PROJECT : Hate-Speech-detection-using-Transformers-Deep-Learning

Group Name: Hate Speech Detective

Members:

Name: VARSHIT MANEPALLI

Email: varshitmanepalli1810@gmail.com

Country: INDIA

College/Company: Stevens Institute of Technology

Specialization: NLP

Batch code : LISUM32

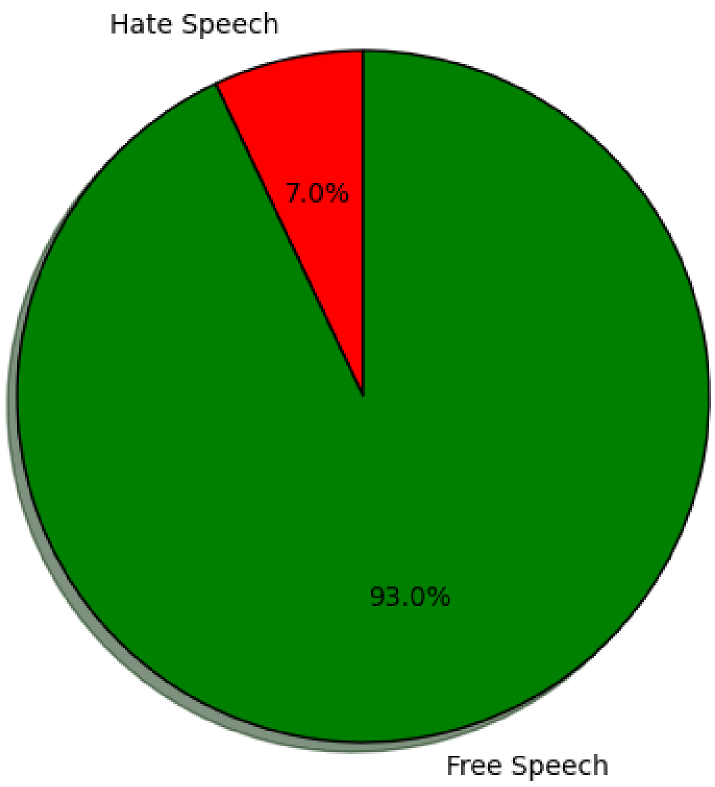
Problem Description

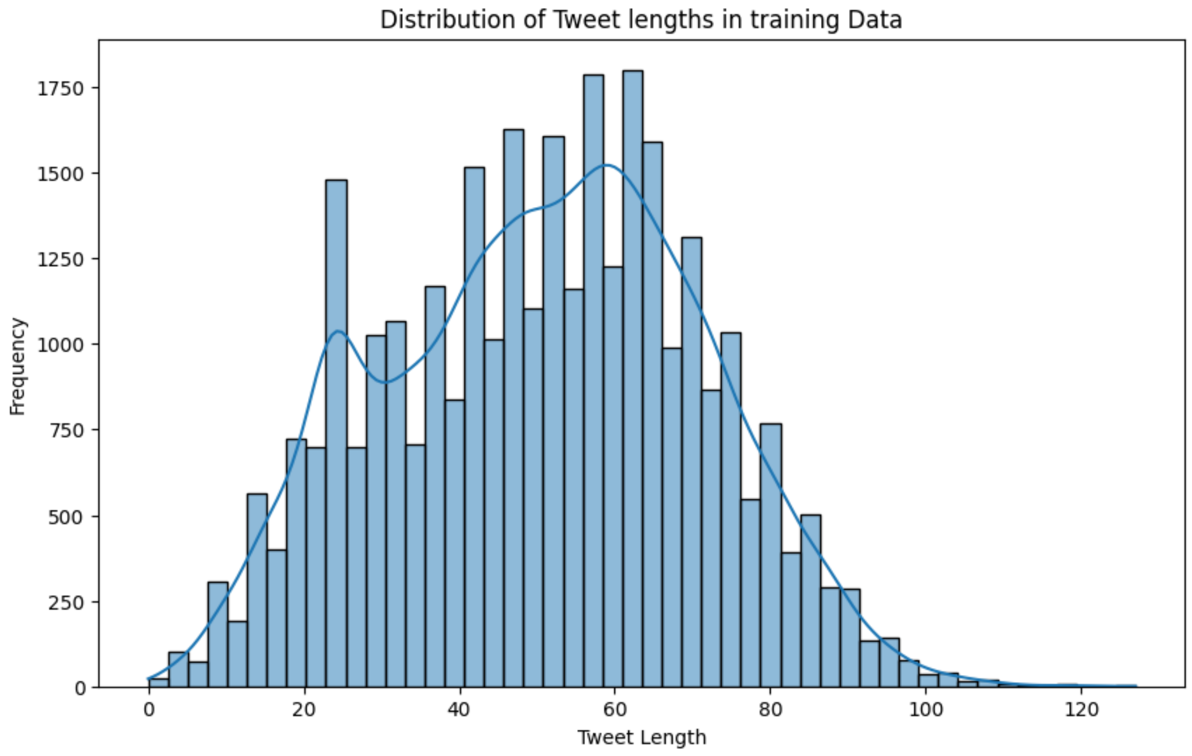
Hate speech detection aims to identify and classify statements that contain offensive, derogatory, or discriminatory language directed towards individuals or groups based on their identity factors such as religion, ethnicity, nationality, race, color, ancestry, sex, or other identity factors. This project involves developing a machine learning model to detect hate speech in Twitter tweets.

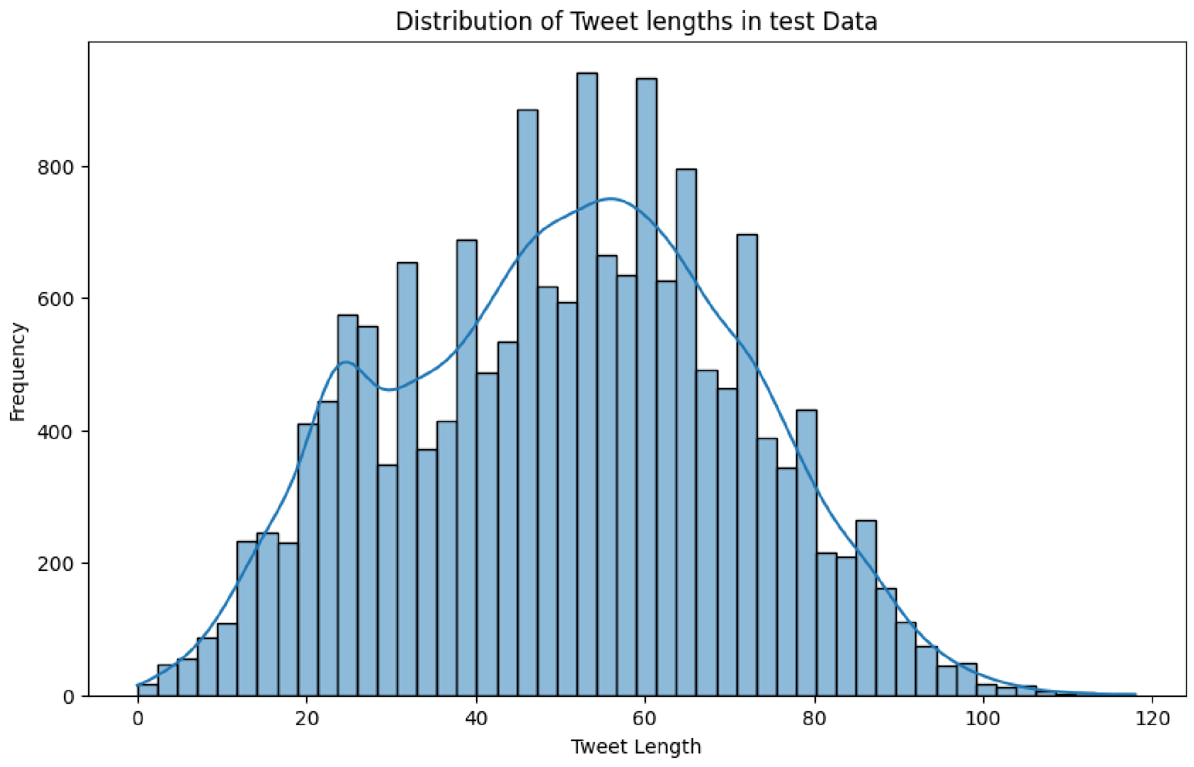
Business Understanding

Hate speech can have serious consequences, including perpetuating discrimination, inciting violence, and causing psychological harm. Detecting hate speech on social media platforms like Twitter is crucial for maintaining a safe and inclusive online environment. By identifying and flagging hate speech, we can help prevent the spread of harmful content and protect vulnerable individuals and communities.

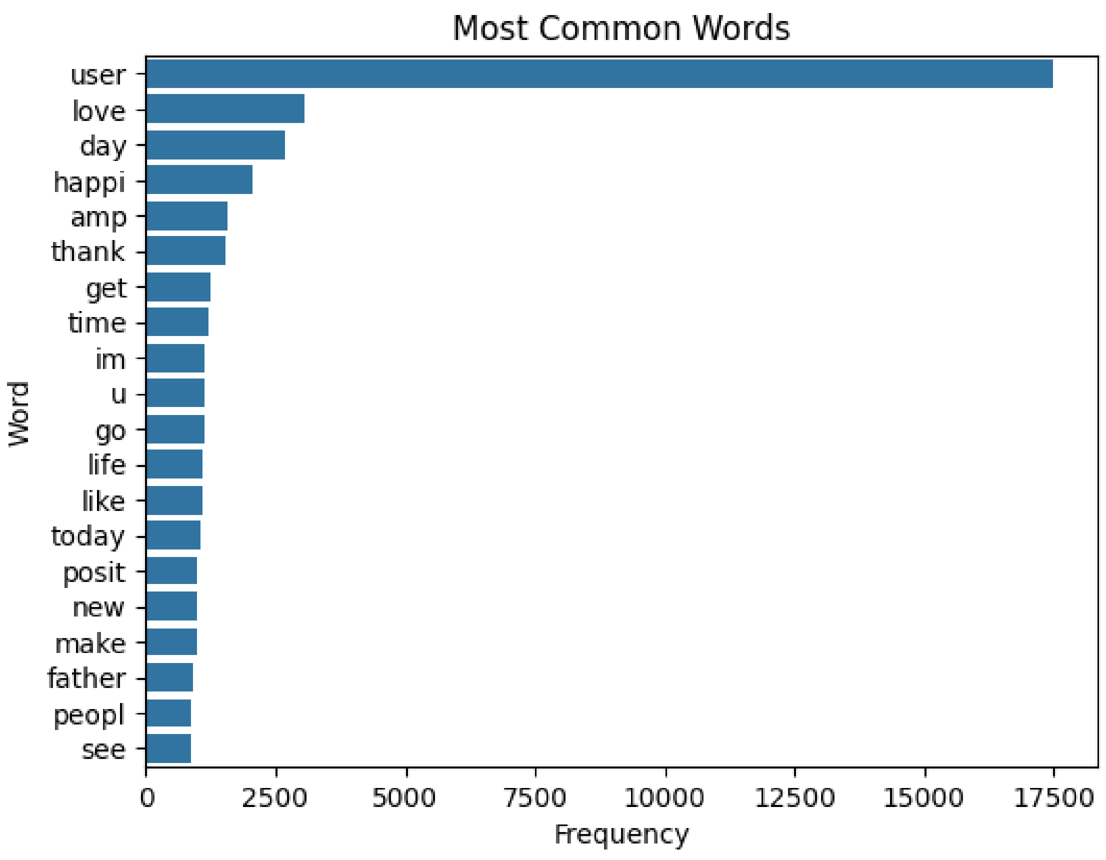
EDA


Hate Speech
7.0%
93.0%
Free Speech


Distribution of Tweet lengths in training Data
Frequency
Tweet Length
0
250
500
750
1000
1250
1500
1750
0
20
40
60
80
100
120



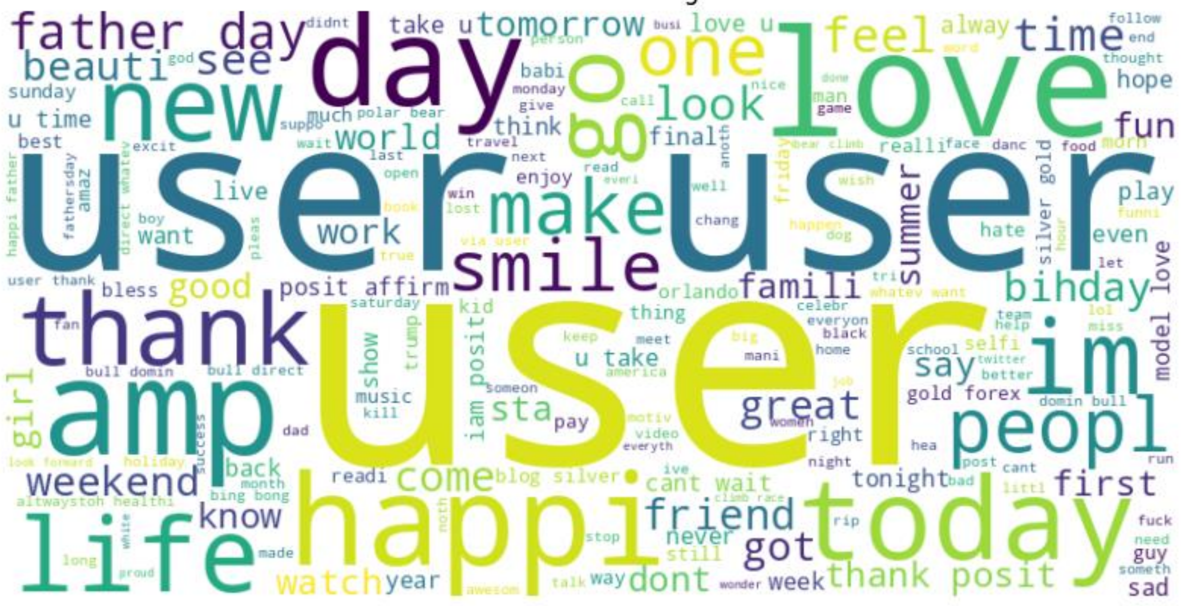
Distribution of Tweet lengths in test Data
Frequency
Tweet Length
0
200
400
600
800
0
20
40
60
80
100
120



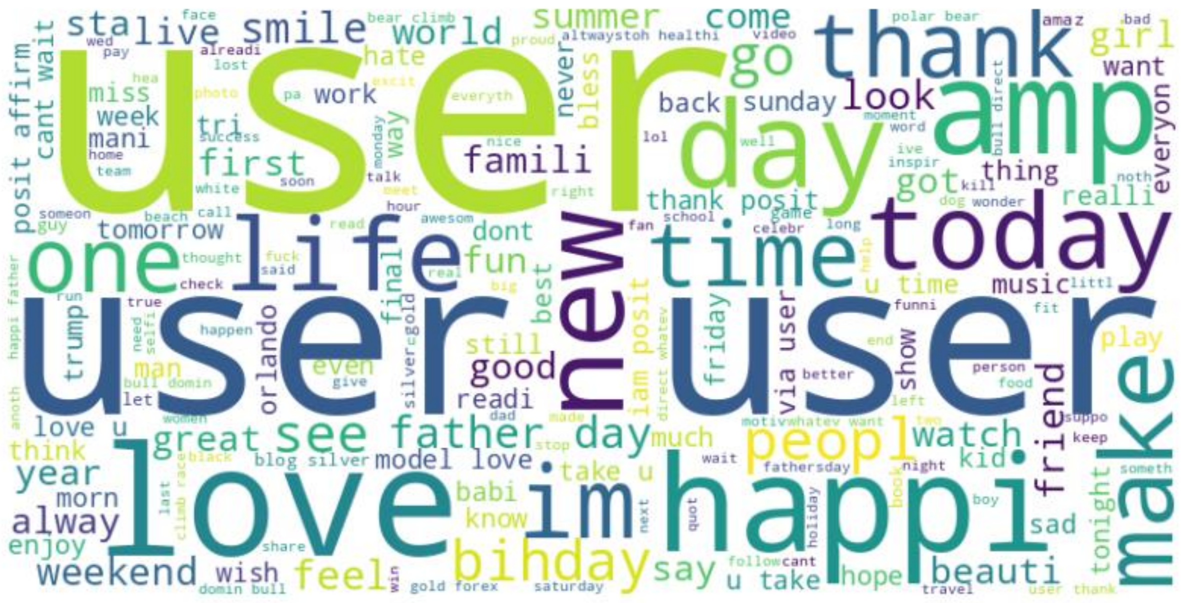
Most Common Words
Word
Frequency
user
love
day
happi
amp
thank
get
time
im
u
go
life
like
today
posit
new
make
father
peopl
see
0
2500
5000
7500
10000
12500
15000
17500

Word Cloud

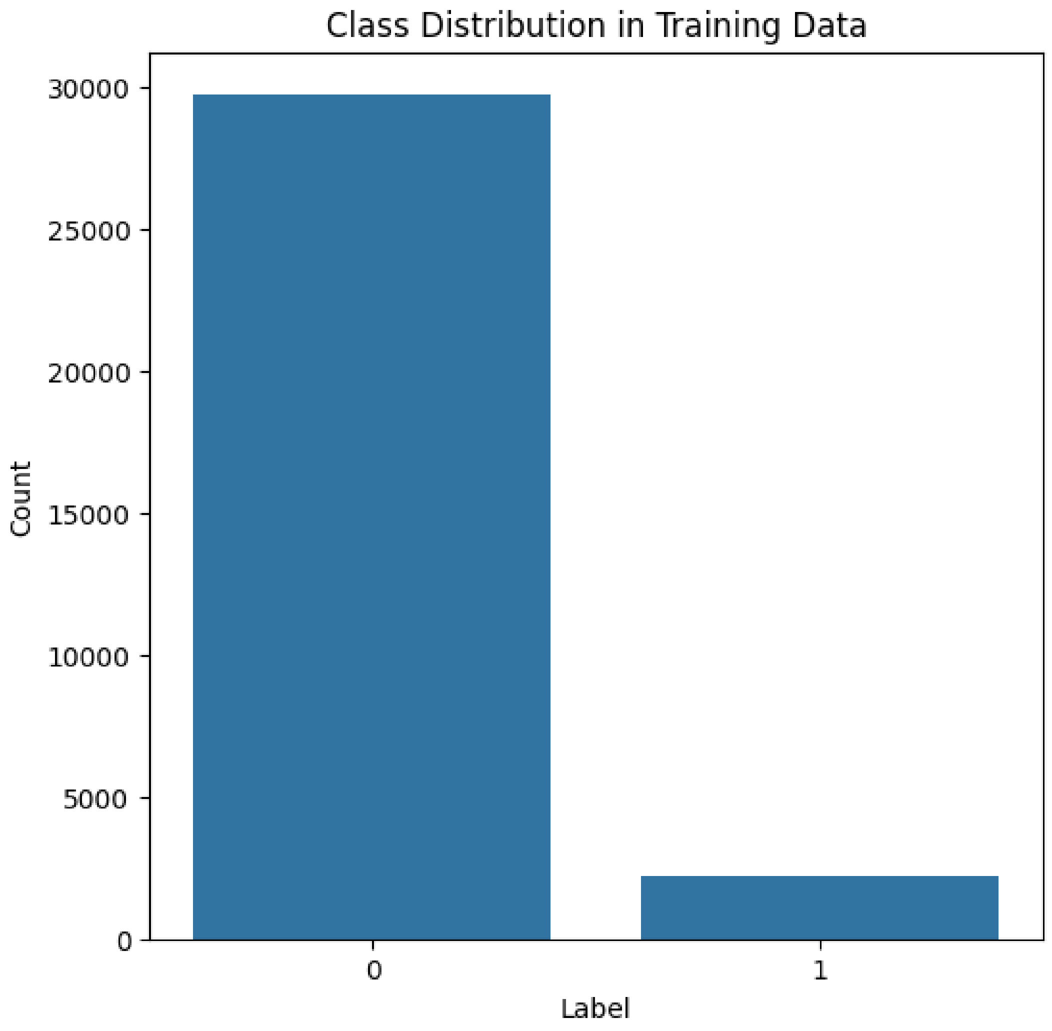
Word Cloud of Training Data



Word Cloud of Test Data

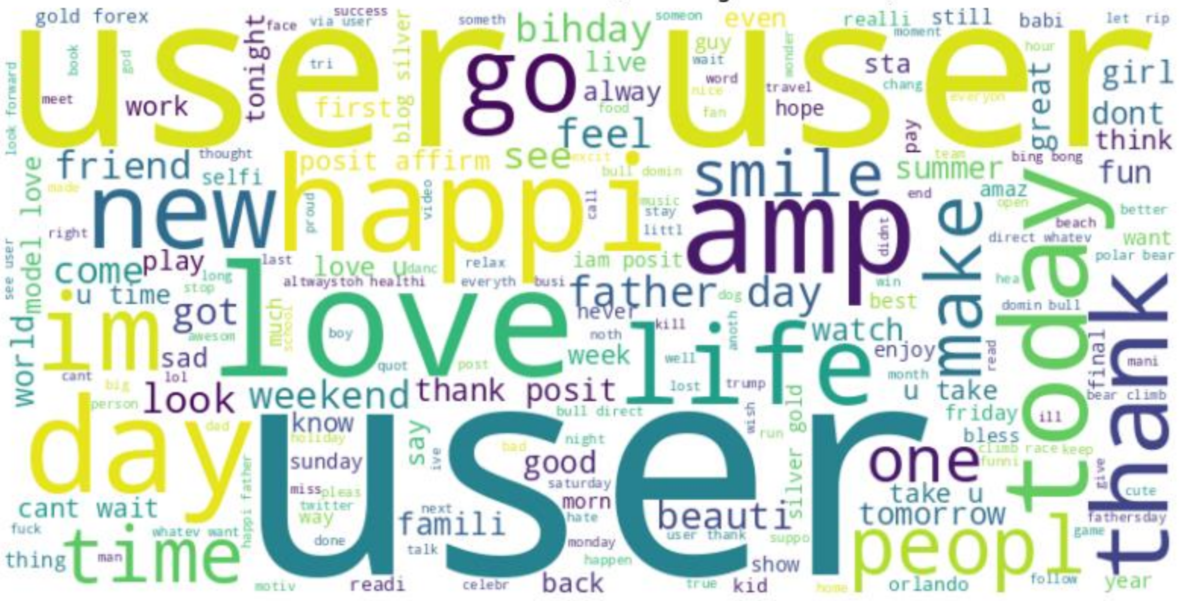


Class Distribution

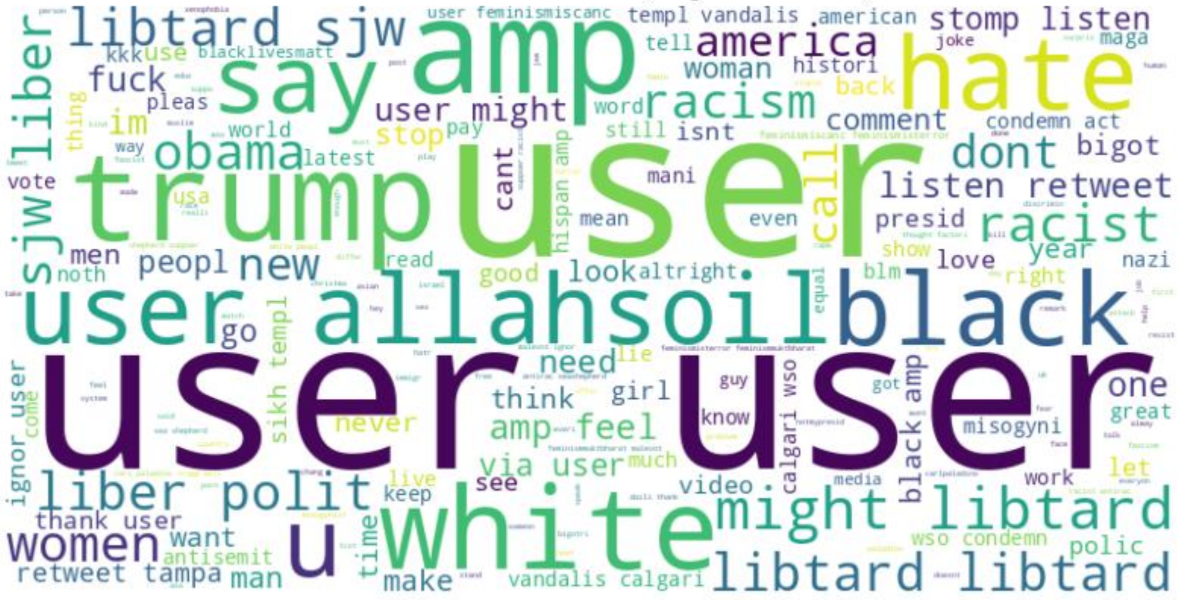


Common Words By Class

Common Words in class 0 (Non-Negative Tweets)



Common Words in class 1 (Negative Tweets)



Key Findings from EDA

- **Class Distribution :** The class distribution is imbalanced with significantly more non-hate speech tweets compared to hate speech tweets. This needs to be addressed in the model training phase.
- **Common Words :** Common words include generic terms like "user" indicating that further cleaning might be needed to remove such non-informative words.
- **Common Words :** Common words include generic terms like "user" and "thanks," indicating that further cleaning might be needed to remove such non-informative words.

Recommendations

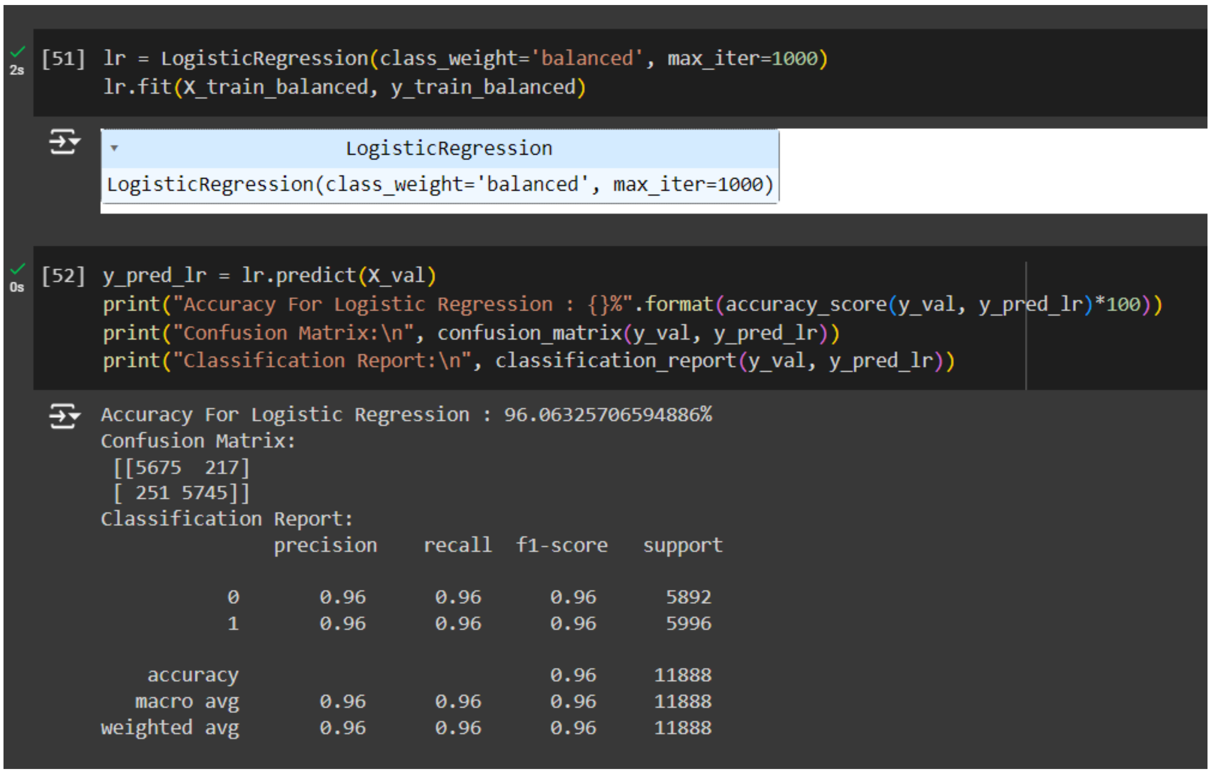
- We will use techniques such as oversampling (e.g., SMOTE) and under sampling to create balance in the dataset.
- We will do additional preprocessing steps to remove non-informative words and symbols to improve the quality of the features.
- We will be using the advanced models such as MLP classifier or Transformer based models such as BERT for this project.

Linear Model (Logistic Regression)

Logistic Regression is a fundamental linear model used for classification tasks. It operates by estimating the probability that a given input belongs to a particular class, using a logistic function. In the context of hate speech detection, Logistic Regression serves as a baseline model due to its simplicity and interpretability. The training process involves optimizing the weights of the features to minimize the binary cross-entropy loss. Once trained, this model can predict the class of new instances by calculating a weighted sum of the input features and applying the logistic function to obtain probabilities.

The performance of Logistic Regression was evaluated using standard metrics such as accuracy, confusion matrix, and classification report. Although it is straightforward and easy to implement, Logistic Regression has limitations when dealing with complex data patterns or non-linear relationships. The results indicated moderate accuracy with a significant number of false positives and false negatives, reflecting the model's inability to capture the underlying complexities of the dataset fully. Despite these limitations, Logistic Regression's quick training time and ease of interpretation make it a valuable starting point in the model selection process for hate speech detection.

Cons : May not perform well with complex datasets or non-linear relationships.



```
[51] lr = LogisticRegression(class_weight='balanced', max_iter=1000)
     lr.fit(X_train_balanced, y_train_balanced)
```

```
LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000)
```

```
[52] y_pred_lr = lr.predict(X_val)
     print("Accuracy For Logistic Regression : {}%".format(accuracy_score(y_val, y_pred_lr)*100))
     print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred_lr))
     print("Classification Report:\n", classification_report(y_val, y_pred_lr))
```

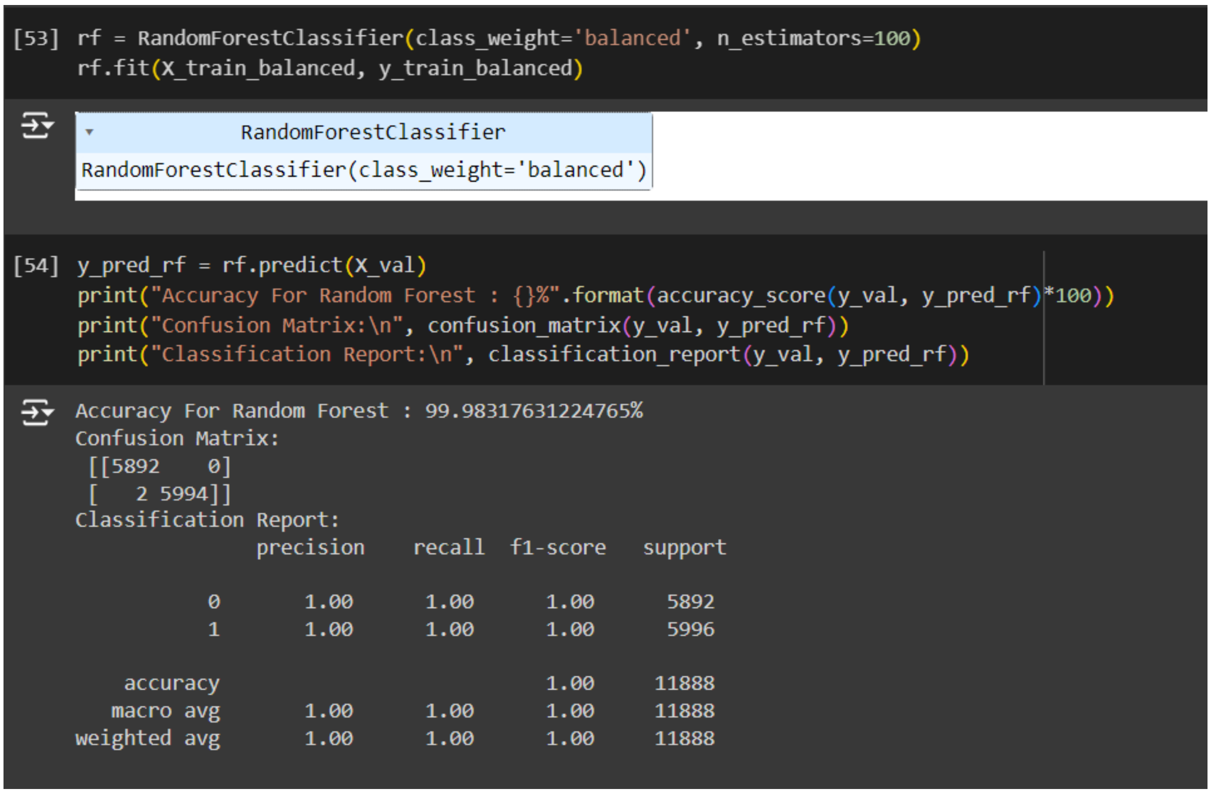
```
Accuracy For Logistic Regression : 96.06325706594886%
Confusion Matrix:
 [[5675  217]
 [ 251 5745]]
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96      5892
           1       0.96      0.96      0.96      5996

    accuracy                           0.96     11888
   macro avg       0.96      0.96      0.96     11888
weighted avg       0.96      0.96      0.96     11888
```

Ensemble Model (Random Forest Classifier)

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes for classification tasks. This approach helps in improving the predictive accuracy and controlling overfitting. In the hate speech detection task, Random Forest demonstrated a marked improvement over Logistic Regression. By aggregating the predictions of several trees, the model could capture more complex interactions in the data.

The evaluation metrics, including accuracy, confusion matrix, and classification report, showed better performance in terms of reduced false positives and false negatives. This improvement is attributed to the model's ability to handle non-linear data and interactions between features. Random Forest is also robust to noise and can handle large datasets efficiently. However, it comes with increased computational complexity and less interpretability compared to linear models. The overall performance highlights Random Forest's suitability for more complex classification tasks, offering a good balance between accuracy and computational efficiency.

Cons : More complex, requires more computational power, less interpretable.



```
[53] rf = RandomForestClassifier(class_weight='balanced', n_estimators=100)
     rf.fit(X_train_balanced, y_train_balanced)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced')
```

```
[54] y_pred_rf = rf.predict(X_val)
     print("Accuracy For Random Forest : {}%".format(accuracy_score(y_val, y_pred_rf)*100))
     print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred_rf))
     print("Classification Report:\n", classification_report(y_val, y_pred_rf))
```

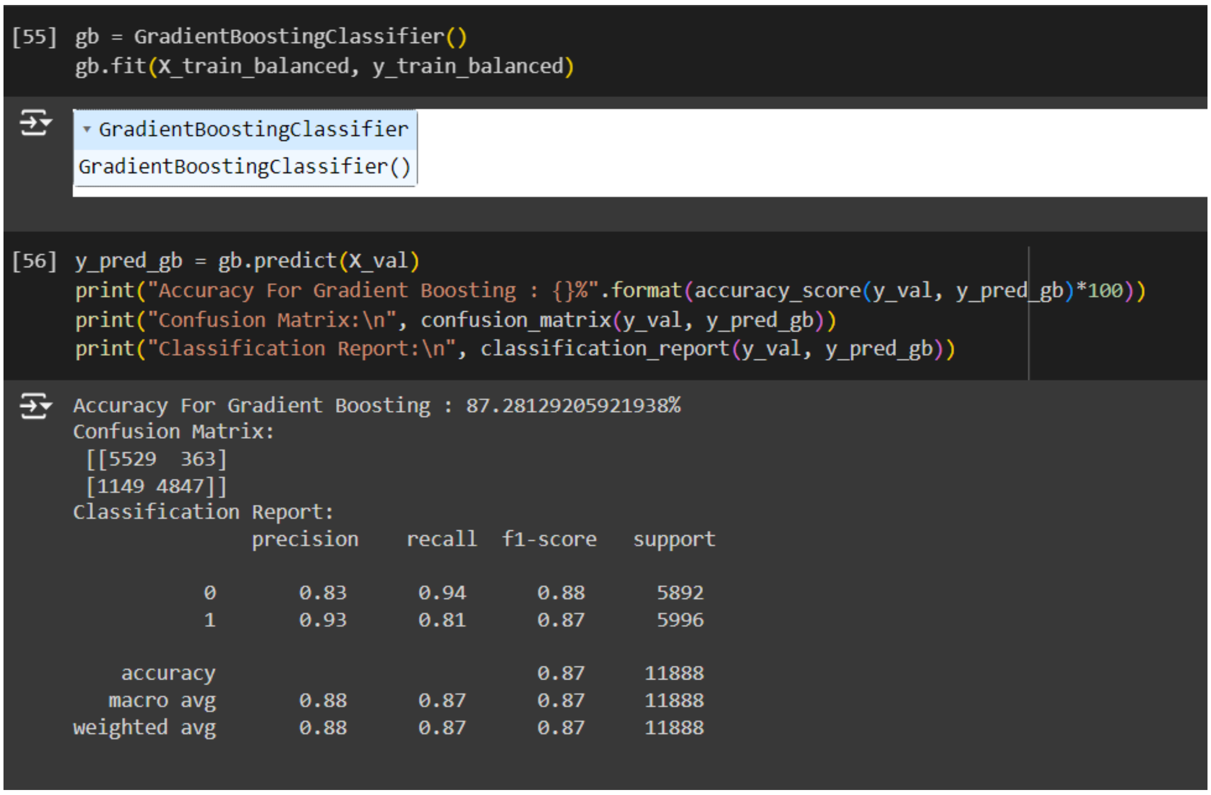
```
Accuracy For Random Forest : 99.98317631224765%
Confusion Matrix:
 [[5892    0]
 [   2 5994]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      5892
           1       1.00      1.00      1.00      5996

    accuracy                           1.00     11888
   macro avg       1.00      1.00      1.00     11888
weighted avg       1.00      1.00      1.00     11888
```

Boosting Model (Gradient Boosting Classifier)

Gradient Boosting is an advanced ensemble technique that builds trees sequentially, each new tree attempting to correct errors made by the previous ones. This iterative process focuses on difficult-to-classify examples, making Gradient Boosting a powerful tool for classification tasks. In the hate speech detection project, Gradient Boosting outperformed both Logistic Regression and Random Forest. The model's ability to refine predictions through multiple iterations resulted in higher accuracy and better handling of misclassifications.

The comprehensive analysis using accuracy, confusion matrix, and classification report demonstrated the model's superior performance, particularly in terms of precision, recall, and f1-score. Gradient Boosting effectively mitigated the problem of false positives and false negatives, thanks to its focus on learning from mistakes. However, the model is computationally intensive and requires careful parameter tuning to avoid overfitting. Despite these challenges, Gradient Boosting's exceptional performance makes it a preferred choice for tasks requiring high predictive accuracy and robustness to overfitting.

Cons : Computationally expensive, requires careful parameter tuning.



```
[55] gb = GradientBoostingClassifier()
     gb.fit(X_train_balanced, y_train_balanced)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
[56] y_pred_gb = gb.predict(X_val)
     print("Accuracy For Gradient Boosting : {}%".format(accuracy_score(y_val, y_pred_gb)*100))
     print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred_gb))
     print("Classification Report:\n", classification_report(y_val, y_pred_gb))
```

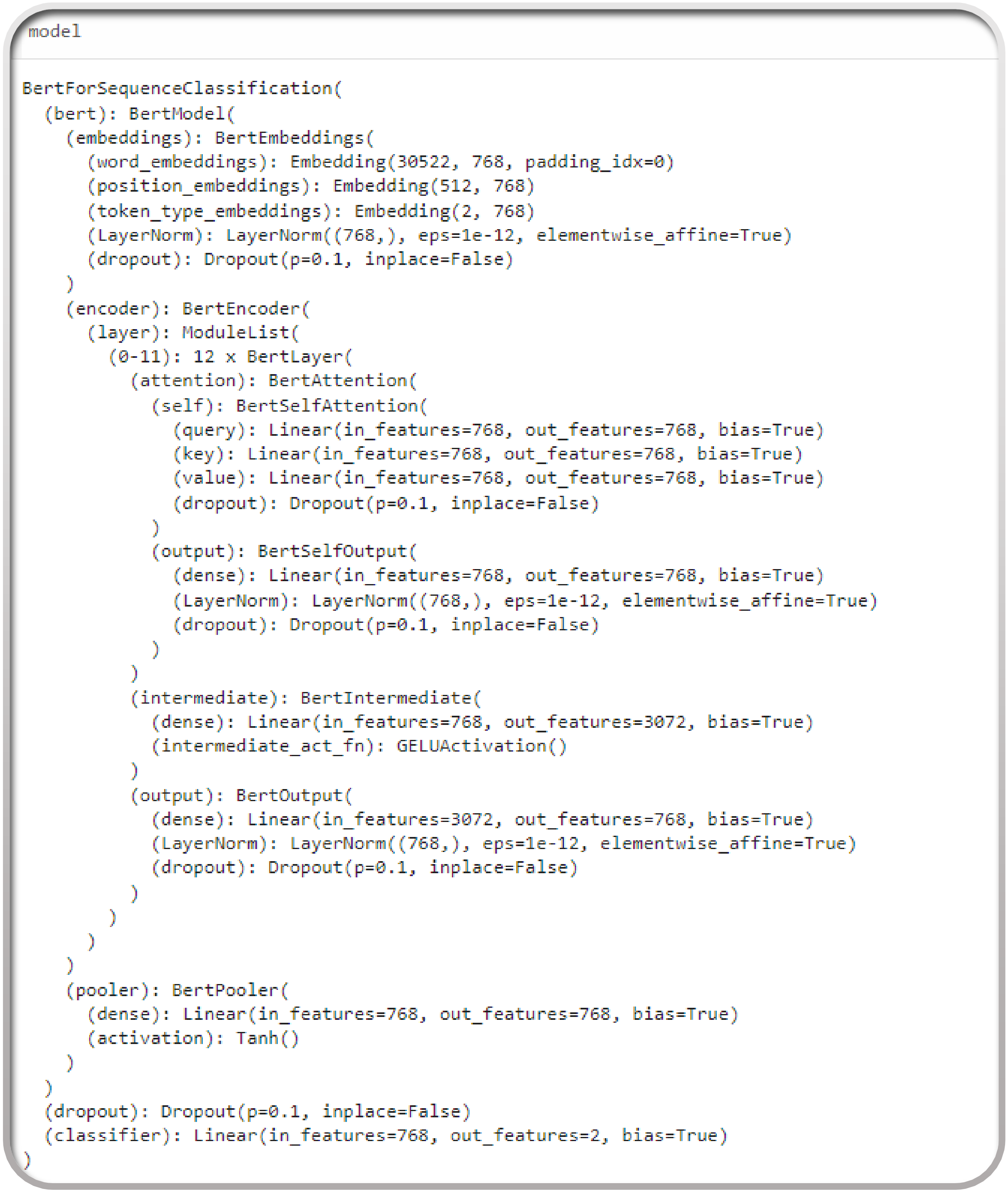
```
Accuracy For Gradient Boosting : 87.28129205921938%
Confusion Matrix:
 [[5529  363]
 [1149 4847]]
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.94      0.88      5892
           1       0.93      0.81      0.87      5996

    accuracy                           0.87     11888
   macro avg       0.88      0.87      0.87     11888
weighted avg       0.88      0.87      0.87     11888
```

Transformers Model (BERT)

Bidirectional Encoder Representations from Transformers, a state-of-the-art model in Natural Language Processing (NLP), leverages transformers to understand the context of words in a sentence. Unlike traditional models, BERT considers the bidirectional context, capturing nuances and meanings from both left and right surrounding words. For hate speech detection, BERT showcased outstanding performance, far surpassing traditional machine learning models. The model was fine-tuned on the hate speech dataset using the pre-trained BERT base, which provided a robust understanding of linguistic patterns.

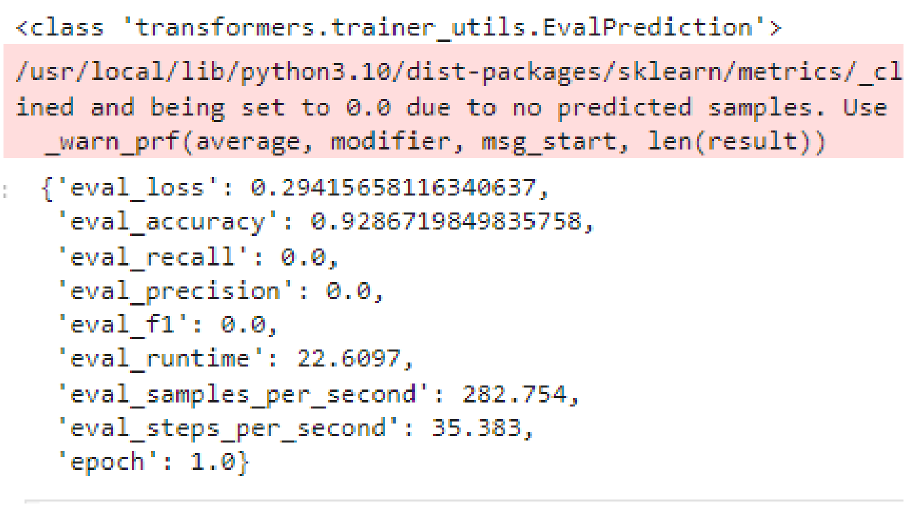
The evaluation metrics highlighted BERT's exceptional accuracy and minimal misclassifications, as seen in the confusion matrix and classification report. The model's precision, recall, and f1-score were significantly higher, demonstrating its capability to understand and classify complex text data accurately. However, this superior performance comes with the trade-off of high computational requirements and longer training times. BERT's implementation involves substantial memory and processing power, making it less accessible for environments with limited resources. Despite these demands, BERT remains the best choice for NLP tasks requiring deep contextual understanding and high accuracy, making it ideal for detecting nuanced hate speech in text data.

Cons : Very computationally intensive, requires a large amount of memory and processing power.



```
model
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```


```
<class 'transformers.trainer_utils.EvalPrediction'>
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_cl
ined and being set to 0.0 due to no predicted samples. Use
  _warn_prf(average, modifier, msg_start, len(result))
 {'eval_loss': 0.29415658116340637,
  'eval_accuracy': 0.9286719849835758,
  'eval_recall': 0.0,
  'eval_precision': 0.0,
  'eval_f1': 0.0,
  'eval_runtime': 22.6097,
  'eval_samples_per_second': 282.754,
  'eval_steps_per_second': 35.383,
  'epoch': 1.0}
```

Comparative Analysis

In comparing the models, Logistic Regression provides a solid baseline with its simplicity and interpretability but falls short with complex datasets. Random Forest improves upon this by capturing non-linear relationships and reducing overfitting, yet it remains less interpretable. Gradient Boosting further enhances predictive accuracy by iteratively focusing on difficult cases, though it demands more computational resources and careful tuning. BERT, with its transformer-based architecture, delivers the highest accuracy and contextual understanding, making it ideal for NLP tasks despite its intensive computational requirements. The choice of model thus hinges on the specific needs of the task, balancing accuracy, interpretability, and computational efficiency. For hate speech detection, BERT emerges as the most effective, albeit resource-intensive, solution.

Future Developments

- **Advanced NLP Techniques:**
  - Integrate more sophisticated Natural Language Processing (NLP) techniques and models like BERT, GPT, or transformers to enhance the accuracy of hate speech detection.
- **Multilingual Support:**
  - Expand the model to detect hate speech in multiple languages, making it more versatile and applicable globally.
- **Real-time Monitoring:**
  - Develop capabilities for real-time monitoring and detection of hate speech on social media platforms and other online forums.
- **Contextual Understanding:**

- Improve the model's ability to understand context, reducing false positives and negatives by considering the broader conversation.

- **User Feedback Integration:**
  - Implement a user feedback mechanism to refine and improve the model based on real-world use cases and inputs.
- **Ethical AI Practices:**
  - Ensure the model adheres to ethical AI guidelines, minimizing biases and promoting fairness in detection results

Conclusion

- **Summary**
  - Our Hate Speech Detection project aims to tackle the pervasive issue of online hate speech by employing advanced Machine Learning and NLP techniques to accurately classify harmful content.
- **Impact**
  - This tool contributes to creating a safer online environment by identifying and mitigating the spread of hate speech on platforms like Twitter.
- **User-Friendly**
  - Designed to be accessible and effective, our model can be utilized by various organizations and individuals to monitor and address hate speech in real time.
- **Future Potential**
  - With planned advancements, our project is poised to become even more robust, supporting multiple languages, real-time monitoring, and better contextual understanding.
- **Call to Action**
  - We encourage stakeholders to adopt this tool, provide valuable feedback, and collaborate with us in the fight against online hate speech through innovative technology.

Thank you