HateBot Machine Learning

1. Introduction

Hate speech is something that is a big issue nowadays especially with social media like Twitter. Negativity that comes out of it can take a toll on a person's mental health. To minimize the damage of that, the HateBot can return if a tweet is hate speech or not by being given a Tweet. This way we can ensure that the world becomes if not a better place then at least a place where we can influence the amount of negativity we get.

Note: This is the second notebook of this project. The EDA is conducted in a separate Jupyter notebook that can be found in the submission.

2. Setup

In this part of the notebook, I am going to do the importing of libraries/modules and the dataset and setup the global settings needed for the figures.

2.1 Imports

In [49]:
```
# Imports
import pandas as pd

from collections import Counter

import matplotlib
import matplotlib.pyplot as plt

import scattertext as st
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_extraction.text import HashingVectorizer, CountVectorizer

print('pandas version:', pd.__version__)
print('matplotlib version:', matplotlib.__version__)
print('scattertext version:', st.__version__)
```

```
%matplotlib inline
```

```
pandas version: 1.1.3
matplotlib version: 3.3.2
scattertext version: 0.1.2
```

2.2 Importing the dataset that will be used

The dataset is stored in a CSV format (comma-separated values file) in the file `HateBotDataset.csv` . This dataset is already cleaned in the EDA phase.

In [50]:
```
df = pd.read_csv("CleanedHateBotDataset.csv")
```

2.2.1 Check the data in the dataset

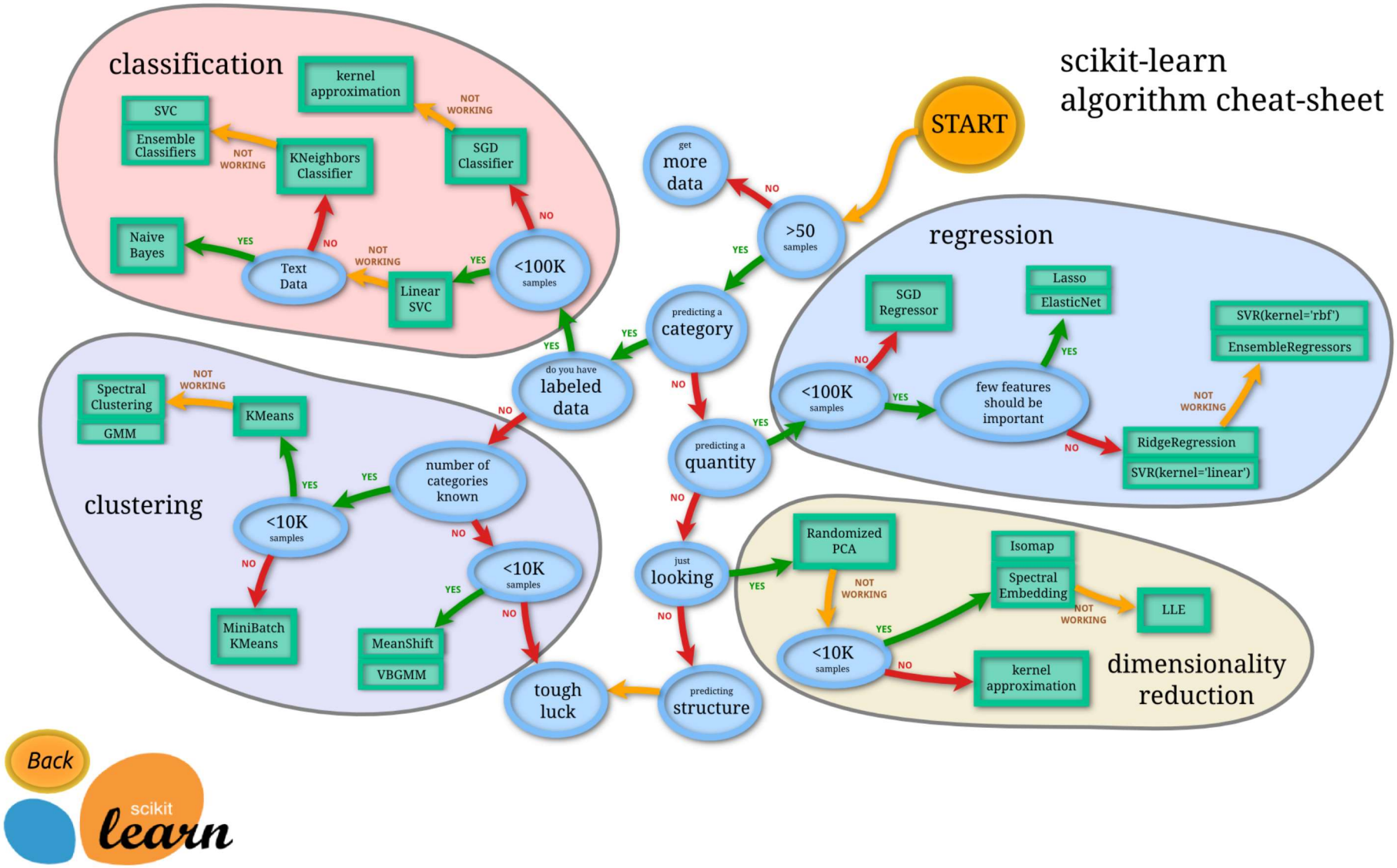
To be sure, we want to see the first five entries of the dataset to make sure that we have loaded the correct file.

In [51]:
```
df.head()
```

Out[51]:

	count	neither	class	tweet	marked	is_hate_speech
0	3	3	2	RT As a woman you shouldnt complain about cl...	0	False
1	3	0	1	RT boy dats coldtyga dwn bad for cuffin dat ...	3	True
2	3	0	1	RT Dawg RT You ever fuck a bitch and she st...	3	True
3	3	1	1	RT she look like a tranny	2	True
4	6	0	1	RT The shit you hear about me might be true ...	6	True

2.3 Selection of an algorithm for the Machine Learning



The algorithms that will be used for this challenge are Support Vector Classification (SVC) and Naive Bayes. The following algorithms were selected because we have our sample size (less than 100k) and because we are using text data (because the Tweets are text). Naive bayes can be very good with low amounts of data. Decision trees work better with lots of data compared to Naive Bayes but our sample size is not big enough.

2.4 Data description

From the EDA phase, it has been decided that we are going to train via the `tweet` column (data type: `string`) and the `is_hate_speech` (data type: `Boolean`) column.

3. Machine learning using a Linear SVC

3.1 Training using a SVC with a CountVectorizer

The CountVectorizer converts a collection of text documents to a matrix of token counts. This is needed because machine learning algorithms cannot run on raw text data. One limitation of these methods is that the vocabulary can become very large.

3.1.1 Get all the words that are in all the tweets

In [52]:
```
d = Counter(" ".join(df.tweet).split(" ")).items()
most_used_words = dict(sorted(d, key=lambda item: item[1], reverse=True))
words = {key for key, val in most_used_words.items()}

count = 0
for elem in iter(most_used_words):
    if count == 6:
        break
    print(elem)
    count = count + 1
```

```
a
bitch
RT
the
I
```

3.1.2 Create and fit the CountVectorizer

Here we tokenize the words and build the vocabulary.

In [53]:
```
countVectorizer = CountVectorizer()
countVectorizer.fit(words)
```

Out[53]:
```
CountVectorizer()
```

3.1.3 Encode all the words from the Tweets

In [54]:
```
countVector = countVectorizer.transform(words)
```

```
print(countVector.shape)
```

```
(34011, 28861)
```

We can see that we have 34011 rows of data with 28861 columns

3.1.4 Transform every tweet into a pandas Series

We add a new column called vectorised_words which has for every row in the dataset the tweet itself but after it was transformed via our count vectorizer.

In [55]:
```
df['vectorised_words'] = countVectorizer.transform(df.tweet.values)
```

In [56]:
```
type(df['vectorised_words'])
df.head()
```

Out[56]:

	count	neither	class	tweet	marked	is_hate_speech	vectorised_words
0	3	3	2	RT As a woman you shouldnt complain about cl...	0	False	(0, 2290)\t1\n (0, 2735)\t1\n (0, 2799)\t1...
1	3	0	1	RT boy dats coldtyga dwn bad for cuffin dat ...	3	True	(0, 2290)\t1\n (0, 2735)\t1\n (0, 2799)\t1...
2	3	0	1	RT Dawg RT You ever fuck a bitch and she st...	3	True	(0, 2290)\t1\n (0, 2735)\t1\n (0, 2799)\t1...
3	3	1	1	RT she look like a tranny	2	True	(0, 2290)\t1\n (0, 2735)\t1\n (0, 2799)\t1...
4	6	0	1	RT The shit you hear about me might be true ...	6	True	(0, 2290)\t1\n (0, 2735)\t1\n (0, 2799)\t1...

3.2 Training the SVC

3.2.1 Create the needed variables

In [57]:
```
svc_1 = SVC(kernel='linear')
```

In [58]:
```
X = countVectorizer.transform(df.tweet.values)
y = df['is_hate_speech'].astype(int)
```

In [59]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
```

As you can see from the test_size parameter, we keep 30% of the data for testing.

3.2.2 Fit the SVC

In [60]:
```
svc_1.fit(X_train, y_train)
```

Out[60]:
```
SVC(kernel='linear')
```

3.2.3 Evaluate the predictions

In [61]:
```
y_pred = svc_1.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84      1239
           1       0.97      0.97      0.97      6196

    accuracy                           0.95      7435
   macro avg       0.91      0.90      0.91      7435
weighted avg       0.95      0.95      0.95      7435
```

In [62]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.947679892400807
Precision: 0.9679290894439968
Recall: 0.9693350548741123
```

We can see that this model is quite succesfull - it's accuracy is almost at 95% which in our case is more than sufficient because the initial target accuracy was over 75%.

3.2.4 Hyperparameter tuning

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. To reach to the highest performance possible of a model, we need to try different hyperparameters. To achieve these, GridSearch will be used:

In [63]:
```
from sklearn.model_selection import GridSearchCV
```

In [64]:
```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, n_jobs = -1)
```

In [65]:
```
grid.fit(X_train,y_train)
```

```
estimations = grid.best_estimator_
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 15.6min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed: 24.6min finished
```

In [66]:
```
print(estimations)
```

```
SVC(C=10, gamma=0.01, kernel='sigmoid')
```

From the GridSearch, we can see that we should use SVC(C=10, gamma=0.01, kernel='sigmoid'. Now we will make that so that we have the best possible SVC.

In [67]:
```
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
```

```
[[1094  145]
 [ 215 5981]]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1239
           1       0.98      0.97      0.97      6196

    accuracy                           0.95      7435
   macro avg       0.91      0.92      0.91      7435
weighted avg       0.95      0.95      0.95      7435
```

3.2.4.1 Creating the hypertuned version

In [68]:
```
svc_1 = SVC(C=10, gamma=0.01, kernel='sigmoid')
X = countVectorizer.transform(df.tweet.values)
y = df['is_hate_speech'].astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
svc_1.fit(X_train, y_train)
y_pred = svc_1.predict(X_test)
print(classification_report(y_test,y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.84      0.88      0.86      1239
              1       0.98      0.97      0.97      6196

    accuracy                            0.95      7435
   macro avg       0.91      0.92      0.91      7435
weighted avg       0.95      0.95      0.95      7435

Accuracy: 0.9515803631472763
Precision: 0.9763303950375449
Recall: 0.9653001936733376
```

Without the hyperparameter tuning, we had the following results:

```
Accuracy: 0.94
Precision: 0.96
Recall: 0.96
```

With it, we had the following results:

```
Accuracy: 0.95
Precision: 0.97
Recall: 0.96
```

We can see that there is an increase in the Accuracy and precision metrics of our model

3.2.5 Save as a pickle

For the deployment part of the project, we will be having a Django web application where the user can enter the Tweet they want to get checked and then return if the Tweet is hate or not. To do so, we need the already trained model. For this, I have decided to use pickle because it is very easy to integrate with Django. It is used for serializing and de-serializing of Python object structures. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.

Now we will extract this SVC model to a pickle:

In [69]:
```
import pickle
pickl = {
    'vectorizer': countVectorizer,
    'regressor': svc_1
```

```
}
pickle.dump( pickl, open( "HateBotModels.p", "wb" ) )
```

3.3 Training using a SVC with a HashingVectorizer

This strategy has several advantages:

- it is very low memory scalable to large datasets as there is no need to store a vocabulary dictionary in memory
- it is fast to pickle and un-pickle as it holds no state besides the constructor parameters
- it can be used in a streaming (partial fit) or parallel pipeline as there is no state computed during fit.

There are also a couple of cons (vs using a CountVectorizer with an in-memory vocabulary):

- there is no way to compute the inverse transform (from feature indices to string feature names) which can be a problem when trying to introspect which features are most important to a model.
- there can be collisions: distinct tokens can be mapped to the same feature index. However in practice this is rarely an issue if n_features is large enough (e.g. 2 ** 18 for text classification problems).
- no IDF weighting as this would render the transformer stateful.

Source: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html

3.3.1 Create the HashingVectorizer

In [70]:
```
hashingVectorizer = HashingVectorizer(n_features=20)
vector = hashingVectorizer.transform(df.tweet)
df['vectorised_words'] = hashingVectorizer.transform(df.tweet.values)
```

3.3.2 Create the needed variables and split the dataset into test and train

In [71]:
```
svc_2 = SVC(kernel='linear')
```

In [72]:
```
X = hashingVectorizer.transform(df.tweet.values)
y = df['is_hate_speech'].astype(int)
```

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
```

3.3.3 Fit the SVC

In [74]:
```
svc_2.fit(X_train, y_train)
```

Out[74]:
```
SVC(kernel='linear')
```

3.3.4 Evaluate the results

In [75]:
```
y_pred = svc_2.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1239
           1       0.83      1.00      0.91      6196

    accuracy                           0.83      7435
   macro avg       0.42      0.50      0.45      7435
weighted avg       0.69      0.83      0.76      7435

C:\Users\karin\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [76]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8333557498318762
Precision: 0.8333557498318762
Recall: 1.0
```

We can see that when using the SVC with a HashingVectorizer, the accuracy of the model goes down. this can be explained by the fact that distinct tokens can be mapped to the same feature index using the `HashingVectorizer` in contract to using a `CountVectorizer`. However, this is still a very good result (because the accuracy is 0.83) even though it is worse than our previous one. It is important to note that here we have a perfect recall score of 1 - that means that all relevant instances were retrieved by the search (but says nothing about how many irrelevant instances were also retrieved).

3.3.5 Hyperparameter tuning

As we saw in the previous model, hyperparameter tuning can improve the performance of our model. That's why we would also do it for this one.

In [77]:
```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, n_jobs = -1)
```

In [78]:
```
grid.fit(X_train,y_train)
estimations = grid.best_estimator_
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  8.2min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed: 22.5min finished
```

In [79]:
```
print(estimations)
```

```
SVC(C=1, gamma=1)
```

From the hypertuning, we can see that the best model is one with the following parameters:

```
SVC(C=1, gamma=1)
```

In [80]:
```
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
```

```
[[   2 1237]
 [   0 6196]]
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1239
           1       0.83      1.00      0.91      6196

    accuracy                           0.83      7435
   macro avg       0.92      0.50      0.46      7435
weighted avg       0.86      0.83      0.76      7435
```

In [81]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8333557498318762
Precision: 0.8333557498318762
Recall: 1.0
```

4. Machine learning using Naive Bayes

In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. They are among the simplest Bayesian network model but coupled with kernel density estimation, they can achieve higher accuracy levels.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.

4.1 Naive Bayes with a CountVectorizer

4.1.1 Imports

In [82]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

4.1.2 Split the dataset into test and train

In [83]:
```
X = countVectorizer.transform(df.tweet.values).toarray()
y = df['is_hate_speech'].astype(int)
```

In [84]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
```

4.1.3 Create the Gaussian Naive Bayes

In [85]:
```
gnb = GaussianNB()
```

4.1.4 Fit the model

In [86]:
```
gnb.fit(X_train, y_train)
```

Out[86]:
```
GaussianNB()
```

4.1.5 Predict

In [87]:
```
y_pred = gnb.predict(X_test)
```

4.1.6 Calculate the metrics (accuracy, precision, recall)

In [88]:
```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.655413584398117
Precision: 0.9037777777777778
Recall: 0.6563912201420271
```

4.1.7 Hyperparameter tuning

We will try to tune the var_smoothing parameter. This variable artificially adds a user-defined value to the distribution's variance (whose default value is derived from the training data set). This essentially widens (or "smooths") the curve and accounts for more samples that are further away from the distribution mean.

In [89]:
```
parameters = {
    'var_smoothing': [1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14, 1e-15]
}
clf = GridSearchCV(nb_classifier, parameters, cv=5, verbose=2)
clf.fit(X, y)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[CV] var_smoothing=0.01 ..............................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] ............................... var_smoothing=0.01, total=  40.2s
[CV] var_smoothing=0.01 ..............................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   40.6s remaining:    0.0s
[CV] ............................... var_smoothing=0.01, total=  29.5s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=  24.8s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=  23.2s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=  24.2s
[CV] var_smoothing=0.001 .............................................
[CV] ............................... var_smoothing=0.001, total=  29.7s
[CV] var_smoothing=0.001 .............................................
[CV] ............................... var_smoothing=0.001, total=  24.2s
[CV] var_smoothing=0.001 .............................................
[CV] ............................... var_smoothing=0.001, total=  22.8s
[CV] var_smoothing=0.001 .............................................
[CV] ............................... var_smoothing=0.001, total=  21.5s
[CV] var_smoothing=0.001 .............................................
```

```
[CV] ............................. var_smoothing=0.001, total=  24.6s
[CV] var_smoothing=0.0001 .....................................
[CV] ............................. var_smoothing=0.0001, total=  24.1s
[CV] var_smoothing=0.0001 .....................................
[CV] ............................. var_smoothing=0.0001, total=  25.8s
[CV] var_smoothing=0.0001 .....................................
[CV] ............................. var_smoothing=0.0001, total=  29.4s
[CV] var_smoothing=0.0001 .....................................
[CV] ............................. var_smoothing=0.0001, total=  23.5s
[CV] var_smoothing=0.0001 .....................................
[CV] ............................. var_smoothing=0.0001, total=  24.2s
[CV] var_smoothing=1e-05 ......................................
[CV] ............................. var_smoothing=1e-05, total=  24.1s
[CV] var_smoothing=1e-05 ......................................
[CV] ............................. var_smoothing=1e-05, total=  23.8s
[CV] var_smoothing=1e-05 ......................................
[CV] ............................. var_smoothing=1e-05, total=  23.0s
[CV] var_smoothing=1e-05 ......................................
[CV] ............................. var_smoothing=1e-05, total=  22.5s
[CV] var_smoothing=1e-05 ......................................
[CV] ............................. var_smoothing=1e-05, total=  23.9s
[CV] var_smoothing=1e-06 ......................................
[CV] ............................. var_smoothing=1e-06, total=  24.2s
[CV] var_smoothing=1e-06 ......................................
[CV] ............................. var_smoothing=1e-06, total=  24.0s
[CV] var_smoothing=1e-06 ......................................
[CV] ............................. var_smoothing=1e-06, total=  26.7s
[CV] var_smoothing=1e-06 ......................................
[CV] ............................. var_smoothing=1e-06, total=  27.7s
[CV] var_smoothing=1e-06 ......................................
[CV] ............................. var_smoothing=1e-06, total=  25.8s
[CV] var_smoothing=1e-07 ......................................
[CV] ............................. var_smoothing=1e-07, total=  26.6s
[CV] var_smoothing=1e-07 ......................................
[CV] ............................. var_smoothing=1e-07, total=  27.2s
[CV] var_smoothing=1e-07 ......................................
[CV] ............................. var_smoothing=1e-07, total=  24.9s
[CV] var_smoothing=1e-07 ......................................
[CV] ............................. var_smoothing=1e-07, total=  29.2s
[CV] var_smoothing=1e-07 ......................................
[CV] ............................. var_smoothing=1e-07, total=  26.9s
[CV] var_smoothing=1e-08 ......................................
[CV] ............................. var_smoothing=1e-08, total=  29.0s
[CV] var_smoothing=1e-08 ......................................
[CV] ............................. var_smoothing=1e-08, total=  26.7s
[CV] var_smoothing=1e-08 ......................................
[CV] ............................. var_smoothing=1e-08, total=  29.2s
```

```
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=  30.8s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=  28.6s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=  29.4s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=  27.8s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=  34.8s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=  36.1s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=  39.7s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=  44.8s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=  37.6s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=  45.3s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=  32.7s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=  30.4s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=  29.3s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=  28.8s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=  29.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=  31.3s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=  31.3s
[CV] var_smoothing=1e-12 ..........................................
[CV] ............................ var_smoothing=1e-12, total=  30.7s
[CV] var_smoothing=1e-12 ..........................................
[CV] ............................ var_smoothing=1e-12, total=  32.1s
[CV] var_smoothing=1e-12 ..........................................
[CV] ............................ var_smoothing=1e-12, total=  37.9s
[CV] var_smoothing=1e-12 ..........................................
[CV] ............................ var_smoothing=1e-12, total=  32.3s
[CV] var_smoothing=1e-12 ..........................................
[CV] ............................ var_smoothing=1e-12, total=  34.7s
[CV] var_smoothing=1e-13 ..........................................
[CV] ............................ var_smoothing=1e-13, total=  32.9s
[CV] var_smoothing=1e-13 ..........................................
```

```
[CV] ............................. var_smoothing=1e-13, total=  35.5s
[CV] var_smoothing=1e-13 ..........................................
[CV] ............................. var_smoothing=1e-13, total=  33.8s
[CV] var_smoothing=1e-13 ..........................................
[CV] ............................. var_smoothing=1e-13, total=  33.4s
[CV] var_smoothing=1e-13 ..........................................
[CV] ............................. var_smoothing=1e-13, total=  34.7s
[CV] var_smoothing=1e-14 ..........................................
[CV] ............................. var_smoothing=1e-14, total=  31.9s
[CV] var_smoothing=1e-14 ..........................................
[CV] ............................. var_smoothing=1e-14, total=  35.0s
[CV] var_smoothing=1e-14 ..........................................
[CV] ............................. var_smoothing=1e-14, total=  31.8s
[CV] var_smoothing=1e-14 ..........................................
[CV] ............................. var_smoothing=1e-14, total=  31.1s
[CV] var_smoothing=1e-14 ..........................................
[CV] ............................. var_smoothing=1e-14, total=  33.0s
[CV] var_smoothing=1e-15 ..........................................
[CV] ............................. var_smoothing=1e-15, total=  32.5s
[CV] var_smoothing=1e-15 ..........................................
[CV] ............................. var_smoothing=1e-15, total=  34.2s
[CV] var_smoothing=1e-15 ..........................................
[CV] ............................. var_smoothing=1e-15, total=  36.2s
[CV] var_smoothing=1e-15 ..........................................
[CV] ............................. var_smoothing=1e-15, total=  31.4s
[CV] var_smoothing=1e-15 ..........................................
[CV] ............................. var_smoothing=1e-15, total=  33.9s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed: 35.4min finished
```

Out[89]:
```
GridSearchCV(cv=5, estimator=GaussianNB(),
             param_grid={'var_smoothing': [0.01, 0.001, 0.0001, 1e-05, 1e-06,
                                           1e-07, 1e-08, 1e-09, 1e-10, 1e-11,
                                           1e-12, 1e-13, 1e-14, 1e-15]},
             verbose=2)
```

In [103…
```
pd.DataFrame(clf.cv_results_).sort_values(by=['rank_test_score', 'mean_fit_time']).head()
```

Out[103…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_var_smoothing	params	split0_test_score	split1_test_score	split2_test_score
0	0.030201	0.004070	0.006997	0.002531	0.01	{'var_smoothing': 0.01}	0.825701	0.824491	0.829131
1	0.026600	0.000802	0.005602	0.000492	0.001	{'var_smoothing': 0.001}	0.825096	0.823078	0.828525
2	0.022804	0.000759	0.004801	0.001327	0.0001	{'var_smoothing': 0.0001}	0.825096	0.822877	0.828727

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_var_smoothing	params	split0_test_score	split1_test_score	split2_test_score
7	0.009139	0.007476	0.006263	0.007670	1e-09	{'var_smoothing': 1e-09}	0.825096	0.822675	0.828727
9	0.013457	0.002956	0.000000	0.000000	1e-11	{'var_smoothing': 1e-11}	0.825096	0.822675	0.828727

We can see that the highest result was achieved when using a var_smoothing parameter with a value of 0.001

In [91]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
gnb = GaussianNB(var_smoothing=0.001)
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8852723604572966
Precision: 0.9004646979463349
Recall: 0.9694964493221433
```

We can see that after hypertuning the `var_smoothing` our model's performance improved significantly.

4.2 Naive Bayes with HashingVectorizer

4.2.1 Split the dataset into test and train

In [92]:
```
hashingVectorizer = HashingVectorizer(n_features=20)
vector = hashingVectorizer.transform(df.tweet)
```

In [93]:
```
X = hashingVectorizer.transform(df.tweet.values).toarray()
y = df['is_hate_speech'].astype(int)
```

In [94]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
```

4.2.3 Create the Gaussian Naive Bayes

In [95]:
```
gnb = GaussianNB()
```

4.2.4 Fit the model

In [96]:
```
gnb.fit(X_train, y_train)
```

Out[96]:
```
GaussianNB()
```

4.2.5 Predict

In [97]:
```
y_pred = gnb.predict(X_test)
```

4.2.6 Calculate the metrics (accuracy, precision, recall)

In [98]:
```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.828782784129119
Precision: 0.840033153750518
Recall: 0.9814396384764364
```

4.2.7 Hyperparameter tuning

We will try to tune the var_smoothing parameter. This variable artificially adds a user-defined value to the distribution's variance (whose default value is derived from the training data set). This essentially widens (or "smooths") the curve and accounts for more samples that are further away from the distribution mean.

In [99]:
```
parameters = {
    'var_smoothing': [1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14, 1e-15]
}
clf = GridSearchCV(nb_classifier, parameters, cv=5, verbose=2)
clf.fit(X, y)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=   0.1s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=   0.0s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=   0.0s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=   0.0s
[CV] var_smoothing=0.01 ..............................................
[CV] ............................... var_smoothing=0.01, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[CV] ............................. var_smoothing=0.01, total=   0.0s
[CV] var_smoothing=0.001 ..........................................
[CV] ............................. var_smoothing=0.001, total=   0.0s
[CV] var_smoothing=0.001 ..........................................
[CV] ............................. var_smoothing=0.001, total=   0.0s
[CV] var_smoothing=0.001 ..........................................
[CV] ............................. var_smoothing=0.001, total=   0.0s
[CV] var_smoothing=0.001 ..........................................
[CV] ............................. var_smoothing=0.001, total=   0.0s
[CV] var_smoothing=0.001 ..........................................
[CV] ............................. var_smoothing=0.001, total=   0.0s
[CV] var_smoothing=0.0001 .........................................
[CV] ............................ var_smoothing=0.0001, total=   0.0s
[CV] var_smoothing=0.0001 .........................................
[CV] ............................ var_smoothing=0.0001, total=   0.0s
[CV] var_smoothing=0.0001 .........................................
[CV] ............................ var_smoothing=0.0001, total=   0.0s
[CV] var_smoothing=0.0001 .........................................
[CV] ............................ var_smoothing=0.0001, total=   0.0s
[CV] var_smoothing=0.0001 .........................................
[CV] ............................ var_smoothing=0.0001, total=   0.0s
[CV] var_smoothing=1e-05 ..........................................
[CV] ............................. var_smoothing=1e-05, total=   0.0s
[CV] var_smoothing=1e-05 ..........................................
[CV] ............................. var_smoothing=1e-05, total=   0.0s
[CV] var_smoothing=1e-05 ..........................................
[CV] ............................. var_smoothing=1e-05, total=   0.0s
[CV] var_smoothing=1e-05 ..........................................
[CV] ............................. var_smoothing=1e-05, total=   0.0s
[CV] var_smoothing=1e-05 ..........................................
[CV] ............................. var_smoothing=1e-05, total=   0.0s
[CV] var_smoothing=1e-06 ..........................................
[CV] ............................. var_smoothing=1e-06, total=   0.0s
[CV] var_smoothing=1e-06 ..........................................
[CV] ............................. var_smoothing=1e-06, total=   0.0s
[CV] var_smoothing=1e-06 ..........................................
[CV] ............................. var_smoothing=1e-06, total=   0.0s
[CV] var_smoothing=1e-06 ..........................................
[CV] ............................. var_smoothing=1e-06, total=   0.0s
[CV] var_smoothing=1e-06 ..........................................
[CV] ............................. var_smoothing=1e-06, total=   0.0s
[CV] var_smoothing=1e-07 ..........................................
[CV] ............................. var_smoothing=1e-07, total=   0.0s
[CV] var_smoothing=1e-07 ..........................................
[CV] ............................. var_smoothing=1e-07, total=   0.0s
```

```
[CV] var_smoothing=1e-07 ..........................................
[CV] ............................ var_smoothing=1e-07, total=   0.0s
[CV] var_smoothing=1e-07 ..........................................
[CV] ............................ var_smoothing=1e-07, total=   0.0s
[CV] var_smoothing=1e-07 ..........................................
[CV] ............................ var_smoothing=1e-07, total=   0.0s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=   0.0s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=   0.0s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=   0.0s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=   0.0s
[CV] var_smoothing=1e-08 ..........................................
[CV] ............................ var_smoothing=1e-08, total=   0.0s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=   0.0s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=   0.0s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=   0.0s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=   0.0s
[CV] var_smoothing=1e-09 ..........................................
[CV] ............................ var_smoothing=1e-09, total=   0.0s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=   0.0s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=   0.0s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=   0.0s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=   0.0s
[CV] var_smoothing=1e-10 ..........................................
[CV] ............................ var_smoothing=1e-10, total=   0.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=   0.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=   0.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=   0.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=   0.0s
[CV] var_smoothing=1e-11 ..........................................
[CV] ............................ var_smoothing=1e-11, total=   0.0s
[CV] var_smoothing=1e-12 ..........................................
```

```
[CV] ............................ var_smoothing=1e-12, total=   0.0s
[CV] var_smoothing=1e-12 .....................................
[CV] ............................ var_smoothing=1e-12, total=   0.0s
[CV] var_smoothing=1e-12 .....................................
[CV] ............................ var_smoothing=1e-12, total=   0.0s
[CV] var_smoothing=1e-12 .....................................
[CV] ............................ var_smoothing=1e-12, total=   0.0s
[CV] var_smoothing=1e-12 .....................................
[CV] ............................ var_smoothing=1e-12, total=   0.0s
[CV] var_smoothing=1e-13 .....................................
[CV] ............................ var_smoothing=1e-13, total=   0.0s
[CV] var_smoothing=1e-13 .....................................
[CV] ............................ var_smoothing=1e-13, total=   0.0s
[CV] var_smoothing=1e-13 .....................................
[CV] ............................ var_smoothing=1e-13, total=   0.0s
[CV] var_smoothing=1e-13 .....................................
[CV] ............................ var_smoothing=1e-13, total=   0.0s
[CV] var_smoothing=1e-13 .....................................
[CV] ............................ var_smoothing=1e-13, total=   0.0s
[CV] var_smoothing=1e-14 .....................................
[CV] ............................ var_smoothing=1e-14, total=   0.0s
[CV] var_smoothing=1e-14 .....................................
[CV] ............................ var_smoothing=1e-14, total=   0.0s
[CV] var_smoothing=1e-14 .....................................
[CV] ............................ var_smoothing=1e-14, total=   0.0s
[CV] var_smoothing=1e-14 .....................................
[CV] ............................ var_smoothing=1e-14, total=   0.0s
[CV] var_smoothing=1e-14 .....................................
[CV] ............................ var_smoothing=1e-14, total=   0.0s
[CV] var_smoothing=1e-15 .....................................
[CV] ............................ var_smoothing=1e-15, total=   0.0s
[CV] var_smoothing=1e-15 .....................................
[CV] ............................ var_smoothing=1e-15, total=   0.0s
[CV] var_smoothing=1e-15 .....................................
[CV] ............................ var_smoothing=1e-15, total=   0.0s
[CV] var_smoothing=1e-15 .....................................
[CV] ............................ var_smoothing=1e-15, total=   0.0s
[CV] var_smoothing=1e-15 .....................................
[CV] ............................ var_smoothing=1e-15, total=   0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    1.6s finished
```

Out[99]:

```
GridSearchCV(cv=5, estimator=GaussianNB(),
             param_grid={'var_smoothing': [0.01, 0.001, 0.0001, 1e-05, 1e-06,
                                           1e-07, 1e-08, 1e-09, 1e-10, 1e-11,
                                           1e-12, 1e-13, 1e-14, 1e-15]},
             verbose=2)
```

In [100…
```
pd.DataFrame(clf.cv_results_).sort_values(by=['rank_test_score', 'mean_fit_time']).head()
```

Out[100…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_var_smoothing	params	split0_test_score	split1_test_score	split2_test_score
0	0.030201	0.004070	0.006997	0.002531	0.01	{'var_smoothing': 0.01}	0.825701	0.824491	0.829131
1	0.026600	0.000802	0.005602	0.000492	0.001	{'var_smoothing': 0.001}	0.825096	0.823078	0.828525
2	0.022804	0.000759	0.004801	0.001327	0.0001	{'var_smoothing': 0.0001}	0.825096	0.822877	0.828727
7	0.009139	0.007476	0.006263	0.007670	1e-09	{'var_smoothing': 1e-09}	0.825096	0.822675	0.828727
9	0.013457	0.002956	0.000000	0.000000	1e-11	{'var_smoothing': 1e-11}	0.825096	0.822675	0.828727

We can see that the highest result was achieved when using a var_smoothing parameter with a value of 0.01

In [102…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=46)
gnb = GaussianNB(var_smoothing=0.01)
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8282447881640888
Precision: 0.8390075809786354
Recall: 0.9824080051646223
```

We can see that after hypertuning the `var_smoothing` our model's precision and recall improved slightly.

5. Conclusions

We can see that from the three types of classifications we tried, the one with the highest accuracy was the SVC with a CountVectorizer so this is going to be the one we use. That is not very surprising because SVM Classifiers offer good accuracy and perform faster prediction compared to the Naive Bayes algorithm. They also use less memory because they use a subset of training points in the decision phase.

References

How to Encode Text Data for Machine Learning with scikit-learn (article)- https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/
Text Classification (article) - https://monkeylearn.com/text-classification/

Credits

The following Jupyter notebook was made by Karina Kozarova as part of the Artificial Intelligence specialization at Fontys University of Applied Sciences