate tweets.

6.2 Model Building

1) **Import Libraries**

   Libraries related to model building and evaluation are imported, including Scikit-Learn's tools for machine learning, various classifiers (SVM, Decision Tree, K-Nearest Neighbors, etc.), and evaluation metrics.

2) **Load Dataset**

   The training (train_E6oV3lV.csv)and test datasets (test_tweets_anuFYb8.csv) are loaded, and preprocessing steps (cleaning, expansion of contractions, removing non-alphabetic characters, removing stopwords, lemmatization, and tokenization) are applied.

3) **Oversampling**

   Oversampling is performed to balance the distribution of classes in the training data. The minority class (label 1) is oversampled to match the count of the majority class (label 0).

4) **Model Building**

   The dataset is split into training and validation sets, and multiple machine learning models (Naive Bayes, Random Forest, Decision Tree, Logistic Regression, SVM, K-Nearest Neighbors, SGD, and XGBoost) are trained and evaluated using TF-IDF features. Model evaluation metrics include accuracy, precision, recall, F1 score, and ROC-AUC score.

5) **Evaluation Visualization**

   Precision-recall curves and ROC curves are plotted for model evaluation, providing insights into the trade-off between precision and recall and the receiver operating characteristic, respectively.

6) **Model Testing**

A RandomForestClassifier model is trained using the training data and tested on the test dataset. The model is saved as a pickle file (model_RF.pkl). Predictions are made on the test data, and the number of hate and non-hate predictions is reported.

7) **Word Frequency Analysis on Test Data**

Word frequency analysis is conducted on the test data to identify the most common words in hate and non-hate tweets. Word clouds are generated to visually represent these word frequencies.

6.3 Code for Modeling

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve, classification_report, PrecisionRecallDisplay
from sklearn.pipeline import make_pipeline

# oversampling
label_0 = train_data[train_data['label'] == 0] # label counting
label_1 = train_data[train_data['label'] == 1]
oversample = label_1.sample(len(label_0), replace=True, random_state=123)
train_oversampled = pd.concat([oversample, label_0], axis=0)
train_oversampled = train_oversampled.sample(frac = 1)
print(train_oversampled['label'].value_counts())
train_oversampled

# Splitting the data into training and testing sets
X_train, X_val, y_train, y_val = train_test_split(train_oversampled['features'], train_oversampled['label'],
test_size=0.2, random_state=123)

# set up models and test them
test_model = [MultinomialNB(), RandomForestClassifier(), DecisionTreeClassifier(),
        LogisticRegression(solver='saga'), SVC(), KNeighborsClassifier(),
            SGDClassifier(), XGBClassifier()]

ALL_Cases_results = pd.DataFrame()
```

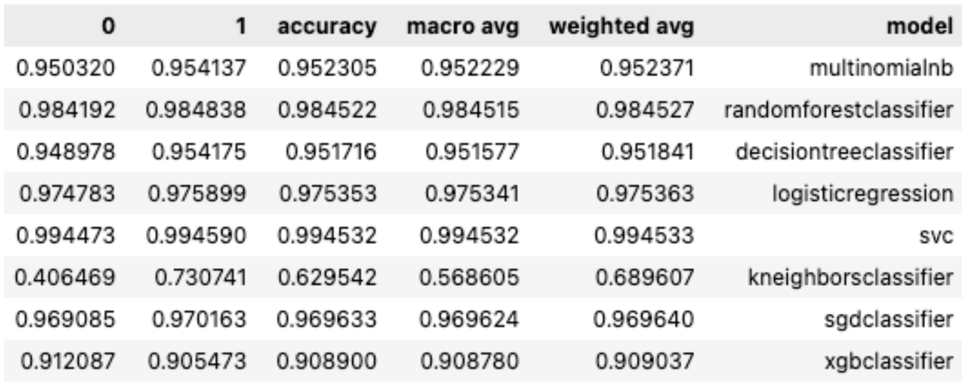
```
file = pd.DataFrame()

for val in test_model:
    # Feature extraction using TF-IDF
    # Choose a model from list
    pipe = make_pipeline(TfidfVectorizer(), val)
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_val)
    model = list(pipe.named_steps)
    result = classification_report(y_pred, y_val, output_dict=True)
    df_results = pd.DataFrame(result)
    df_results['model'] = model[1]
    file = pd.concat([file, df_results], axis=0)
    total_result6 = file

ALL_Cases_results = pd.concat([ALL_Cases_results, file[file.index == 'f1-score']], axis=0)
```

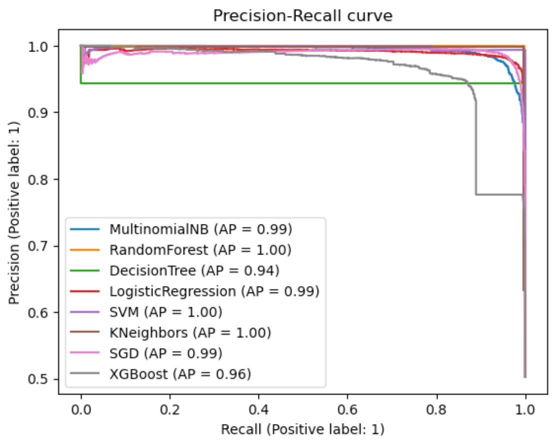
7. Model Performance Evaluation

# f1-scores for 8 models

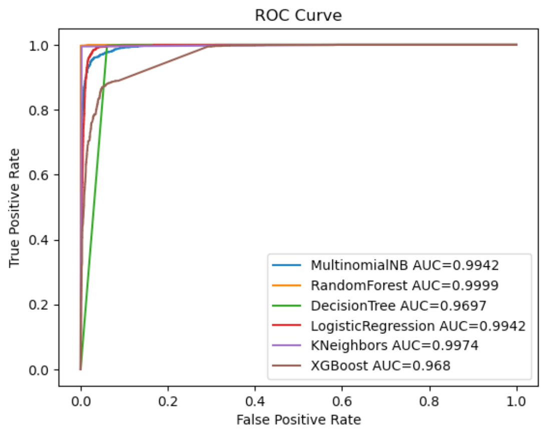


0	1	accuracy	macro avg	weighted avg	model
0.950320	0.954137	0.952305	0.952229	0.952371	multinomialnb
0.984192	0.984838	0.984522	0.984515	0.984527	randomforestclassifier
0.948978	0.954175	0.951716	0.951577	0.951841	decisiontreeclassifier
0.974783	0.975899	0.975353	0.975341	0.975363	logisticregression
0.994473	0.994590	0.994532	0.994532	0.994533	svc
0.406469	0.730741	0.629542	0.568605	0.689607	kneighborsclassifier
0.969085	0.970163	0.969633	0.969624	0.969640	sgdclassifier
0.912087	0.905473	0.908900	0.908780	0.909037	xgbclassifier

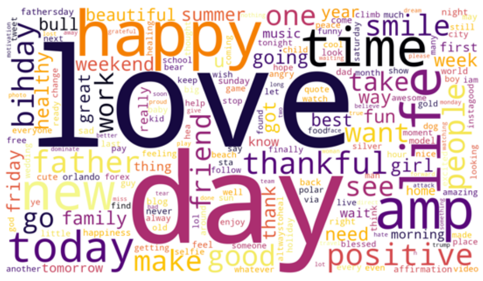
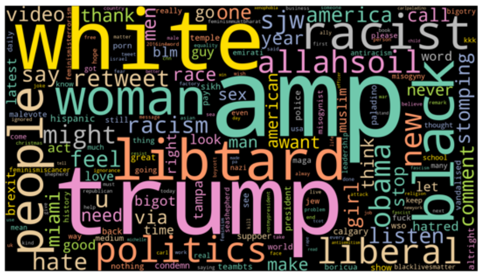
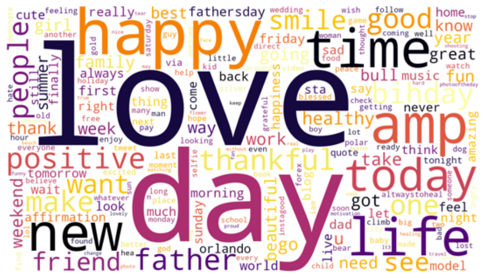
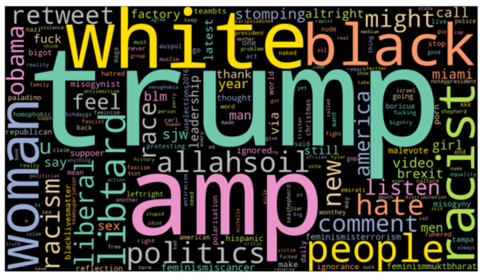
# precision-recall curves for 8 models



ROC scores and curves for 6 models



8. Model Test

	Not hate speech in tweets	Hate speech in tweets
Training Dataset Result (total 31962 tweets)	`{'love': 2766, 'day': 2735, 'happy': 1679, 'amp': 1319, 'life': 1221, 'time': 1205, 'today': 1066, 'get': 949, 'like': 945, 'positive': 932}`	`{'amp': 283, 'trump': 216, 'white': 153, 'libtard': 149, 'black': 146, 'like': 140, 'woman': 120, 'racist': 109, 'people': 106, 'politics': 97}`
		
Test Dataset Result (total 17197 tweets)	`{'love': 1523, 'day': 1417, 'happy': 937, 'amp': 708, 'time': 666, 'life': 627, 'today': 580, 'new': 533, 'get': 509, 'positive': 489}`	`{'trump': 196, 'amp': 117, 'white': 111, 'black': 92, 'woman': 91, 'racist': 72, 'people': 64, 'like': 59, 'libtard': 57, 'politics': 51}`
		

The analysis of the top words associated with hate speech in both the training and test datasets reveals a notable consistency, despite slight variations in their order of occurrence. Key terms such as 'amp,' 'like,' 'white,' 'woman,' and 'politics' consistently emerge among the top words for hate speech, underlining their significance and robustness across datasets. This consistency in the identification of specific terms implies a reliable discriminatory power of these words in recognizing hate speech. Even though the order of appearance might differ, the persistent presence of these words in the top rankings across datasets highlights their importance and contributes to the overall robustness of the model's performance in distinguishing hate speech content.

9. LSTM Model Building & Training

1) **Over sampling for Imbalanced Data**

   The training data is oversampled to address class imbalance. Class 1 instances are sampled with replacement to match the number of instances in Class 0.

2) **Split Data to Training and Test**

   The oversampled data is split into training and validation sets.

3) **Tokenize and Pad Text Data**

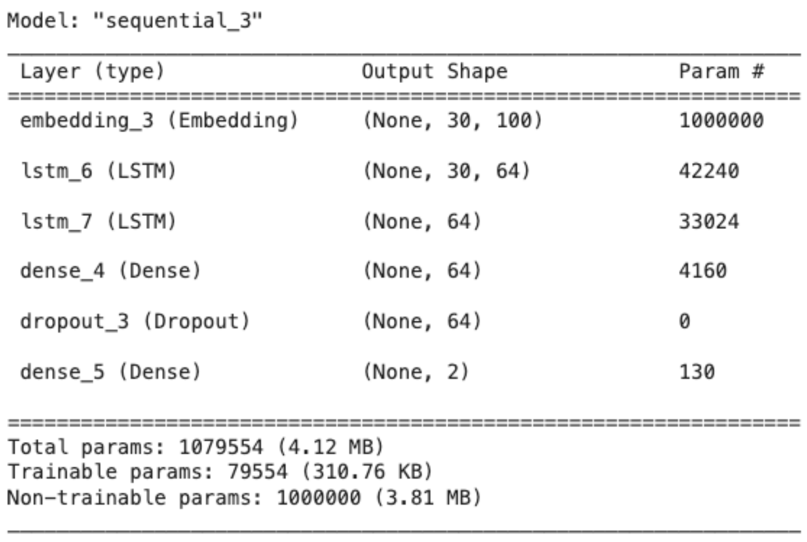
   Text data is tokenized and padded to create sequences of equal length. The Tokenizer class is used to convert text to sequences of integers, and the pad_sequences function is employed to ensure uniform sequence length.

4) **Load GloVe Embeddings**

   GloVe (Global Vectors for Word Representation) pre-trained word embeddings are loaded. These embeddings are used to create an embedding matrix for the embedding layer of the neural network.

5) **NN Model Architecture**

   A sequential neural network model (below table) is defined using Keras. The model includes an embedding layer with pre-trained word embeddings, two LSTM layers, a dense layer, dropout layer, and the output layer. The model is compiled using the Adam optimizer and sparse categorical crossentropy loss.



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 30, 100)           1000000

 lstm_6 (LSTM)               (None, 30, 64)            42240

 lstm_7 (LSTM)               (None, 64)                33024

 dense_4 (Dense)             (None, 64)                4160

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 2)                 130

=================================================================
Total params: 1079554 (4.12 MB)
Trainable params: 79554 (310.76 KB)
Non-trainable params: 1000000 (3.81 MB)
_________________________________________________________________
```

6) Train and Evaluate Model

The model is trained on the training data and evaluated on the validation set. The training process includes ten epochs, and the validation accuracy is printed at the end of training.

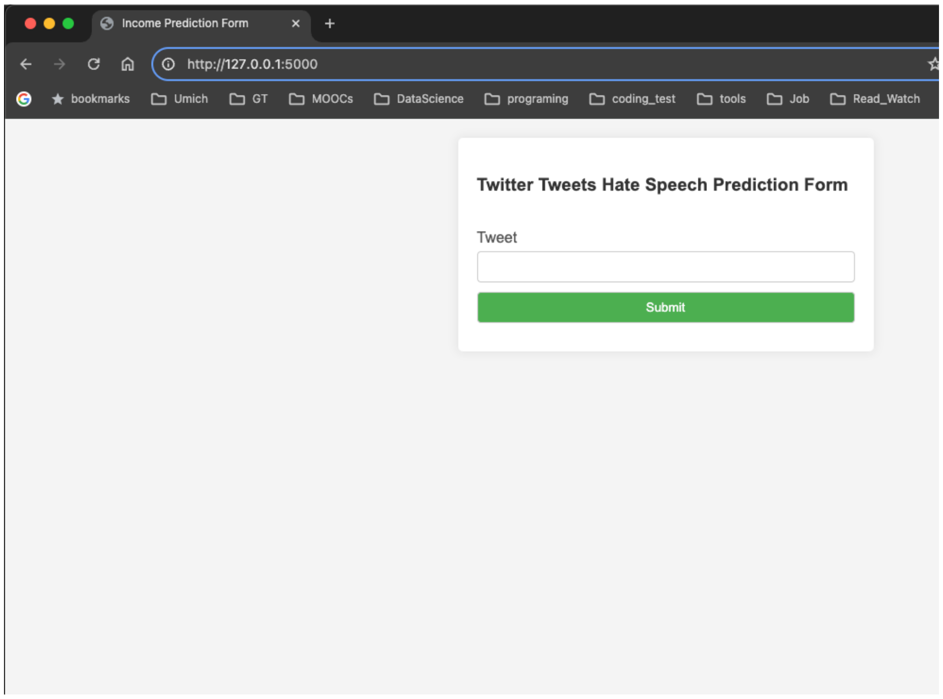
```
# Train and Evaluate model
model.fit(train_padded, train_df['label'], epochs=10,
validation_data=(val_padded, val_df['label']))
val_loss, val_accuracy = model.evaluate(val_padded, val_df['label'])
print(f'Validation Accuracy: {val_accuracy * 100:.2f}%')
```

```
Epoch 1/10
1486/1486 [==============================] - 57s 36ms/step - loss:
0.3129 - accuracy: 0.8761 - val_loss: 0.1990 - val_accuracy: 0.9224
Epoch 2/10
1486/1486 [==============================] - 52s 35ms/step - loss:
0.1740 - accuracy: 0.9397 - val_loss: 0.1507 - val_accuracy: 0.9418
Epoch 3/10
1486/1486 [==============================] - 50s 34ms/step - loss:
0.1100 - accuracy: 0.9657 - val_loss: 0.0895 - val_accuracy: 0.9688
Epoch 4/10
1486/1486 [==============================] - 52s 35ms/step - loss:
0.0806 - accuracy: 0.9771 - val_loss: 0.0809 - val_accuracy: 0.9759
Epoch 5/10
…
Epoch 10/10
1486/1486 [==============================] - 51s 35ms/step - loss:
0.0265 - accuracy: 0.9928 - val_loss: 0.0569 - val_accuracy: 0.9862
372/372 [==============================] - 4s 12ms/step - loss: 0.0569
- accuracy: 0.9862
Validation Accuracy: 98.62%
```

The neural network model was trained and evaluated on the provided dataset with a total of 31,962 tweets. The model comprises an embedding layer with pre-trained GloVe word embeddings, followed by two Long Short-Term Memory (LSTM) layers, a dense layer with ReLU activation, and a dropout layer to prevent overfitting. During training, the model exhibited consistent improvement in performance across epochs, achieving a final validation accuracy of 98.62% after 10 epochs. The training process included minimizing the sparse categorical cross entropy loss function using the Adam optimizer. The evaluation results demonstrate the robustness of the model in accurately classifying hate speech, with a validation accuracy exceeding 98%. This high accuracy suggests the effectiveness of the neural network in learning and generalizing patterns associated with hate speech within the training dataset.

9. Model Deployment

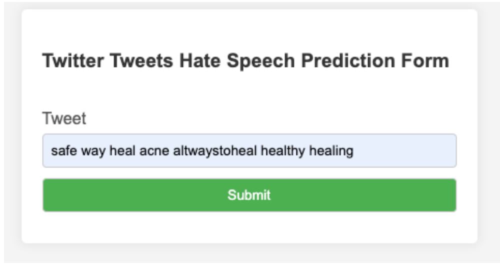
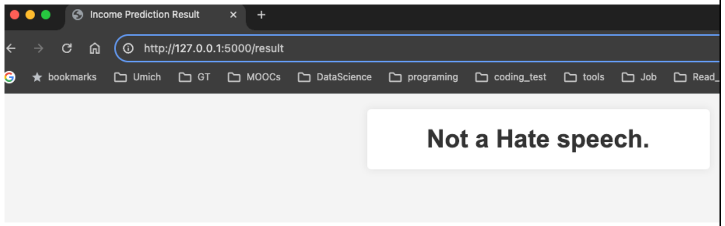
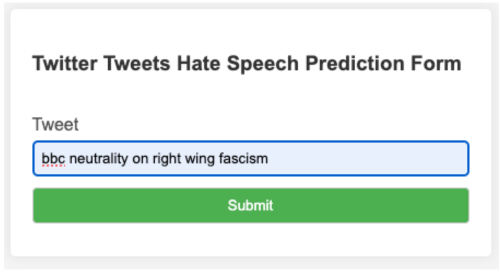
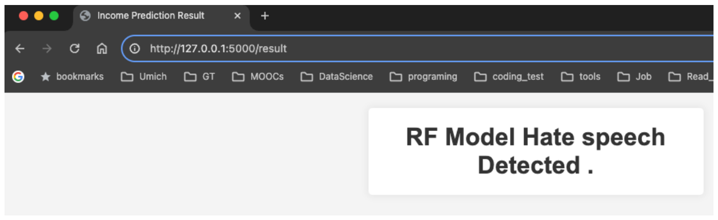
Model deployment using Flask



Twitter Tweets Hate Speech Prediction Form

Tweet

10. Model Inference

 Twitter Tweets Hate Speech Prediction Form Tweet: safe way heal acne altwaystoheal healthy healing 	Result  Not a Hate speech.
 Twitter Tweets Hate Speech Prediction Form Tweet: bbc neutrality on right wing fascism 	Result  RF Model Hate speech Detected .

11. Conclusion and limitation

In this project, the Hate Speech Detection has successfully addressed the pressing issue of identifying and mitigating hate speech on social media, particularly on Twitter.

The initial exploratory data analysis highlighted the significance of preprocessing steps in handling noise and ensuring consistency in the representation of tweets. Feature engineering, oversampling for class imbalance, and thoughtful selection of machine learning models contributed to robust model training and evaluation.

The comprehensive approach encompassed data preprocessing, multiple model selections, and the development of a powerful LSTM model, showcasing the efficacy of advanced techniques in natural language processing. The inclusion of Transformer-based architectures, such as the LSTM model, demonstrated the project's adaptability to state-of-the-art techniques. The model exhibited exceptional performance, achieving a validation accuracy of 98.62%, underscoring its ability to accurately classify hate speech content.

The project further extended its impact through model deployment using Flask, making the Hate Speech Detection model ready for integration with online platforms. Real-time inference results reaffirmed the model's effectiveness in classifying new tweets, reinforcing its potential for active hate speech detection.

In conclusion, while the Hate Speech Detection project has achieved commendable success in developing an accurate LSTM model and addressing hate speech on Twitter, certain limitations and challenges were encountered during deployment. Two significant issues proved challenging to overcome.

Firstly, continuous errors occurred during the deployment of the LSTM model through Flask. It was identified that these errors were attributed to difficulties in importing TensorFlow within the local environment. This issue was exacerbated by repeated installations and uninstallations of

TensorFlow due to concurrent involvement in multiple recent projects. Despite extensive troubleshooting, the deployment through Flask was not successful.

Secondly, attempts to deploy the models on PythonAnywhere.com were hindered by the failure to open the console and difficulties in reading the models, resulting in an unsuccessful deployment. It is hypothesized that exploring alternative platforms for deployment might provide a viable solution to these challenges.

These deployment-related limitations emphasize the importance of considering the intricacies of the local environment and exploring diverse deployment platforms. Future iterations of the project may benefit from a more controlled and isolated environment during model development, and alternative platforms with robust support for TensorFlow deployment should be explored to ensure successful model integration and real-time hate speech detection.

Despite these challenges, the Hate Speech Detection project represents a significant step forward in leveraging advanced natural language processing techniques to combat hate speech online. The successful development of the LSTM model, coupled with insightful exploratory data analysis and model evaluation, provides a strong foundation for further refinement and future enhancements.

12. GitHub Repo link

https://github.com/syoungk7/Hate_Speech_Detection/data_preprocessing/