Importing Libraries

In [18]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.utils import shuffle
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import heapq
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

import nltk

### Run these two lines if you have not installed the following nltk packages ###

# nltk.download('punkt')
# nltk.download('stopwords')
```

Exercise - 0) Preprocessing Text Data

Function to load and merge dataset with target

In [19]:

```
def twenty_newsgroup_to_df():

    categories = ['sci.med', 'comp.graphics']
    newsgroups_train = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quot

    df = pd.DataFrame([newsgroups_train.data, newsgroups_train.target.tolist()]).T
    df.columns = ['text', 'target']

    targets = pd.DataFrame( newsgroups_train.target_names)
    targets.columns=['title']

    out = pd.merge(df, targets, left_on='target', right_index=True)
    df = shuffle(out)
    return df
```

Function to do preprocessing of textual data to remove punctuation, stop-words

In [20]:

```
def do_pre_processing(sentence):

    stop_words = set(stopwords.words('english') + list(string.punctuation))

    word_tokens = word_tokenize(sentence)

    filtered_sentence = [token.lower() for token in word_tokens if token.lower() not in sto
    return filtered_sentence
```

Function to make vocabulary of unique words from dataset

In [21]:

```
def make_vocab_with_features(df,features_to_make):
    corporus=[]

    ### doing preprocessing on dataset
    for index, row in df.iterrows():
        corporus.append(do_pre_processing(row['text']))


    wordfreq = {}
    for sentence in corporus:
        for token in sentence:
            if token not in wordfreq.keys():
                wordfreq[token] = 1
            else:
                wordfreq[token] += 1
    most_freq = heapq.nlargest(features_to_make, wordfreq, key=wordfreq.get)
    return corporus, most_freq
```

Function to convert dataset to bag of word representation

In [22]:

```
def convert_to_bag_of_words(corporus,most_freq):
    sentence_vectors = []
    for sentence_tokens in corporus:
        sent_vec = []
        for token in most_freq:
            if token in sentence_tokens:
                sent_vec.append(1)
            else:
                sent_vec.append(0)
        sentence_vectors.append(sent_vec)

    sentence_vectors = np.asarray(sentence_vectors)
    return sentence_vectors
```

Function to convert dataset to TF-IDF word representation

In [23]:

```
def convert_to_tf_idf(corpus,most_freq):
    word_idf_values = {}
    for token in most_freq:
        doc_containing_word = 0
        for document in corpus:
            if token in document:
                doc_containing_word += 1
        word_idf_values[token] = np.log(len(corpus)/(1 + doc_containing_word))

    word_tf_values = {}
    for token in most_freq:
        sent_tf_vector = []
        for document in corpus:
            doc_freq = 0
            for word in document:
                if token == word:
                    doc_freq += 1

            if len(document)==0:
                word_tf = doc_freq
            else:
                word_tf = doc_freq/(len(document))

            sent_tf_vector.append(word_tf)
        word_tf_values[token] = sent_tf_vector

    tfidf_values = []
    for token in word_tf_values.keys():
        tfidf_sentences = []
        for tf_sentence in word_tf_values[token]:
            tf_idf_score = tf_sentence * word_idf_values[token]
            tfidf_sentences.append(tf_idf_score)
        tfidf_values.append(tfidf_sentences)

    tf_idf_model = np.asarray(tfidf_values)
    tf_idf_model = np.transpose(tf_idf_model)
    return tf_idf_model
```

Function to split the dataset

In [24]:

```
def split(X,Y):

    Xtrain = X[:1400]
    Ytrain = Y[:1400]

    Xval = X[1400:1700]
    Yval = Y[1400:1700]

    Xtest = X[1700:]
    Ytest = Y[1700:]


    return Xtrain, Ytrain, Xval, Yval, Xtest, Ytest
```

Main Function which implements naive bayes algorithm

In [25]:

```
### This function uses naive baise algorithm to learn the probabilities of each unique word
### do prediction respectively

def main(Xtrain, Ytrain, Xval, Yval, Xtest, Ytest):

    ### merge feature array and target vector
    X_df = pd.DataFrame(Xtrain)
    y_df = pd.DataFrame(Ytrain)
    full_dataset_train = X_df.copy()
    full_dataset_train['target'] = y_df

    ### seperate the dataset into respective classes
    med = full_dataset_train[full_dataset_train['target'] == 1]
    graphic = full_dataset_train[full_dataset_train['target'] == 0]

    ### find probability of each target class
    med_prob = med.shape[0]/X_df.shape[0]
    graphic_prob = graphic.shape[0]/X_df.shape[0]


    ### Calculate total words in each target class features
    total_words_med=0
    total_words_grap=0
    for i in range(X_df.shape[1]):
        counts_med = med[i].sum()
        total_words_med+= counts_med
        counts_graph = graphic[i].sum()
        total_words_grap+= counts_graph

    ### Calculate probabilities of each word in each target class features
    each_word_prob_med = []
    each_word_prob_graphics = []

    for i in range(X_df.shape[1]):
        col_count_med = med[i].sum()
        prob_for_col_med = (col_count_med + 1)/(total_words_med + X_df.shape[1])
        each_word_prob_med.append(prob_for_col_med)

        col_count_graph = graphic[i].sum()
        prob_for_col_graph = (col_count_graph + 1)/(total_words_grap + X_df.shape[1])
        each_word_prob_graphics.append(prob_for_col_graph)

    ### Calculate the probabilty of each new row on respective set using the naive bayes al
    ### For each target class we run the naive bayes algorithm and get final probaility of
    ### class. Then we assign the row that class which has higest probabilty.

    ### Do prediction on train sets using naive bayes algorithm
    correct = 0
    total= 0
    for index, row in full_dataset_train.iterrows():

        rs = np.array(row)
        indexs = list(np.where(rs>=1)[0])

        all_sentence_prob_med = sum([each_word_prob_med[i] for i in indexs if i!=X_df.shape
        all_sentence_prob_graph = sum([each_word_prob_graphics[i] for i in indexs if i!=X_d
        if all_sentence_prob_med > all_sentence_prob_graph:
            y_hat = 1
        else:
            y_hat = 0

        if y_hat == row['target']:
            correct+=1
        total+=1
    print("Accuracy on train set: ",correct/total )


    ### Do prediction on validation sets using naive bayes algorithm
    X_df = pd.DataFrame(Xval)
    y_df = pd.DataFrame(Yval)
    full_dataset_validate = X_df.copy()
    full_dataset_validate['target'] = y_df

    correct = 0
    total= 0
    for index, row in full_dataset_validate.iterrows():

        rs = np.array(row)
        indexs = list(np.where(rs>=1)[0])

        all_sentence_prob_med = sum([each_word_prob_med[i] for i in indexs if i!=X_df.shape
        all_sentence_prob_graph = sum([each_word_prob_graphics[i] for i in indexs if i!=X_d
        if all_sentence_prob_med > all_sentence_prob_graph:
            y_hat = 1
        else:
            y_hat = 0

        if y_hat == row['target']:
            correct+=1
        total+=1
    print("Accuracy on validation set: ",correct/total )

    ### Do prediction on test sets using naive bayes algorithm
    X_df = pd.DataFrame(Xtest)
    y_df = pd.DataFrame(Ytest)
    full_dataset_test = X_df.copy()
    full_dataset_test['target'] = y_df

    correct = 0
    total= 0
    for index, row in full_dataset_test.iterrows():

        rs = np.array(row)
        indexs = list(np.where(rs>=1)[0])

        all_sentence_prob_med = sum([each_word_prob_med[i] for i in indexs if i!=X_df.shape
        all_sentence_prob_graph = sum([each_word_prob_graphics[i] for i in indexs if i!=X_d
        if all_sentence_prob_med > all_sentence_prob_graph:
            y_hat = 1
        else:
            y_hat = 0

        if y_hat == row['target']:
            correct+=1
        total+=1
    print("Accuracy on test set: ",correct/total )
```

Exercise - 1) Implementing Naive Bayes Classifier for

Text Data

Using bag of words representation

Convert dataset to bag of words representation and split the dataset into train, validate and test set

In [26]:

```
### Here I have used bag of word representation for dataset

df = twenty_newsgroup_to_df()
corporus, most_freq = make_vocab_with_features(df,10000) ###doing preprocessing on dataset
sentence_vectors = convert_to_bag_of_words(corporus,most_freq) ### calling bag of words fun

X = sentence_vectors
y = np.array(df['target'])
y = y.astype('int')
```

In [27]:

```
Xtrain, Ytrain, Xval, Yval, Xtest, Ytest = split(X, y)
```

Run main function to get accuracies

In [28]:

```
main(Xtrain, Ytrain, Xval, Yval, Xtest, Ytest)
```

```
Accuracy on train set:  0.89
Accuracy on validation set:  0.88
Accuracy on test set:  0.8897338403041825
```

Using tf-idf word representation

Convert dataset to TF-IDF word representation and split the dataset into train, validate and test set

Previously, in case of bag of word, I made 10,000 features for each row. But, here I have just used 1,000 features for each row because it was taking alot of time to run the algorithm when I was using 10,000 features. This is why , I got less accuracy than bag of word represtation. Accuracy can be increased if we use more features for each row!!!

In [29]:

```
### Here I have used TF-IDF word representation for dataset

df__td_idf = twenty_newsgroup_to_df()
df__td_idf['text'].replace('', np.nan, inplace=True)
df__td_idf.dropna(subset=['text'], inplace=True)

### make_vocab_with_features function second argument describe the number of features to ma
corporus_td_idf, most_freq_td_idf = make_vocab_with_features(df__td_idf, 1000)###doing prep
sentence_vectors_td_idf = convert_to_tf_idf(corporus_td_idf,most_freq_td_idf) ### calling T

X = sentence_vectors_td_idf
y = np.array(df__td_idf['target'])
y = y.astype('int')

Xtrain_td_idf, Ytrain_td_idf, Xval_td_idf, Yval_td_idf, Xtest_td_idf, Ytest_td_idf = split(
```

Now we use above splitted data which is TF-IDF representation of words in main function on respective set and get accuracies

In [30]:

```
main(Xtrain_td_idf, Ytrain_td_idf, Xval_td_idf, Yval_td_idf, Xtest_td_idf, Ytest_td_idf )
```

```
Accuracy on train set:  0.4964285714285714
Accuracy on validation set:  0.47333333333333333
Accuracy on test set:  0.547085201793722
```

Exercise - 2) Implementing SVM Classifier via Scikit-Learn

Using bag of words representation

In [31]:

```
### Hyper-paramters space
param_svm = [
    {'C': [1e-4, 1e-3, 1e-2, 1e-1], 'kernel': ['linear']},
    {'C': [120, 200, 500, 1000], 'kernel': ['linear']},
    {'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
    {'C': [120, 200, 500, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]

### Doing grid search with k-fold cross validation to tune hyper paramters
gridsSVC = GridSearchCV(
    SVC(),
    param_grid=param_svm,  # parameters to tune via cross validation
    refit=True,  # fit using all data, on the best detected classifier
    n_jobs=-1,  # number of cores to use for parallelization; -1 for "all cores"
    scoring='accuracy',  # what score are we optimizing?
    cv=StratifiedKFold(n_splits=5),# what type of cross validation to use
    return_train_score=True
)
```

In [32]:

```
### Here I have used bag of word representation for dataset

df = twenty_newsgroup_to_df()
corporus, most_freq = make_vocab_with_features(df,1000) ###doing preprocessing on dataset a
sentence_vectors = convert_to_bag_of_words(corporus,most_freq) ### calling bag of words fun

X = sentence_vectors
y = np.array(df['target'])
y = y.astype('int')
Xtrain, Ytrain, Xval, Yval, Xtest, Ytest = split(X, y)
```

In [33]:

```
gridsSVC.fit(Xtrain, Ytrain)
```

Out[33]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=Fals
e),
             estimator=SVC(), n_jobs=-1,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1],
                          'kernel': ['linear']},
                         {'C': [120, 200, 500, 1000], 'kernel': ['linear']},
                         {'C': [0.0001, 0.001, 0.01, 0.1, 0.5],
                          'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
                         {'C': [120, 200, 500, 1000], 'gamma': [0.001, 0.000
1],
                          'kernel': ['rbf']}],
             return_train_score=True, scoring='accuracy')
```

Optimal Hyper parameters

In [34]:

```
print("Best Parameters are: ", gridsSVC.best_params_)
```

```
Best Parameters are:  {'C': 0.1, 'kernel': 'linear'}
```

GridSearchCV automatically uses the optimal hyperparamters for prediction if we set refit flag to true while optimization. I already set the refit flag to true above so need to first retrain with with optimal hyperparamters while doing prediction on validate and test sets.

Prediction on respective sets

In [35]:

```
predictedY_validation = gridsSVC.predict(Xval)
```

In [36]:

```
print("Best Validation Accuracy is: ", accuracy_score(Yval, predictedY_validation))
```

```
Best Validation Accuracy is:  0.89
```

In [37]:

```
predictedY = gridsSVC.predict(Xtest)
```

In [38]:

```
print("Best Test Accuracy is:", accuracy_score(Ytest, predictedY))
```

```
Best Test Accuracy is: 0.8517110266159695
```

Using TF-IDF word representation

Now we will use TF-IDF representation of dataset and do hyper parameter optimization

In [39]:

```
### Hyper-paramters space
param_svm = [
    {'C': [1e-4, 1e-3, 1e-2, 1e-1], 'kernel': ['linear']},
    {'C': [120, 200, 500, 1000], 'kernel': ['linear']},
    {'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
    {'C': [120, 200, 500, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]

### Doing grid search with k-fold cross validation to tune hyper paramters
gridsSVC_ = GridSearchCV(
    SVC(),
    param_grid=param_svm,  # parameters to tune via cross validation
    refit=True,  # fit using all data, on the best detected classifier
    n_jobs=-1,  # number of cores to use for parallelization; -1 for "all cores"
    scoring='accuracy',  # what score are we optimizing?
    cv=StratifiedKFold(n_splits=5),# what type of cross validation to use
    return_train_score=True
)
```

In [40]:

```
### Here I have used TF-IDF word representation for dataset

df__td_idf = twenty_newsgroup_to_df()
df__td_idf['text'].replace('', np.nan, inplace=True)
df__td_idf.dropna(subset=['text'], inplace=True)

### make_vocab_with_features function second argument describe the number of features to ma
corporus_td_idf, most_freq_td_idf = make_vocab_with_features(df__td_idf, 1000)###doing prep
sentence_vectors_td_idf = convert_to_tf_idf(corporus_td_idf,most_freq_td_idf) ### calling T

X = sentence_vectors_td_idf
y = np.array(df__td_idf['target'])
y = y.astype('int')

Xtrain_td_idf, Ytrain_td_idf, Xval_td_idf, Yval_td_idf, Xtest_td_idf, Ytest_td_idf = split(
```

In [41]:

```
gridsSVC_.fit(Xtrain_td_idf, Ytrain_td_idf)
```

Out[41]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=Fals
e),
             estimator=SVC(), n_jobs=-1,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1],
                          'kernel': ['linear']},
                         {'C': [120, 200, 500, 1000], 'kernel': ['linear']},
                         {'C': [0.0001, 0.001, 0.01, 0.1, 0.5],
                          'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
                         {'C': [120, 200, 500, 1000], 'gamma': [0.001, 0.000
1],
                          'kernel': ['rbf']}],
             return_train_score=True, scoring='accuracy')
```

Optimal Hyper parameters

In [42]:

```
print("Best Parameters are: ", gridsSVC_.best_params_)
```

```
Best Parameters are:  {'C': 120, 'kernel': 'linear'}
```

GridSearchCV automatically uses the optimal hyperparamters for prediction if we set refit flag to true while optimization. I already set the refit flag to true above so need to first retrain with with optimal hyperparamters while doing prediction on validate and test sets.

Prediction on respective sets

In [43]:

```
predictedY_idf_validation = gridsSVC_.predict(Xval_td_idf) ### prediction on validation set
```

In [44]:

```
print("Best Validation Accuracy is: ",accuracy_score(Yval_td_idf, predictedY_idf_validation
```

```
Best Validation Accuracy is:  0.93
```

In [45]:

```
predicted_idf_Y = gridsSVC_.predict(Xtest_td_idf) ### prediction on test set
```

In [46]:

```
print("Best Test Accuracy is: ", accuracy_score(Ytest_td_idf, predicted_idf_Y))
```

```
Best Test Accuracy is:  0.8834080717488789
```