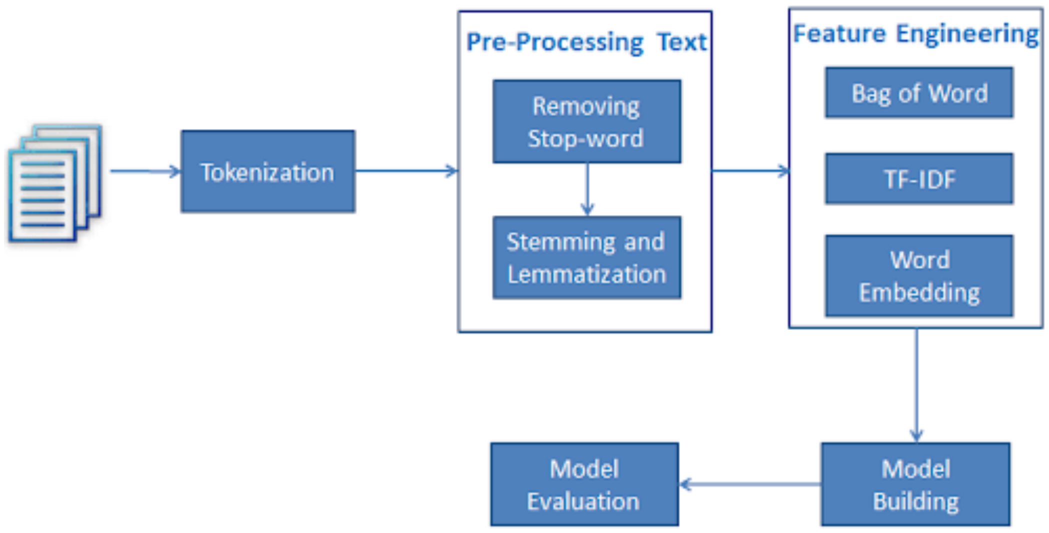
**IDS 566: FINAL PROJECT**

# TEXT CLASSIFICATION



***Submitted by:***

**Raja Amlan (651045470)**

**Anshu Pathak (670161686)**

**Prashant Chaudhary (678226947)**

## Purpose

To classify the documents in 20Newsgroup dataset subject to their corresponding newsgroup by implementing different models on the dataset with the optimal parameters to achieve maximum accuracy possible.

## About the dataset

The 20 Newsgroup dataset contains around 18000 posts on 20 topics from 20 newsgroups. This is split into two parts, one for training and the other for testing. The splitting is done based on the messages posted before and after a specific date.

## Setting up the notebook

This data set is in-built in scikit, so we don’t need to download it explicitly.

i. Open Anaconda Navigator> Launch Jupyter Notebook
ii. Select Open> Run Final_code.ipynb

## Data Preparation and Transformation

The data was extracted using the sklearn.datasets package. Since the dataset is a series of words, the text representations need to be converted into numerical representations.

To do this we used the “Bag of Words” method, where each file is split into single word and then number of occurrences of each word in each file is counted, so that every word takes up a unique value in the dictionary.

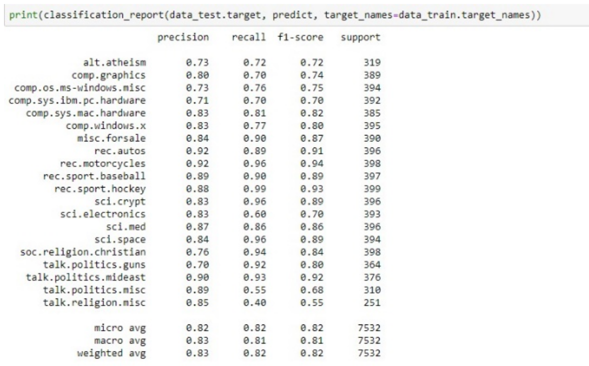
Further, to avoid the problem of large files being assigned higher weights after “Bag of Words”, we use Term Frequency-Inverse Document Frequency (TF-IDF). This does not alter the dimension of the data. Basically, TF-IDF takes the relative frequency of each word in a certain file into account when constructing the feature vector.

After the above steps, we removed stop words and implemented text normalization technique, stemming, to further prepare data for algorithmic processing

## Models Implemented

### *Support Vector Machine:*

Support Vector Machine is based on the Structural Risk Minimization. It is an algorithm which determines the best decision boundary between vectors that belong to a same group and that belong to some different group. As SVM works on vectors, we need to transform the text to vectors before implementation. Since SVM uses overfitting protection, it works well for text categorization as they have the potential to handle large feature spaces.



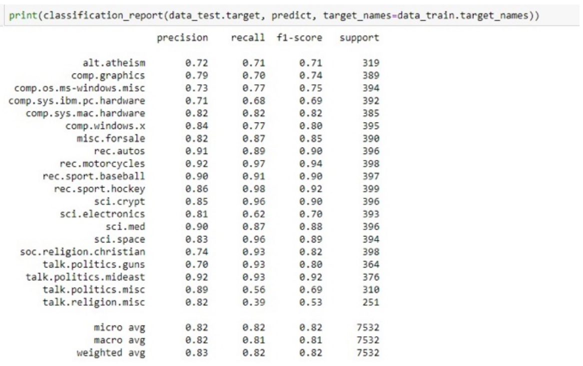
```
print(classification_report(data_test.target, predict, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.73      0.72      0.72       319
           comp.graphics       0.80      0.70      0.74       389
 comp.os.ms-windows.misc       0.73      0.76      0.75       394
comp.sys.ibm.pc.hardware       0.71      0.70      0.70       392
   comp.sys.mac.hardware       0.83      0.81      0.82       385
          comp.windows.x       0.83      0.77      0.80       395
            misc.forsale       0.84      0.90      0.87       390
               rec.autos       0.92      0.89      0.91       396
         rec.motorcycles       0.92      0.96      0.94       398
      rec.sport.baseball       0.89      0.90      0.89       397
        rec.sport.hockey       0.88      0.99      0.93       399
               sci.crypt       0.83      0.96      0.89       396
         sci.electronics       0.83      0.60      0.70       393
                 sci.med       0.87      0.86      0.86       396
               sci.space       0.84      0.96      0.89       394
  soc.religion.christian       0.76      0.94      0.84       398
      talk.politics.guns       0.70      0.92      0.80       364
   talk.politics.mideast       0.90      0.93      0.92       376
      talk.politics.misc       0.89      0.55      0.68       310
      talk.religion.misc       0.85      0.40      0.55       251

               micro avg       0.82      0.82      0.82      7532
               macro avg       0.83      0.81      0.81      7532
            weighted avg       0.83      0.82      0.82      7532
```

After Vectorizing



```
print(classification_report(data_test.target, predict, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.72      0.71      0.71       319
           comp.graphics       0.79      0.70      0.74       389
 comp.os.ms-windows.misc       0.73      0.77      0.75       394
comp.sys.ibm.pc.hardware       0.71      0.68      0.69       392
   comp.sys.mac.hardware       0.82      0.82      0.82       385
          comp.windows.x       0.84      0.77      0.80       395
            misc.forsale       0.82      0.87      0.85       390
               rec.autos       0.91      0.89      0.90       396
         rec.motorcycles       0.92      0.97      0.94       398
      rec.sport.baseball       0.90      0.91      0.90       397
        rec.sport.hockey       0.86      0.98      0.92       399
               sci.crypt       0.85      0.96      0.90       396
         sci.electronics       0.81      0.62      0.70       393
                 sci.med       0.90      0.87      0.88       396
               sci.space       0.83      0.96      0.89       394
  soc.religion.christian       0.74      0.93      0.82       398
      talk.politics.guns       0.70      0.93      0.80       364
   talk.politics.mideast       0.92      0.93      0.92       376
      talk.politics.misc       0.89      0.56      0.69       310
      talk.religion.misc       0.82      0.39      0.53       251

               micro avg       0.82      0.82      0.82      7532
               macro avg       0.82      0.81      0.81      7532
            weighted avg       0.83      0.82      0.82      7532
```

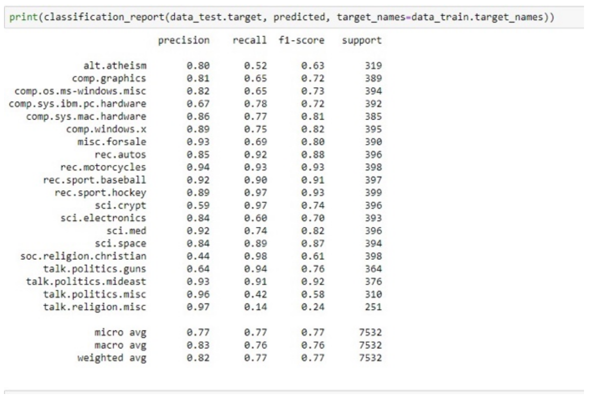
After removing stop words

**<u>Key Observations:</u>**

- The above two images show the results on the test data after applying TF-IDF and removing stop words respectively.
- Overall, majority of the categories have an accuracy of more that 0.80 with lowest in the range of 0.50 and 0.55 for the talk.religion.misc category.

*<u>Naïve Bayes:</u>*

Naïve Bayes methods are supervised learning algorithms which is based on the Bayes theorem with “naïve” assumption of conditional independence between every pair of features given the value of class variable. Naïve Bayes classifiers work well with document classification. They require small amount of training data to estimate the parameters. They are fast compared to more sophisticated methods.



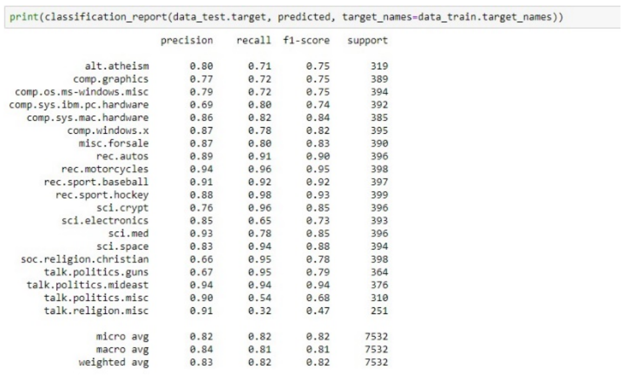
```
print(classification_report(data_test.target, predicted, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.80      0.52      0.63       319
           comp.graphics       0.81      0.65      0.72       389
 comp.os.ms-windows.misc       0.82      0.65      0.73       394
comp.sys.ibm.pc.hardware       0.67      0.78      0.72       392
   comp.sys.mac.hardware       0.86      0.77      0.81       385
          comp.windows.x       0.89      0.75      0.82       395
            misc.forsale       0.93      0.69      0.80       390
               rec.autos       0.85      0.92      0.88       396
         rec.motorcycles       0.94      0.93      0.93       398
      rec.sport.baseball       0.92      0.90      0.91       397
        rec.sport.hockey       0.89      0.97      0.93       399
               sci.crypt       0.59      0.97      0.74       396
         sci.electronics       0.84      0.60      0.70       393
                 sci.med       0.92      0.74      0.82       396
               sci.space       0.84      0.89      0.87       394
  soc.religion.christian       0.44      0.98      0.61       398
      talk.politics.guns       0.64      0.94      0.76       364
   talk.politics.mideast       0.93      0.91      0.92       376
      talk.politics.misc       0.96      0.42      0.58       310
      talk.religion.misc       0.97      0.14      0.24       251

               micro avg       0.77      0.77      0.77      7532
               macro avg       0.83      0.76      0.76      7532
            weighted avg       0.82      0.77      0.77      7532
```

After Vectorization



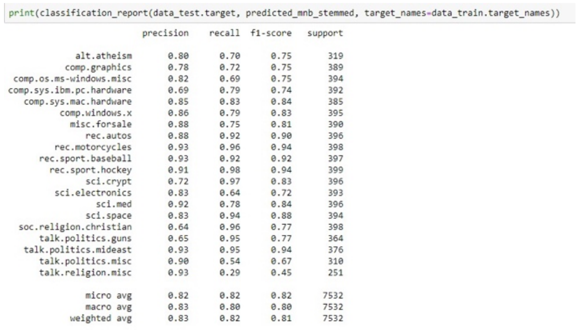
```
print(classification_report(data_test.target, predicted, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.80      0.71      0.75       319
           comp.graphics       0.77      0.72      0.75       389
 comp.os.ms-windows.misc       0.79      0.72      0.75       394
comp.sys.ibm.pc.hardware       0.69      0.80      0.74       392
   comp.sys.mac.hardware       0.86      0.82      0.84       385
          comp.windows.x       0.87      0.78      0.82       395
            misc.forsale       0.87      0.80      0.83       390
               rec.autos       0.89      0.91      0.90       396
         rec.motorcycles       0.94      0.96      0.95       398
      rec.sport.baseball       0.91      0.92      0.92       397
        rec.sport.hockey       0.88      0.98      0.93       399
               sci.crypt       0.76      0.96      0.85       396
         sci.electronics       0.85      0.65      0.73       393
                 sci.med       0.93      0.78      0.85       396
               sci.space       0.83      0.94      0.88       394
  soc.religion.christian       0.66      0.95      0.78       398
      talk.politics.guns       0.67      0.95      0.79       364
   talk.politics.mideast       0.94      0.94      0.94       376
      talk.politics.misc       0.90      0.54      0.68       310
      talk.religion.misc       0.91      0.32      0.47       251

               micro avg       0.82      0.82      0.82      7532
               macro avg       0.84      0.81      0.81      7532
            weighted avg       0.83      0.82      0.82      7532
```

After changing the parameter



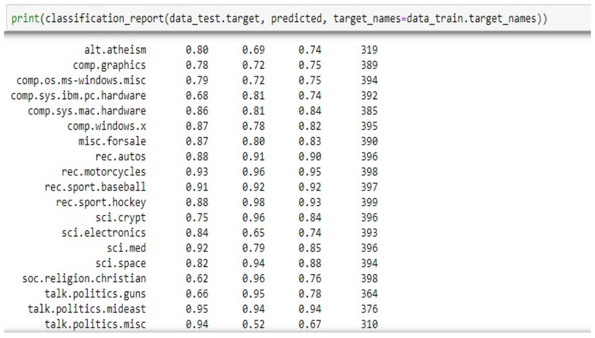
```
print(classification_report(data_test.target, predicted_mnb_stemmed, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.80      0.70      0.75       319
           comp.graphics       0.78      0.72      0.75       389
 comp.os.ms-windows.misc       0.82      0.69      0.75       394
comp.sys.ibm.pc.hardware       0.69      0.79      0.74       392
   comp.sys.mac.hardware       0.85      0.83      0.84       385
          comp.windows.x       0.86      0.79      0.83       395
            misc.forsale       0.88      0.75      0.81       390
               rec.autos       0.88      0.92      0.90       396
         rec.motorcycles       0.93      0.96      0.94       398
      rec.sport.baseball       0.93      0.92      0.92       397
        rec.sport.hockey       0.91      0.98      0.94       399
               sci.crypt       0.72      0.97      0.83       396
         sci.electronics       0.83      0.64      0.72       393
                 sci.med       0.92      0.78      0.84       396
               sci.space       0.83      0.94      0.88       394
  soc.religion.christian       0.64      0.96      0.77       398
      talk.politics.guns       0.65      0.95      0.77       364
   talk.politics.mideast       0.93      0.95      0.94       376
      talk.politics.misc       0.90      0.54      0.67       310
      talk.religion.misc       0.93      0.29      0.45       251

               micro avg       0.82      0.82      0.82      7532
               macro avg       0.83      0.80      0.80      7532
            weighted avg       0.83      0.82      0.81      7532
```

After stemming



```
print(classification_report(data_test.target, predicted, target_names=data_train.target_names))

             alt.atheism       0.80      0.69      0.74       319
           comp.graphics       0.78      0.72      0.75       389
 comp.os.ms-windows.misc       0.79      0.72      0.75       394
comp.sys.ibm.pc.hardware       0.68      0.81      0.74       392
   comp.sys.mac.hardware       0.86      0.81      0.84       385
          comp.windows.x       0.87      0.78      0.82       395
            misc.forsale       0.87      0.80      0.83       390
               rec.autos       0.88      0.91      0.90       396
         rec.motorcycles       0.93      0.96      0.95       398
      rec.sport.baseball       0.91      0.92      0.92       397
        rec.sport.hockey       0.88      0.98      0.93       399
               sci.crypt       0.75      0.96      0.84       396
         sci.electronics       0.84      0.65      0.74       393
                 sci.med       0.92      0.79      0.85       396
               sci.space       0.82      0.94      0.88       394
  soc.religion.christian       0.62      0.96      0.76       398
      talk.politics.guns       0.66      0.95      0.78       364
   talk.politics.mideast       0.95      0.94      0.94       376
      talk.politics.misc       0.94      0.52      0.67       310
```

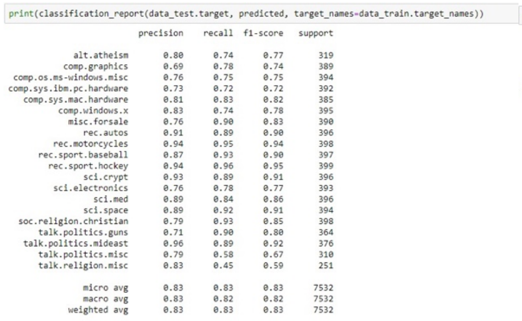
After removing stop words

**Key Observations:**

- The accuracy of "talk" type religion is lowest in all the cases
- The above four images are the results of the Naïve Bayes model on the test data before and after implementation of TF-IDF, removal of stop words, stemming and changing the model parameter (i.e. setting fit_prior=False).
- As inferred from the above images, there is a slight improvement in the accuracy (from 0.77 to 0.82) after changing the parameter and removal of stop words.
- Implementation of stemming after removing stop-words does not bring about any noticeable change in the accuracy.

*Logistic Regression:*

Logistic regression is a linear method, but the predictions are transformed using the logistic function. It uses maximum-likelihood estimation to predict the relevant quotients. The basic logic behind logistic regression is that we linearly map our input *X* into *N* different outputs. Multi class classification is implemented by training multiple logistic regression classifiers, one for each of the K classes in the training dataset.



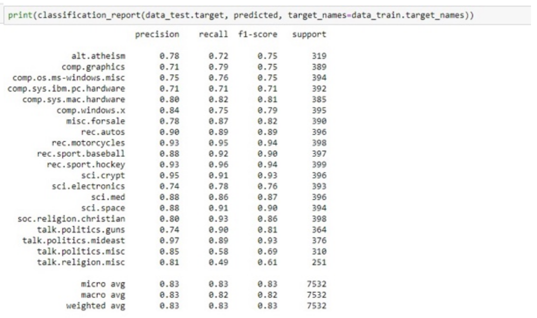
```
print(classification_report(data_test.target, predicted, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.80      0.74      0.77       319
           comp.graphics       0.69      0.78      0.74       389
 comp.os.ms-windows.misc       0.76      0.75      0.75       394
comp.sys.ibm.pc.hardware       0.73      0.72      0.72       392
   comp.sys.mac.hardware       0.81      0.83      0.82       385
          comp.windows.x       0.83      0.74      0.78       395
            misc.forsale       0.76      0.90      0.83       390
               rec.autos       0.91      0.89      0.90       396
         rec.motorcycles       0.94      0.95      0.94       398
      rec.sport.baseball       0.87      0.93      0.90       397
        rec.sport.hockey       0.94      0.96      0.95       399
               sci.crypt       0.93      0.89      0.91       396
         sci.electronics       0.76      0.78      0.77       393
                 sci.med       0.89      0.84      0.86       396
               sci.space       0.89      0.92      0.91       394
  soc.religion.christian       0.79      0.93      0.85       398
      talk.politics.guns       0.71      0.90      0.80       364
   talk.politics.mideast       0.96      0.89      0.92       376
      talk.politics.misc       0.79      0.58      0.67       310
      talk.religion.misc       0.83      0.45      0.59       251

               micro avg       0.83      0.83      0.83      7532
               macro avg       0.83      0.82      0.82      7532
            weighted avg       0.83      0.83      0.83      7532
```

After Vectorization



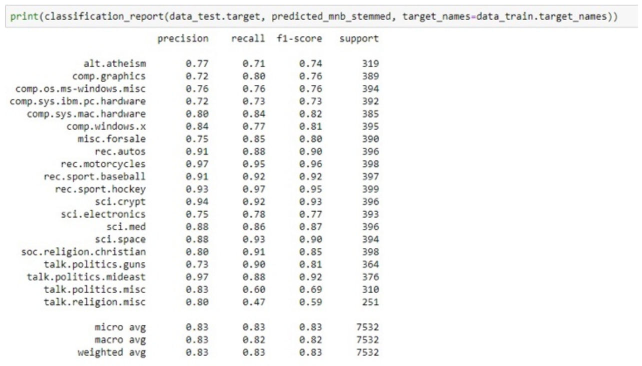
```
print(classification_report(data_test.target, predicted, target_names=data_train.target_names))

                          precision    recall  f1-score   support

             alt.atheism       0.78      0.72      0.75       319
           comp.graphics       0.71      0.79      0.75       389
 comp.os.ms-windows.misc       0.75      0.76      0.75       394
comp.sys.ibm.pc.hardware       0.71      0.71      0.71       392
   comp.sys.mac.hardware       0.80      0.82      0.81       385
          comp.windows.x       0.84      0.75      0.79       395
            misc.forsale       0.78      0.87      0.82       390
               rec.autos       0.90      0.89      0.89       396
         rec.motorcycles       0.93      0.95      0.94       398
      rec.sport.baseball       0.88      0.92      0.90       397
        rec.sport.hockey       0.93      0.96      0.94       399
               sci.crypt       0.95      0.91      0.93       396
         sci.electronics       0.74      0.78      0.76       393
                 sci.med       0.88      0.86      0.87       396
               sci.space       0.88      0.91      0.90       394
  soc.religion.christian       0.80      0.93      0.86       398
      talk.politics.guns       0.74      0.90      0.81       364
   talk.politics.mideast       0.97      0.89      0.93       376
      talk.politics.misc       0.85      0.58      0.69       310
      talk.religion.misc       0.81      0.49      0.61       251

               micro avg       0.83      0.83      0.83      7532
               macro avg       0.83      0.82      0.82      7532
            weighted avg       0.83      0.83      0.83      7532
```

After removing stop words



```
print(classification_report(data_test.target, predicted_mnb_stemmed, target_names=data_train.target_names))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.77      0.71      0.74       319
           comp.graphics       0.72      0.80      0.76       389
 comp.os.ms-windows.misc       0.76      0.76      0.76       394
comp.sys.ibm.pc.hardware       0.72      0.73      0.73       392
   comp.sys.mac.hardware       0.80      0.84      0.82       385
          comp.windows.x       0.84      0.77      0.81       395
            misc.forsale       0.75      0.85      0.80       390
               rec.autos       0.91      0.88      0.90       396
         rec.motorcycles       0.97      0.95      0.96       398
      rec.sport.baseball       0.91      0.92      0.92       397
        rec.sport.hockey       0.93      0.97      0.95       399
               sci.crypt       0.94      0.92      0.93       396
         sci.electronics       0.75      0.78      0.77       393
                 sci.med       0.88      0.86      0.87       396
               sci.space       0.88      0.93      0.90       394
  soc.religion.christian       0.80      0.91      0.85       398
      talk.politics.guns       0.73      0.90      0.81       364
   talk.politics.mideast       0.97      0.88      0.92       376
      talk.politics.misc       0.83      0.60      0.69       310
      talk.religion.misc       0.80      0.47      0.59       251

               micro avg       0.83      0.83      0.83      7532
               macro avg       0.83      0.82      0.82      7532
            weighted avg       0.83      0.83      0.83      7532
```

After Stemming

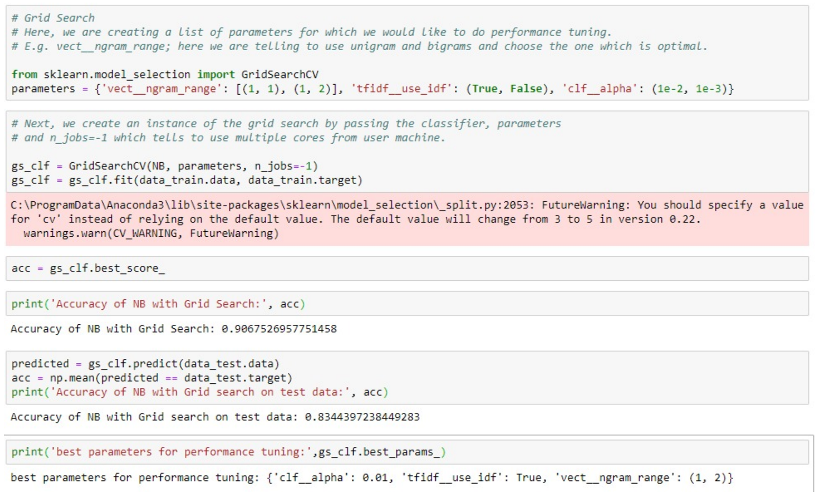
## **Key Observations:**

- The highest accuracy was obtained for "rec" type categories in all the cases.
- The lowest accuracy was obtained for "talk" religion type category in all the cases
- Also, we can see that most of the categories have an accuracy of more than 0.80 after applying TF-IDF, removing stop words and stemming

### *Grid Search:*

Grid search performs hyper-parameter optimization, a method to find the best combination of hyper-parameters for a given model and test dataset. Each of the combination of the parameters, corresponding to a single model, lie on a point of "grid". The aim here is to each of the models and evaluate them after which the best one is selected based on performance.

We have implemented grid search for Naïve Bayes and SVM, which resulted in a better accuracy. The following images show the codes and the accuracy of grid search implementation for both the models.



```
# Grid Search
# Here, we are creating a list of parameters for which we would like to do performance tuning.
# E.g. vect__ngram_range; here we are telling to use unigram and bigrams and choose the one which is optimal.

from sklearn.model_selection import GridSearchCV
parameters = {'vect__ngram_range': [(1, 1), (1, 2)], 'tfidf__use_idf': (True, False), 'clf__alpha': (1e-2, 1e-3)}
```

```
# Next, we create an instance of the grid search by passing the classifier, parameters
# and n_jobs=-1 which tells to use multiple cores from user machine.

gs_clf = GridSearchCV(NB, parameters, n_jobs=-1)
gs_clf = gs_clf.fit(data_train.data, data_train.target)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarning: You should specify a value
for 'cv' instead of relying on the default value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
acc = gs_clf.best_score_
```

```
print('Accuracy of NB with Grid Search:', acc)
```

```
Accuracy of NB with Grid Search: 0.9067526957751458
```

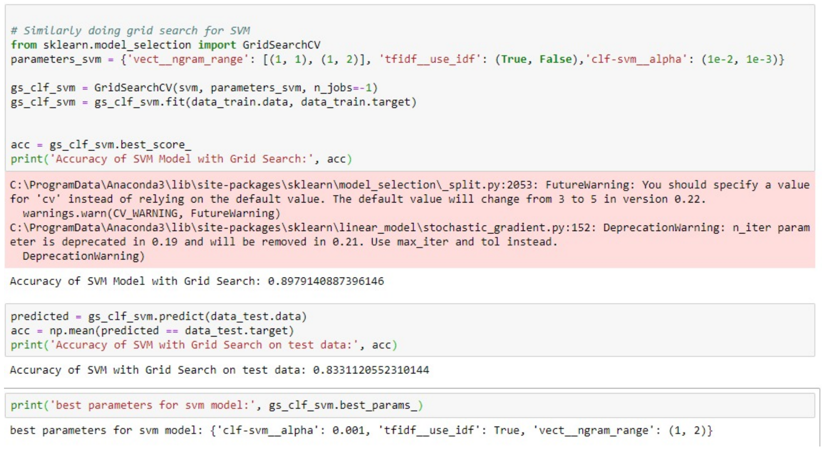
```
predicted = gs_clf.predict(data_test.data)
acc = np.mean(predicted == data_test.target)
print('Accuracy of NB with Grid search on test data:', acc)
```

```
Accuracy of NB with Grid search on test data: 0.8344397238449283
```

```
print('best parameters for performance tuning:',gs_clf.best_params_)
```

```
best parameters for performance tuning: {'clf__alpha': 0.01, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
```

*Naive Bayes*



```
# Similarly doing grid search for SVM
from sklearn.model_selection import GridSearchCV
parameters_svm = {'vect__ngram_range': [(1, 1), (1, 2)], 'tfidf__use_idf': (True, False),'clf-svm__alpha': (1e-2, 1e-3)}

gs_clf_svm = GridSearchCV(svm, parameters_svm, n_jobs=-1)
gs_clf_svm = gs_clf_svm.fit(data_train.data, data_train.target)


acc = gs_clf_svm.best_score_
print('Accuracy of SVM Model with Grid Search:', acc)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarning: You should specify a value
for 'cv' instead of relying on the default value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter param
eter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
Accuracy of SVM Model with Grid Search: 0.8979140887396146
```

```
predicted = gs_clf_svm.predict(data_test.data)
acc = np.mean(predicted == data_test.target)
print('Accuracy of SVM with Grid Search on test data:', acc)
```

```
Accuracy of SVM with Grid Search on test data: 0.8331120552310144
```

```
print('best parameters for svm model:', gs_clf_svm.best_params_)
```

```
best parameters for svm model: {'clf-svm__alpha': 0.001, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
```

*SVM*

**Key Takeaway**

- The only difference in the best parameters of SVM and Naïve Bayes model is, the value of clf-svm_alpha for SVM model is 0.001 whereas the  the value of clf-svm_alpha for Naïve Bayes model is 0.01

**Final Results**

**Train**

| Model | Count Vectorization | After Removing Stopwords | After Stemming | After Changing Parameters | Grid Search |
|---|---|---|---|---|---|
| Naïve Bayes | 0.933 | 0.957 | 0.955 | 0.963 | 0.907 |
| Logistic | 0.97 | 0.975 | 0.97 | n/a | n/a |
| SVM | 0.967 | 0.966 | 0.959 | n/a | 0.898 |

**Test**

| Model | Count Vectorization | After Removing Stop-words | After Stemming | After Changing Parameters | Grid Search |
|---|---|---|---|---|---|
| Naïve Bayes | 0.774 | 0.817 | 0.817 | 0.821 | 0.834 |
| Logistic | 0.828 | 0.83 | 0.832 | n/a | n/a |
| SVM | 0.824 | 0.822 | 0.819 | n/a | 0.833 |

As seen in the above table, Naïve Bayes model gives the best accuracy of 0.834 after implementing grid search. Logistic regression gives a maximum accuracy of 0.832 after removing stop-words and implementing stemming. Finally, SVM model gives a maximum accuracy of 0.833 with Grid Search. Overall, we conclude that Naïve Bayes with Grid Search proves to the best model out of the three.

## Learnings from the project

The project helped us gain insights on how different models can be implemented for text classification, and how tweaking the parameters of the model can help us improve the accuracy. Additionally, we gained an understanding as to how grid search can be implemented for various models, thus further improving the outcomes. Overall, the project was a great learning experience for all of us, as it helped us gain in-depth knowledge as to how machine learning algorithms can be used for text classification.

## References:

https://stackoverflow.com/questions/19335165/what-is-the-difference-between-cross-validation-and-grid-search

https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4

https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a