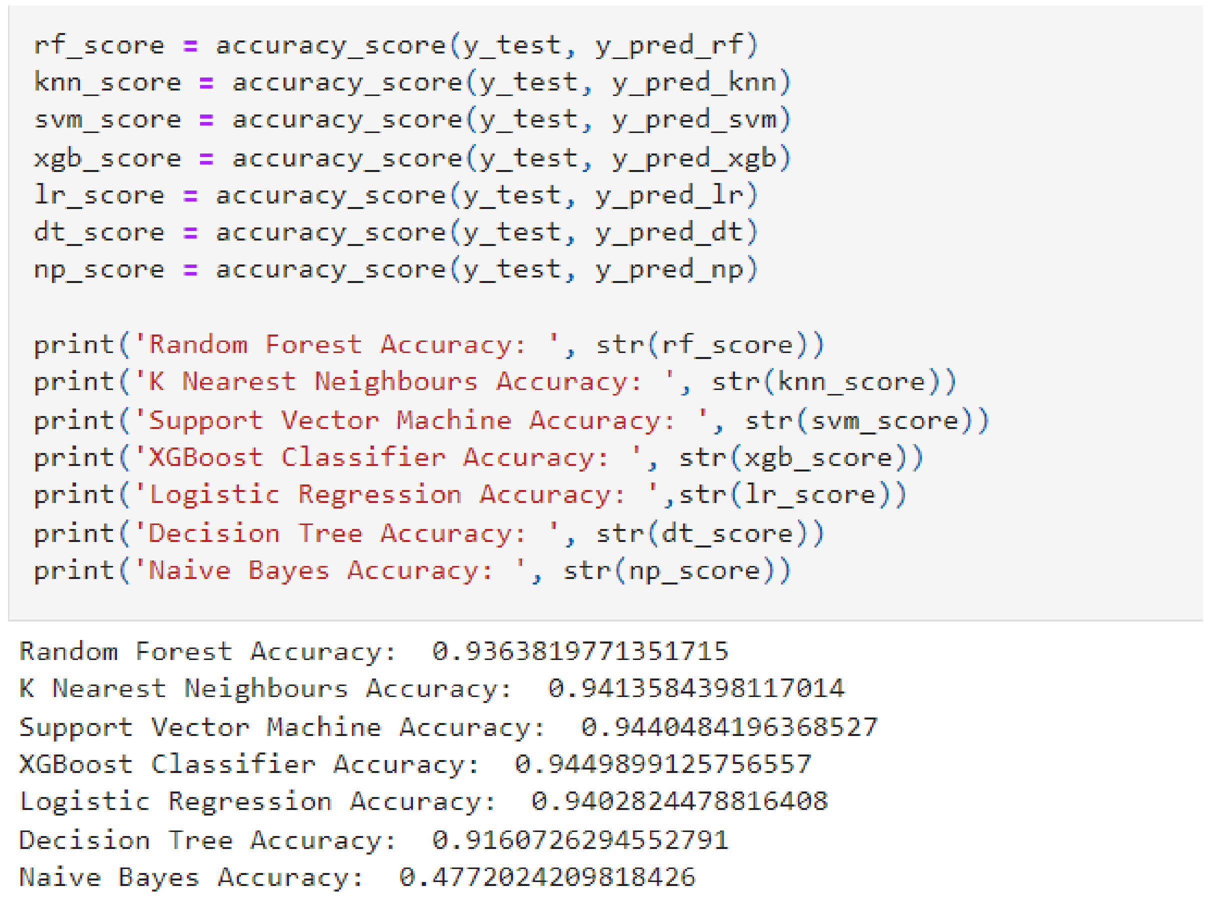
DL FINAL REPORT

Tabulation of the accuracy of different models

Model 1: XGBoost based on Lime Algorithm



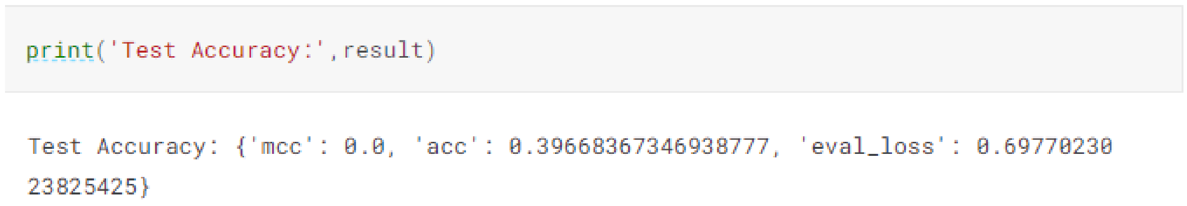
```
rf_score = accuracy_score(y_test, y_pred_rf)
knn_score = accuracy_score(y_test, y_pred_knn)
svm_score = accuracy_score(y_test, y_pred_svm)
xgb_score = accuracy_score(y_test, y_pred_xgb)
lr_score = accuracy_score(y_test, y_pred_lr)
dt_score = accuracy_score(y_test, y_pred_dt)
np_score = accuracy_score(y_test, y_pred_np)

print('Random Forest Accuracy: ', str(rf_score))
print('K Nearest Neighbours Accuracy: ', str(knn_score))
print('Support Vector Machine Accuracy: ', str(svm_score))
print('XGBoost Classifier Accuracy: ', str(xgb_score))
print('Logistic Regression Accuracy: ',str(lr_score))
print('Decision Tree Accuracy: ', str(dt_score))
print('Naive Bayes Accuracy: ', str(np_score))
```

```
Random Forest Accuracy:  0.9363819771351715
K Nearest Neighbours Accuracy:  0.9413584398117014
Support Vector Machine Accuracy:  0.9440484196368527
XGBoost Classifier Accuracy:  0.9449899125756557
Logistic Regression Accuracy:  0.9402824478816408
Decision Tree Accuracy:  0.9160726294552791
Naive Bayes Accuracy:  0.4772024209818426
```

XGBoost with count vectorizer: accuracy 94.5%

Model 2: BERT



```
print('Test Accuracy:',result)
```

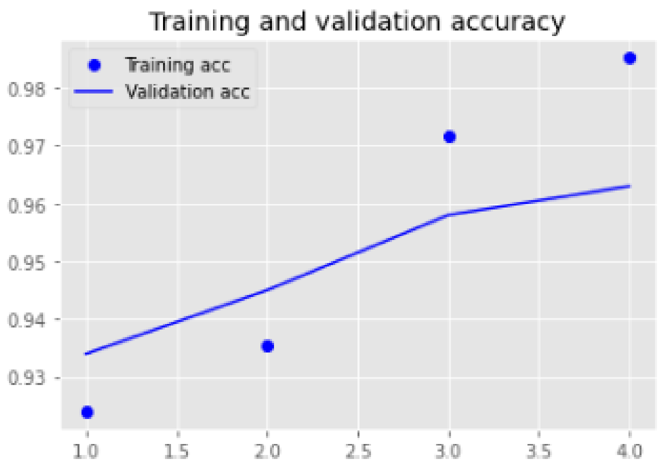
```
Test Accuracy: {'mcc': 0.0, 'acc': 0.39668367346938777, 'eval_loss': 0.69770230
23825425}
```

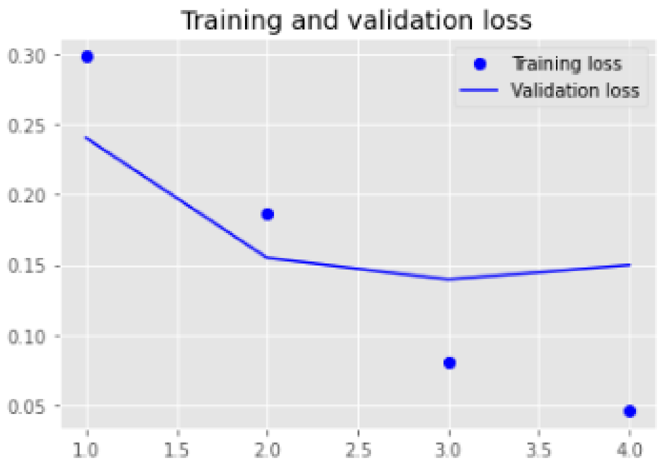
BERT: accuracy: 39.7%

Model 3: LSTM

```
cnfmatrix(y_test, results)
```

```
0.032816773017319965 0.014455007162390936
0.03772192559795112 0.9150062942223379
Precision:  0.6942148760330579 Recall:  0.4652307692307692
f1score:  0.5571112748710391
accuracy:  0.9478230672396579
hate_acc:  0.4652307692307692
non_hate_acc:  0.9844479730991967
```



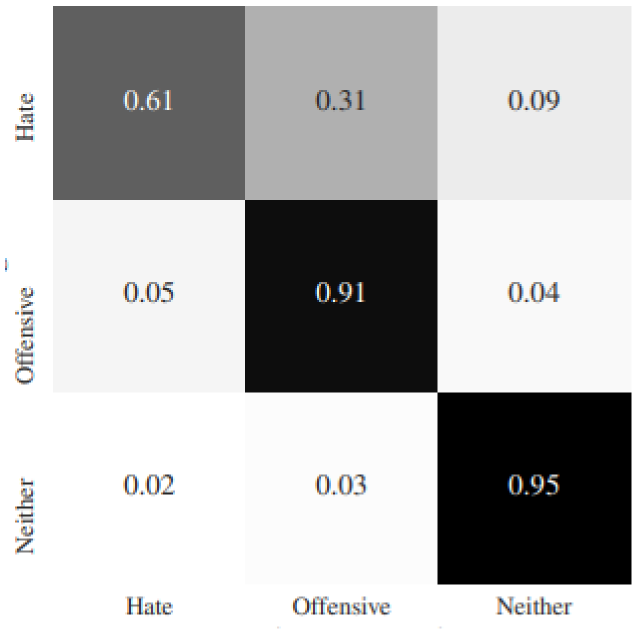


LSTM: accuracy: 94.7%

OUT OF ALL THE MODELS, LSTM AND XGBOOST WITH THE LIME ALGORITHM WORKED THE BEST.

However, the LSTM model was not used to deploy as it took a lot of time to get results and due to insufficient resources on the API due to the sheer size of the training data set.

We saw some of the most common words and hashtags in general and in racist/sexist tweets and extracted features by counting word tokens and Tfidf weighting them. We used unigrams, bigrams, and trigrams as tokens.
Finally, we built a logistic regression model, with XGBoost and LIME algorithm, to classify future tweets into 3 classes.
<u>The best-performing model has an overall precision of 0.91, a recall of 0.90, and an F1 score of 0.90.</u>



Lexical methods are effective ways to identify potentially offensive terms but are inaccurate at identifying hate speech.
While automated classification methods can achieve relatively high accuracy at differentiating between these different classes, close analysis of the results shows that the presence or absence of particular offensive or hateful terms can both help and hinder accurate classification.

<u>Hyperparameter Tuning</u>

N-gram and TF-IDF feature selection approaches were employed to select features. TF-IDF-based XGBoost was able to perform better or on par, compared to the results obtained by neural network models. Although in various literature reviews BERT based models worked well with an average of 89% accuracy, the model was lackluster on my dataset.

Despite transformers-based pre-training typically being much better than the TF-IDF-based MLP models.

For the final model, grid search and random search hyperparameter tuning methods were used.

Conclusion

Hate speech detection using Logistic Regression with the LIME algorithm was the best model and was used to deploy.
Even though the LSTM model has similar accuracy, it could not be used for deployment due to the resources (GPU) required.
Surprisingly the BERT model had the worst accuracy of them all, a bad ROC curve.
In the future, we can try getting real-time data using the Microsoft live server API and getting live data via the news-regional API.
By using cloud services, we could decrease computational time and make the deployment process faster and more lightweight.