Name: Faris Chaudhry
Batch: LISUM25

Hate Speech Detection

Week 12

Team Member Details

Name: Faris Chaudhry

Email: faris.chaudhry@outlook.com

Country: United Kingdom

University: Imperial College London

Specialization: NLP

Problem Description

“The term hate speech is understood as any type of verbal, written or behavioural communication that attacks or uses derogatory or discriminatory language against a person or group based on what they are, in other words, based on their religion, ethnicity, nationality, race, colour, ancestry, sex or another identity factor.

Hate Speech Detection is generally a task of sentiment classification. So, for training, a model that can classify hate speech from a certain piece of text can be achieved by training it on a data that is generally used to classify sentiments. We will use the Twitter tweets to identify tweets containing Hate speech.”

Name: Faris Chaudhry
Batch: LISUM25

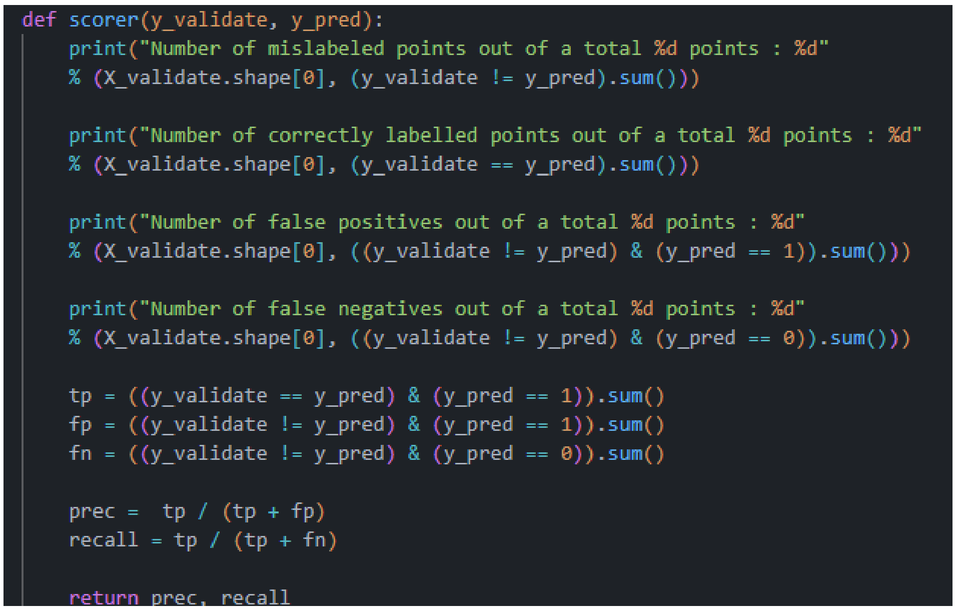
Foreword

Firstly, it is always a challenge to reduce overfitting when creating supervised models, that is, in balancing bias and variance to create optimal results when applied to unseen data. This has been dealt with by splitting the data into a training and testing set to perform cross-validation. Furthermore, since the number of samples labelled as hate speech is significantly smaller, the model could always be lenient in classifying all things as not hate speech and would have a high accuracy (approximately 90%). Therefore, the minority data (hate speech) has been up sampled so a random guess would give an accuracy of roughly 50%. This makes it easier to verify the model's classification abilities as significant in so much as they are not random change.

Secondly, we must choose the weight assigned to false negatives and false positives, and hence balance them accordingly. For example, we might say that false negatives are the worse option, since we want the platform to be completely free from hate speech, perhaps to reassure advertisers; to achieve this we would be stricter in classifying things as hate speech. Users might not like their tweets to get flagged as hate speech unjustifiably, however. Appeals would have to be done manually and require extra resources to deal with. On the other hand, we might consider flagging tweets which aren't hate speech as being detrimental to the platform (for the reasons previously) and therefore might be more lenient in classification. This might lead to more hate speech showing up on the platform, but this can be dealt with quickly if users flag it up. The choice is one that must be made by product stakeholders. However, in the following models, the main metric used is the F1 score, as a compromise between the two.

Model Creation

A general scorer function was created to calculate number of correctly and incorrectly classed samples, and the number of false positives and false negatives. Sklearn metric functions were used to calculate the F1 score and accuracy. In most cases, a TF-IDF transformer was used to weigh token significance with smoothing enables to prevent zero-division errors.

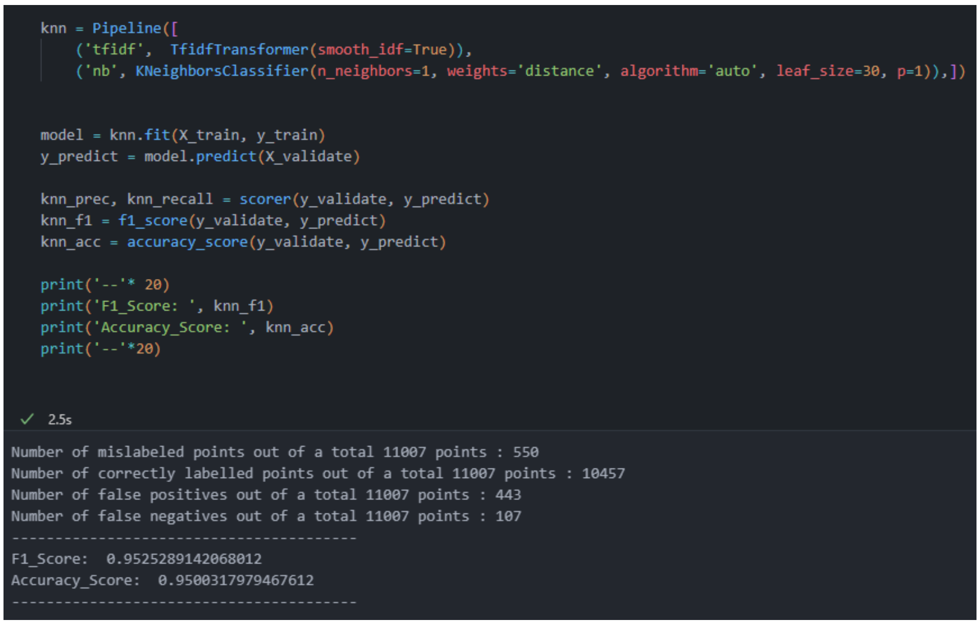


```python
def scorer(y_validate, y_pred):
    print("Number of mislabeled points out of a total %d points : %d"
    % (X_validate.shape[0], (y_validate != y_pred).sum()))

    print("Number of correctly labelled points out of a total %d points : %d"
    % (X_validate.shape[0], (y_validate == y_pred).sum()))

    print("Number of false positives out of a total %d points : %d"
    % (X_validate.shape[0], ((y_validate != y_pred) & (y_pred == 1)).sum()))

    print("Number of false negatives out of a total %d points : %d"
    % (X_validate.shape[0], ((y_validate != y_pred) & (y_pred == 0)).sum()))

    tp = ((y_validate == y_pred) & (y_pred == 1)).sum()
    fp = ((y_validate != y_pred) & (y_pred == 1)).sum()
    fn = ((y_validate != y_pred) & (y_pred == 0)).sum()

    prec =  tp / (tp + fp)
    recall = tp / (tp + fn)

    return prec, recall
```

Name: Faris Chaudhry
Batch: LISUM25

K-Nearest Neighbour (KNN)

KNN was quite successful. The F1 score generally went down as n-Neighbours was increased. Weight was done by distance rather than uniform since we have two classifications which are separate only in a few categories.



```
knn = Pipeline([
    ('tfidf',  TfidfTransformer(smooth_idf=True)),
    ('nb', KNeighborsClassifier(n_neighbors=1, weights='distance', algorithm='auto', leaf_size=30, p=1)),])


model = knn.fit(X_train, y_train)
y_predict = model.predict(X_validate)

knn_prec, knn_recall = scorer(y_validate, y_predict)
knn_f1 = f1_score(y_validate, y_predict)
knn_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', knn_f1)
print('Accuracy_Score: ', knn_acc)
print('--'*20)
```

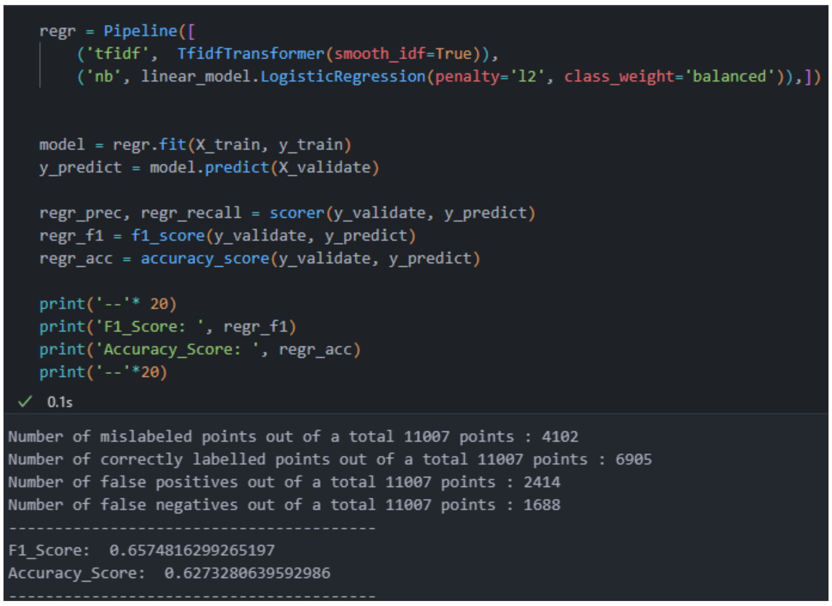
✓ 2.5s

```
Number of mislabeled points out of a total 11007 points : 550
Number of correctly labelled points out of a total 11007 points : 10457
Number of false positives out of a total 11007 points : 443
Number of false negatives out of a total 11007 points : 107
----------------------------------------
F1_Score:  0.9525289142068012
Accuracy_Score:  0.9500317979467612
----------------------------------------
```

Name: Faris Chaudhry
Batch: LISUM25

Logistic Regression

Logistic regression was significantly worse, this might be on account of not fully tuning hyperparameters, however the parameters I tried made very little difference to the F1 score. Balanced class weights were originally used because hate speech was by far the minority but after resampled the minority class, they are even so this makes almost no difference (depending on which the groups the samples are split into).



```
regr = Pipeline([
    ('tfidf',  TfidfTransformer(smooth_idf=True)),
    ('nb', linear_model.LogisticRegression(penalty='l2', class_weight='balanced')),])


model = regr.fit(X_train, y_train)
y_predict = model.predict(X_validate)

regr_prec, regr_recall = scorer(y_validate, y_predict)
regr_f1 = f1_score(y_validate, y_predict)
regr_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', regr_f1)
print('Accuracy_Score: ', regr_acc)
print('--'*20)
```

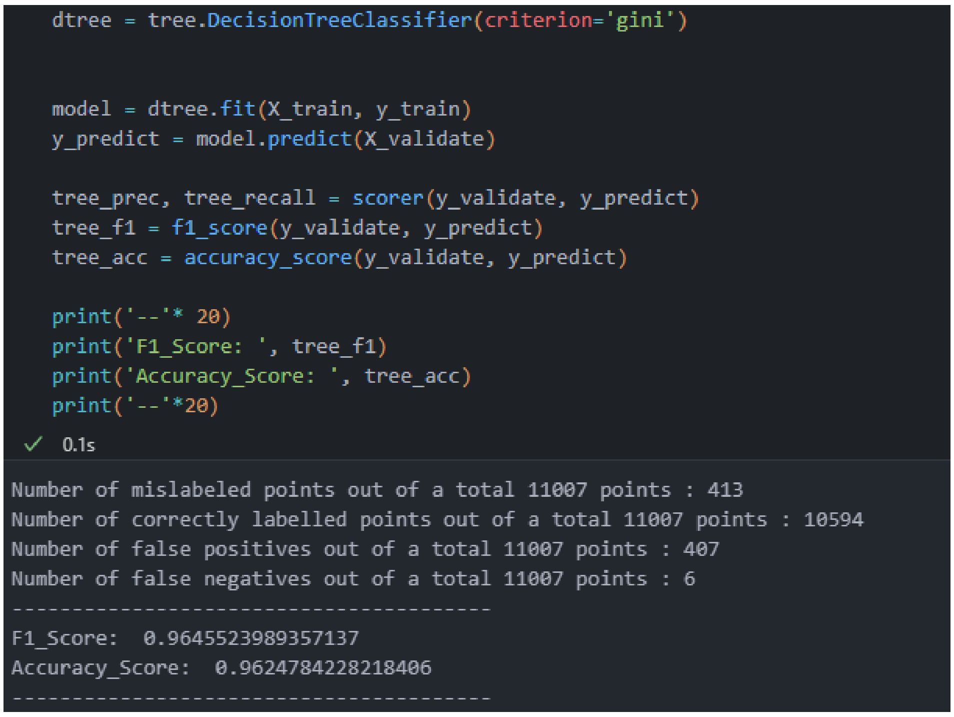
✓ 0.1s

```
Number of mislabeled points out of a total 11007 points : 4102
Number of correctly labelled points out of a total 11007 points : 6905
Number of false positives out of a total 11007 points : 2414
Number of false negatives out of a total 11007 points : 1688
----------------------------------------
F1_Score:  0.6574816299265197
Accuracy_Score:  0.6273280639592986
----------------------------------------
```

Name: Faris Chaudhry
Batch: LISUM25

Decision Trees

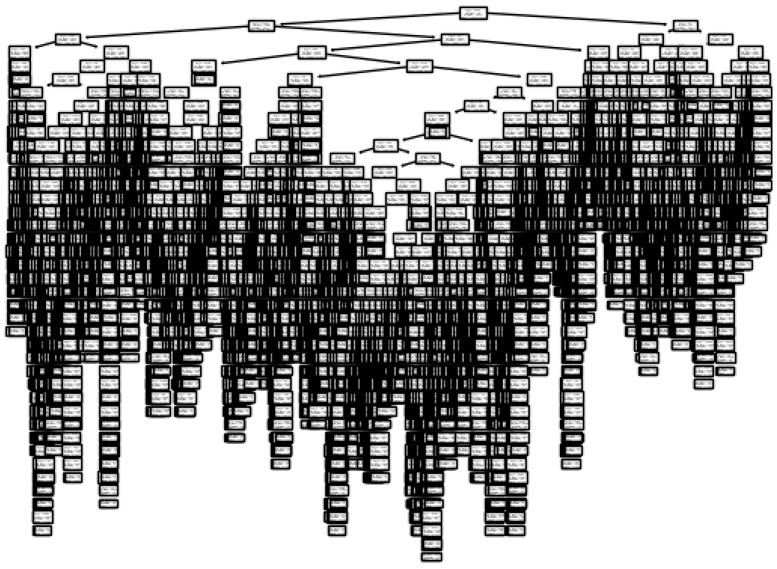
The decision tree method gave the best result. It was the strictest, giving only six false negatives and generally handled outliers better. Since, the depth of the tree was unrestricted, the tree itself is extremely complex and messy when plotted (see below). Different criterion methods were tested (entropy, log loss, gini) and produced similar results (F1 score between 0.955 and 0.0965). Compared to other methods, this one produces the most variance in F1 score whenever the algorithm is trained (most are the exact same). Potentially this is caused by not seeding the tree which could done to give repeatable training results.



```
dtree = tree.DecisionTreeClassifier(criterion='gini')


model = dtree.fit(X_train, y_train)
y_predict = model.predict(X_validate)

tree_prec, tree_recall = scorer(y_validate, y_predict)
tree_f1 = f1_score(y_validate, y_predict)
tree_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', tree_f1)
print('Accuracy_Score: ', tree_acc)
print('--'*20)
```

✓ 0.1s

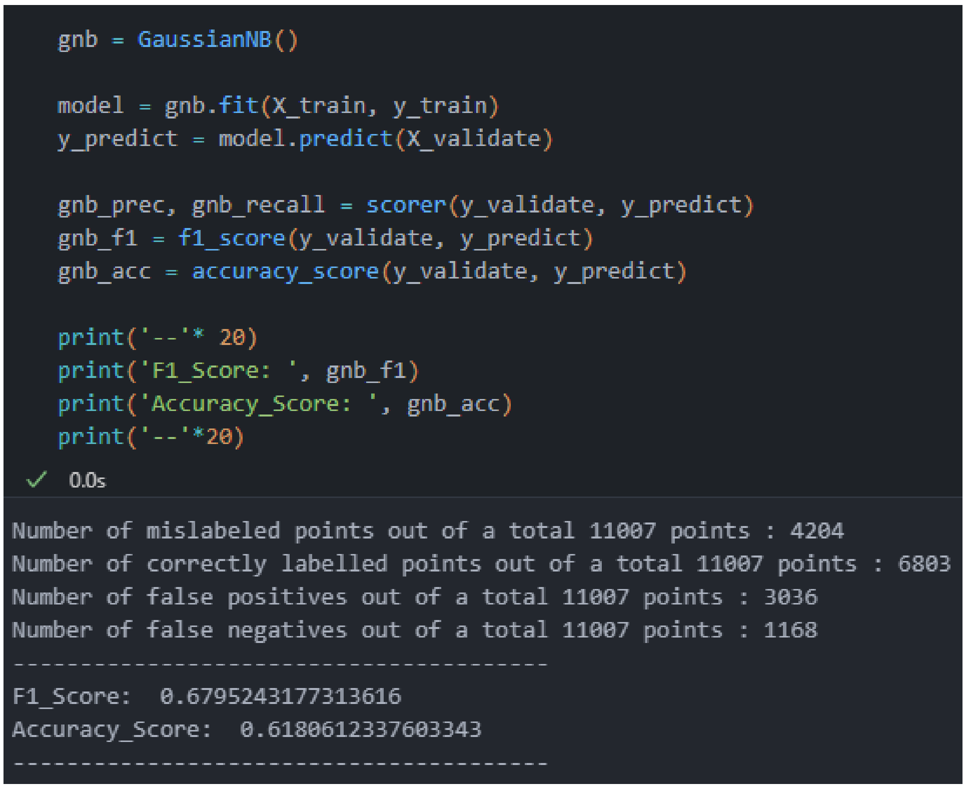
```
Number of mislabeled points out of a total 11007 points : 413
Number of correctly labelled points out of a total 11007 points : 10594
Number of false positives out of a total 11007 points : 407
Number of false negatives out of a total 11007 points : 6
----------------------------------------
F1_Score:  0.9645523989357137
Accuracy_Score:  0.9624784228218406
----------------------------------------
```



Name: Faris Chaudhry
Batch: LISUM25

Gaussian Naïve Bayes (GNB)

GNB suffered from some problems such as not being able to use TF-IDF transformations because the result of the transformation is dense and the GNB requires a dense matrix (this is potentially fixable). In general, the models with F1 score less than 0.7 could be improved by using a CountVectoriser transformation followed by a TF-IDF transformation, however there was numerous incompatibilities between data types when I was trying to implement this.



```
gnb = GaussianNB()

model = gnb.fit(X_train, y_train)
y_predict = model.predict(X_validate)

gnb_prec, gnb_recall = scorer(y_validate, y_predict)
gnb_f1 = f1_score(y_validate, y_predict)
gnb_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', gnb_f1)
print('Accuracy_Score: ', gnb_acc)
print('--'*20)
```

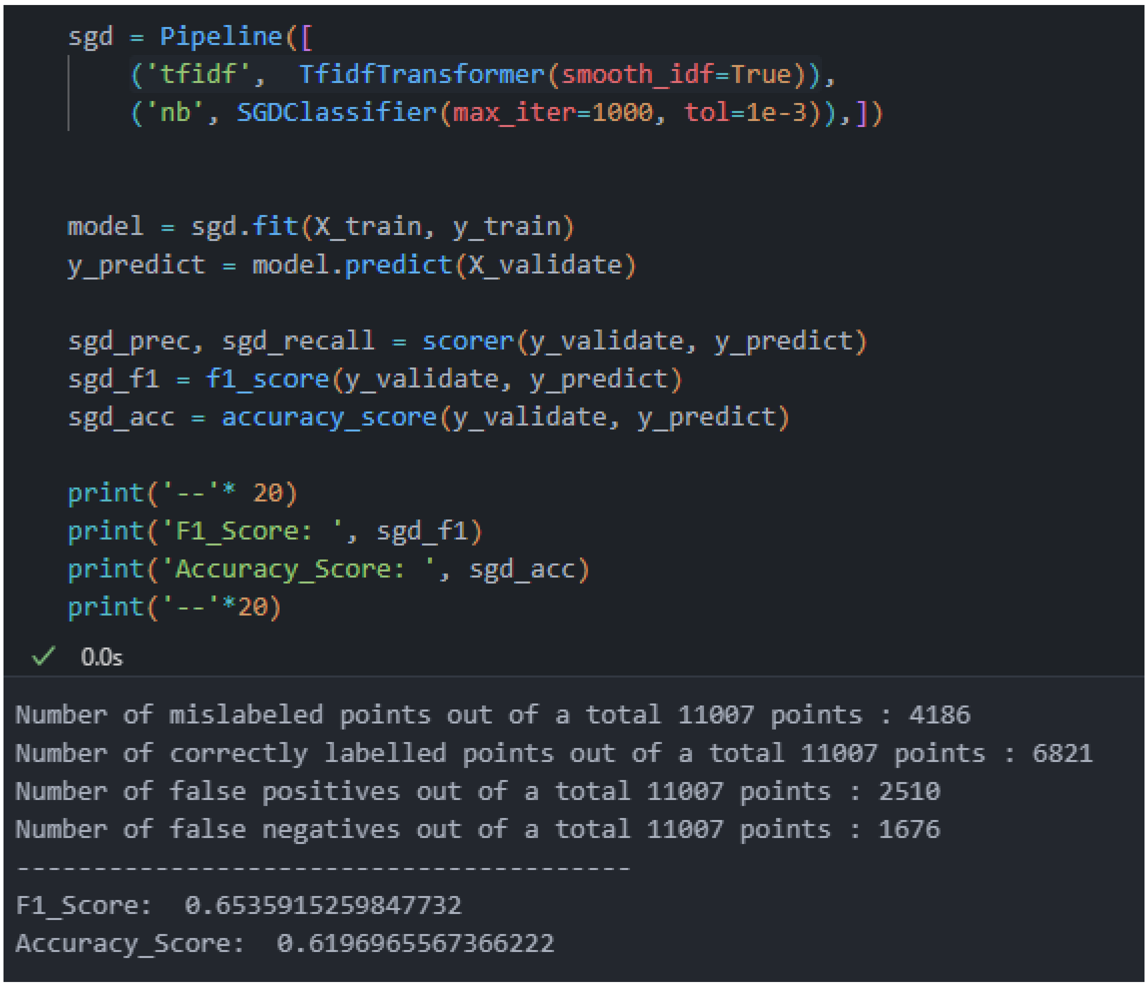
✓ 0.0s

```
Number of mislabeled points out of a total 11007 points : 4204
Number of correctly labelled points out of a total 11007 points : 6803
Number of false positives out of a total 11007 points : 3036
Number of false negatives out of a total 11007 points : 1168
----------------------------------------
F1_Score:  0.6795243177313616
Accuracy_Score:  0.6180612337603343
----------------------------------------
```

Name: Faris Chaudhry
Batch: LISUM25

Stochastic Gradient Descent (SGD)

Would benefit from the same use of CountVectoriser above. Perhaps using accelerated gradient descent or different descent rate would benefit it. As it stands, this was the worst of the tested methods.



```python
sgd = Pipeline([
    ('tfidf',  TfidfTransformer(smooth_idf=True)),
    ('nb', SGDClassifier(max_iter=1000, tol=1e-3)),])


model = sgd.fit(X_train, y_train)
y_predict = model.predict(X_validate)

sgd_prec, sgd_recall = scorer(y_validate, y_predict)
sgd_f1 = f1_score(y_validate, y_predict)
sgd_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', sgd_f1)
print('Accuracy_Score: ', sgd_acc)
print('--'*20)
```

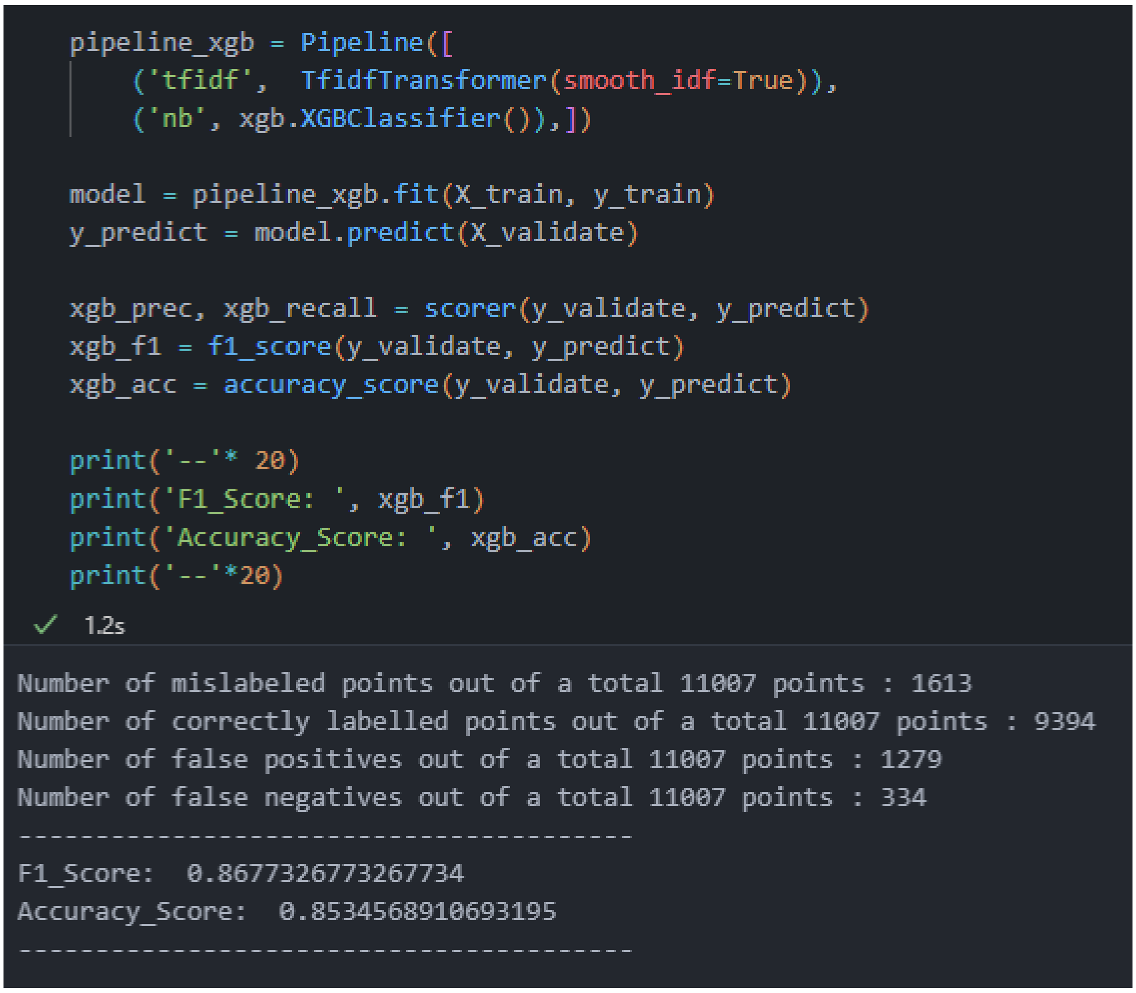
✓ 0.0s

```
Number of mislabeled points out of a total 11007 points : 4186
Number of correctly labelled points out of a total 11007 points : 6821
Number of false positives out of a total 11007 points : 2510
Number of false negatives out of a total 11007 points : 1676
----------------------------------------
F1_Score:  0.6535915259847732
Accuracy_Score:  0.6196965567366222
```

Name: Faris Chaudhry
Batch: LISUM25

XGBClassifier (Ensemble)

XGB worked moderately well. More features would have likely helped it perform better since as maybe creating a list of commonly used words in the samples labelled hate speech. The reason this was not done is because language changes frequently (particularly slurs and derogatory language) so the model would need to be retrained often if using explicit features.



```
pipeline_xgb = Pipeline([
    ('tfidf',  TfidfTransformer(smooth_idf=True)),
    ('nb', xgb.XGBClassifier()),])

model = pipeline_xgb.fit(X_train, y_train)
y_predict = model.predict(X_validate)

xgb_prec, xgb_recall = scorer(y_validate, y_predict)
xgb_f1 = f1_score(y_validate, y_predict)
xgb_acc = accuracy_score(y_validate, y_predict)

print('--'* 20)
print('F1_Score: ', xgb_f1)
print('Accuracy_Score: ', xgb_acc)
print('--'*20)
```

✓ 1.2s

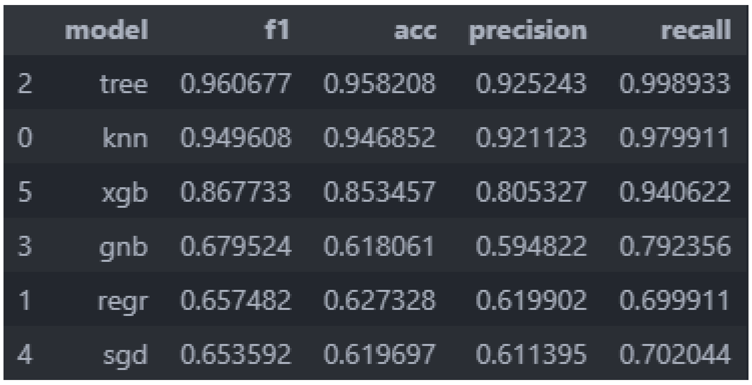
```
Number of mislabeled points out of a total 11007 points : 1613
Number of correctly labelled points out of a total 11007 points : 9394
Number of false positives out of a total 11007 points : 1279
Number of false negatives out of a total 11007 points : 334
----------------------------------------
F1_Score:  0.8677326773267734
Accuracy_Score:  0.8534568910693195
----------------------------------------
```

Name: Faris Chaudhry
Batch: LISUM25

Results and Scores

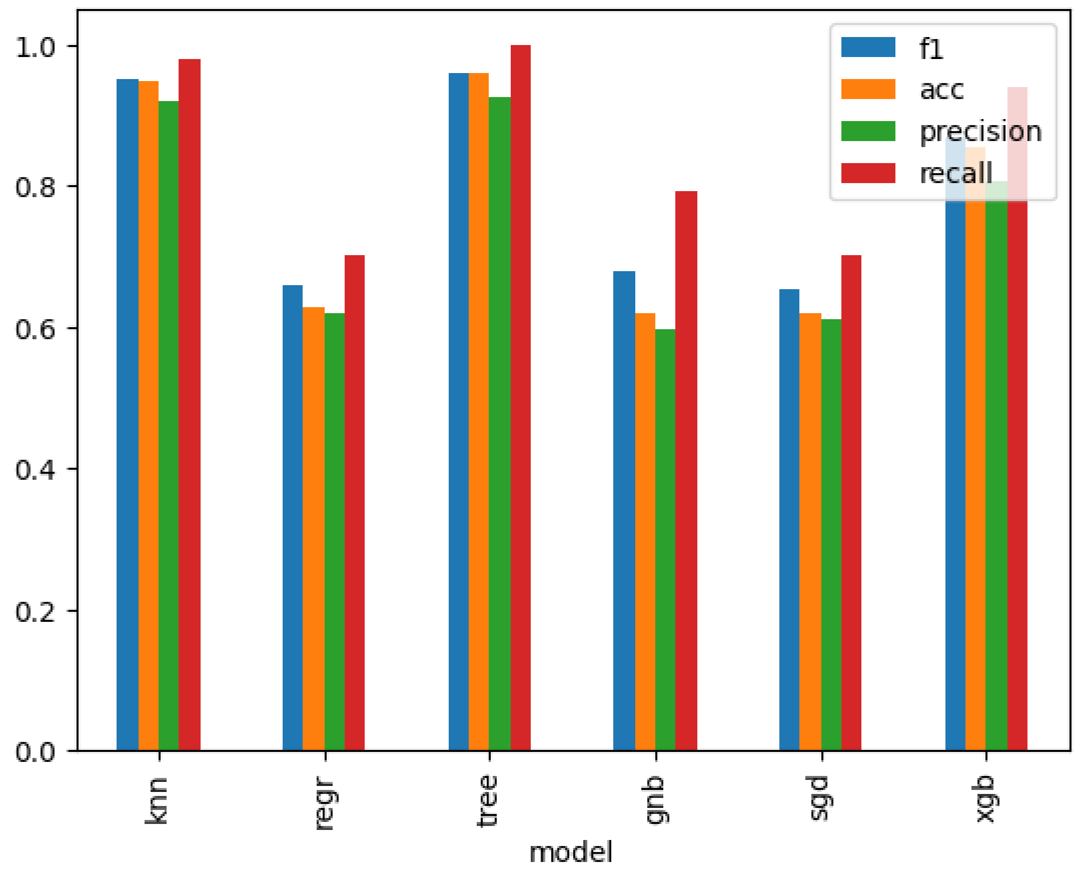
Decision tree ended up being the best in F1 score and first in precision and recall. It was exceptionally good at preventing false negatives which prevents as much hate speech as possible. I believe that logistic regression and XGB classification could benefit the most from hyper tuning their parameters (XGB might also work better with more data features given that it is an ensemble method).

Because we want to eliminate the political leanings of a single group, using a black box methodology is optimal here. Obviously, any sort of censorship is requires some political stance in saying what should be censored, and there can be a bias in the data chosen and how it is labelled, but using black box algorithms over individual people reviewing cases of flagged hate speech and banning it relying on their own discretion (even if following a policy) are more suitable to the task, especially since they are automated and can review tweets before they are put into the public-facing sphere.



	model	f1	acc	precision	recall
2	tree	0.960677	0.958208	0.925243	0.998933
0	knn	0.949608	0.946852	0.921123	0.979911
5	xgb	0.867733	0.853457	0.805327	0.940622
3	gnb	0.679524	0.618061	0.594822	0.792356
1	regr	0.657482	0.627328	0.619902	0.699911
4	sgd	0.653592	0.619697	0.611395	0.702044

Name: Faris Chaudhry
Batch: LISUM25


f1
acc
precision
recall
knn
regr
tree
gnb
sgd
xgb
model