## Feature extraction from 20 newsgroups documents

```
from os import listdir
from os.path import isfile, join
import string
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, \
    ConfusionMatrixDisplay
from timeit import default_timer as timer
from tqdm import tqdm
import time
from google.colab import drive
```

```
drive.mount('/content/drive')
my_path = '/content/drive/MyDrive/20_newsgroups'
#creating a list of folder names to make valid pathnames later
folders = [f for f in listdir(my_path)]
```

```
Mounted at /content/drive
```

```
folders
```

```
['comp.os.ms-windows.misc',
 'comp.graphics',
 'rec.motorcycles',
 'misc.forsale',
 'alt.atheism',
 'comp.windows.x',
 'comp.sys.mac.hardware',
 'rec.autos',
 'rec.sport.baseball',
 'comp.sys.ibm.pc.hardware',
 'talk.politics.guns',
 'sci.electronics',
 'sci.space',
 'sci.med',
 'talk.politics.misc',
 'sci.crypt',
 'soc.religion.christian',
 'talk.religion.misc',
 'rec.sport.hockey',
 'talk.politics.mideast']
```

```
#creating a 2D list to store list of all files in different folders

files = []
for folder_name in folders:
    folder_path = join(my_path, folder_name)
    files.append([f for f in listdir(folder_path)])
```

```
#checking total no. of files gathered

sum(len(files[i]) for i in range(20))
```

```
20007
```

```
#creating a list of pathnames of all the documents
#this would serve to split our dataset into train & test later without any bias

pathname_list = []
for fo in range(len(folders)):
    for fi in files[fo]:
        pathname_list.append(join(my_path, join(folders[fo], fi)))
```

```
len(pathname_list)
```

```
20007
```

```
#making an array containing the classes each of the documents belong to

Y = []
for folder_name in folders:
    folder_path = join(my_path, folder_name)
    num_of_files= len(listdir(folder_path))
    for i in range(num_of_files):
        Y.append(folder_name)
```

```
len(Y)
```

```
20007
```

## splitting the data into train test

```
from sklearn.model_selection import train_test_split
```

```
doc_train, doc_test, Y_train, Y_test = train_test_split(pathname_list, Y, random_state=0, test_size=0.25)
```

# functions for word extraction from documents

```
stopwords = ['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any', 'are', "aren't", 'as', 'at',
 'be', 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by',
 'can', "can't", 'cannot', 'could', "couldn't", 'did', "didn't", 'do', 'does', "doesn't", 'doing', "don't", 'down', 'during',
 'each', 'few', 'for', 'from', 'further',
 'had', "hadn't", 'has', "hasn't", 'have', "haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here', "here's",
 'hers', 'herself', 'him', 'himself', 'his', 'how', "how's",
 'i', "i'd", "i'll", "i'm", "i've", 'if', 'in', 'into', 'is', "isn't", 'it', "it's", 'its', 'itself',
 "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself',
 'no', 'nor', 'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours' 'ourselves', 'out', 'over', 'own',
 'same', "shan't", 'she', "she'd", "she'll", "she's", 'should', "shouldn't", 'so', 'some', 'such',
 'than', 'that',"that's", 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', "there's", 'these', 'they', "they'd",
 "they'll", "they're", "they've", 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 'very',
 'was', "wasn't", 'we', "we'd", "we'll", "we're", "we've", 'were', "weren't", 'what', "what's", 'when', "when's", 'where',
 "where's", 'which', 'while', 'who', "who's", 'whom', 'why', "why's",'will', 'with', "won't", 'would', "wouldn't",
 'you', "you'd", "you'll", "you're", "you've", 'your', 'yours', 'yourself', 'yourselves',
 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten', 'hundred', 'thousand', '1st', '2nd', '3rd',
 '4th', '5th', '6th', '7th', '8th', '9th', '10th']
```

```
#function to preprocess the words list to remove punctuations

def preprocess(words):
    #we'll make use of python's translate function,that maps one set of characters to another
    #we create an empty mapping table, the third argument allows us to list all of the characters
    #to remove during the translation process

    #first we will try to filter out some  unnecessary data like tabs
    table = str.maketrans('', '', '\t')
    words = [word.translate(table) for word in words]

    punctuations = (string.punctuation).replace("'", "")
    # the character: ' appears in a lot of stopwords and changes meaning of words if removed
    #hence it is removed from the list of symbols that are to be discarded from the documents
    trans_table = str.maketrans('', '', punctuations)
    stripped_words = [word.translate(trans_table) for word in words]

    #some white spaces may be added to the list of words, due to the translate function & nature of our documents
    #we remove them below
    words = [str for str in stripped_words if str]

    #some words are quoted in the documents & as we have not removed ' to maintain the integrity of some stopwords
    #we try to unquote such words below
    p_words = []
    for word in words:
        if (word[0] and word[len(word)-1] == "'"):
            word = word[1:len(word)-1]
        elif(word[0] == "'"):
            word = word[1:len(word)]
        else:
            word = word
        p_words.append(word)

    words = p_words.copy()
```

```
    #we will also remove just-numeric strings as they do not have any significant meaning in text classification
    words = [word for word in words if not word.isdigit()]

    #we will also remove single character strings
    words = [word for word in words if not len(word) == 1]

    #after removal of so many characters it may happen that some strings have become blank, we remove those
    words = [str for str in words if str]

    #we also normalize the cases of our words
    words = [word.lower() for word in words]

    #we try to remove words with only 2 characters
    words = [word for word in words if len(word) > 2]

    return words


#function to remove stopwords

def remove_stopwords(words):
    words = [word for word in words if not word in stopwords]
    return words


#function to convert a sentence into list of words

def tokenize_sentence(line):
    words = line[0:len(line)-1].strip().split(" ")
    words = preprocess(words)
    words = remove_stopwords(words)

    return words


#function to remove metadata

def remove_metadata(lines):
    for i in range(len(lines)):
        if(lines[i] == '\n'):
            start = i+1
            break
    new_lines = lines[start:]
    return new_lines


!pip install chardet



import chardet



def detect_encoding(path):

    with open(path, 'rb') as f:

        raw_data = f.read()

    return chardet.detect(raw_data)['encoding']



def tokenize(path):

    # Detect the best guess for encoding

    encoding = detect_encoding(path)

    print(f"Detected encoding: {encoding}")

    # Open the file with the detected encoding

    with open(path, 'r', encoding=encoding, errors='replace') as f:

        text_lines = f.readlines()



    #removing the meta-data at the top of each document

    text_lines = remove_metadata(text_lines)
```

```
    #initiazing an array to hold all the words in a document

    doc_words = []



    #traverse over all the lines and tokenize each one with the help of helper function: tokenize_sentence

    for line in text_lines:

        doc_words.append(tokenize_sentence(line))



    return doc_words
```

```
Requirement already satisfied: chardet in /usr/local/lib/python3.11/dist-packages (5.2.0)
```

```
#a simple helper function to convert a 2D array to 1D, without using numpy

def flatten(list):
    new_list = []
    for i in list:
        for j in i:
            new_list.append(j)
    return new_list
```

## using the above functions on actual documents

```
len(folders)
```

```
20
```

```
list_of_words = []

for document in doc_train:
        list_of_words.append(flatten(tokenize(document)))
```

```
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
```

```
len(list_of_words)
```

```
15005
```

```
len(flatten(list_of_words))
```

```
1934068
```

from above lengths we observe that the code has been designed in as such a way that the 2D list: list_of_words contains the vocabulary of each document file in the each of its rows, and collectively contains all the words we extract from the 20_newsgroups folder

```
import numpy as np
np_list_of_words = np.asarray(flatten(list_of_words))
```

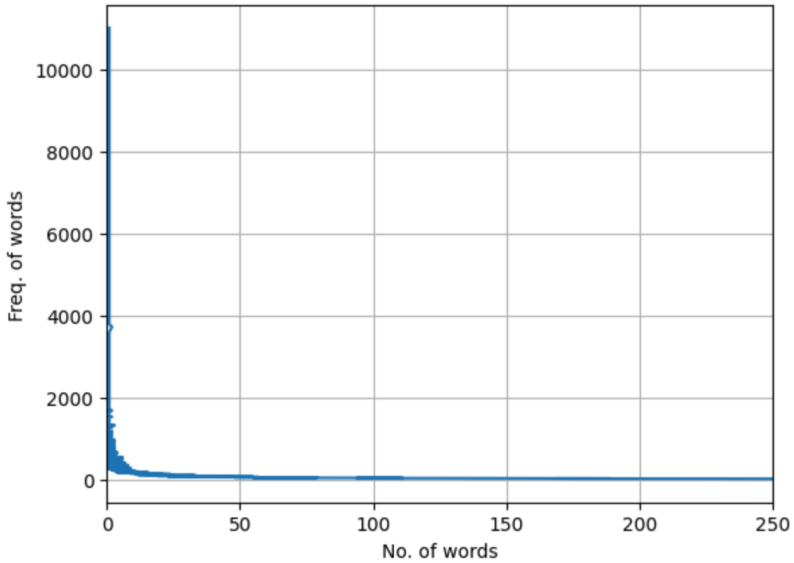
```
#finding the number of unique words that we have extracted from the documents

words, counts = np.unique(np_list_of_words, return_counts=True)
len(words)
```

```
147768
```

```
#sorting the unique words according to their frequency

freq, wrds = (list(i) for i in zip(*(sorted(zip(counts, words), reverse=True))))
```

```
f_o_w = []
n_o_w = []
for f in sorted(np.unique(freq), reverse=True):
    f_o_w.append(f)
    n_o_w.append(freq.count(f))
```

```
import matplotlib.pyplot as plt
y = f_o_w
x = n_o_w
plt.xlim(0,250)
plt.xlabel("No. of words")
plt.ylabel("Freq. of words")
plt.plot(x, y)
plt.grid()
plt.show()
```




## we'll start making our train data here onwards

```
#deciding the no. of words to use as feature

n = 5000
features = wrds[0:n]
print(features)
```

```
['writes', 'article', 'people', 'like', 'just', 'know', 'get', 'think', 'also', 'use', 'time', 'good', 'new', 'may', 'even', 'now'
```

```
#creating a dictionary that contains each document's vocabulary and ocurence of each word of the vocabulary

dictionary = {}
doc_num = 1
for doc_words in list_of_words:
    #print(doc_words)
    np_doc_words = np.asarray(doc_words)
    w, c = np.unique(np_doc_words, return_counts=True)
    dictionary[doc_num] = {}
    for i in range(len(w)):
        dictionary[doc_num][w[i]] = c[i]
    doc_num = doc_num + 1
```

```
dictionary.keys()
```

```
8221, 8222, 8223, 8224, 8225, 8226, 8227, 8228, 8229, 8230, 8231, 8232, 8233, 8234, 8235, 8236, 8237, 8238, 8239, 8240, 8241,
8242, 8243, 8244, 8245, 8246, 8247, 8248, 8249, 8250, 8251, 8252, 8253, 8254, 8255, 8256, 8257, 8258, 8259, 8260, 8261, 8262,
8263, 8264, 8265, 8266, 8267, 8268, 8269, 8270, 8271, 8272, 8273, 8274, 8275, 8276, 8277, 8278, 8279, 8280, 8281, 8282, 8283,
8284, 8285, 8286, 8287, 8288, 8289, 8290, 8291, 8292, 8293, 8294, 8295, 8296, 8297, 8298, 8299, 8300, 8301, 8302, 8303, 8304,
8305, 8306, 8307, 8308, 8309, 8310, 8311, 8312, 8313, 8314, 8315, 8316, 8317, 8318, 8319, 8320, 8321, 8322, 8323, 8324, 8325,
8326, 8327, 8328, 8329, 8330, 8331, 8332, 8333, 8334, 8335, 8336, 8337, 8338, 8339, 8340, 8341, 8342, 8343, 8344, 8345, 8346,
8347, 8348, 8349, 8350, 8351, 8352, 8353, 8354, 8355, 8356, 8357, 8358, 8359, 8360, 8361, 8362, 8363, 8364, 8365, 8366, 8367,
8368, 8369, 8370, 8371, 8372, 8373, 8374, 8375, 8376, 8377, 8378, 8379, 8380, 8381, 8382, 8383, 8384, 8385, 8386, 8387, 8388,
8389, 8390, 8391, 8392, 8393, 8394, 8395, 8396, 8397, 8398, 8399, 8400, 8401, 8402, 8403, 8404, 8405, 8406, 8407, 8408, 8409,
8410, 8411, 8412, 8413, 8414, 8415, 8416, 8417, 8418, 8419, 8420, 8421, 8422, 8423, 8424, 8425, 8426, 8427, 8428, 8429, 8430,
8431, 8432, 8433, 8434, 8435, 8436, 8437, 8438, 8439, 8440, 8441, 8442, 8443, 8444, 8445, 8446, 8447, 8448, 8449, 8450, 8451,
8452, 8453, 8454, 8455, 8456, 8457, 8458, 8459, 8460, 8461, 8462, 8463, 8464, 8465, 8466, 8467, 8468, 8469, 8470, 8471, 8472,
8473, 8474, 8475, 8476, 8477, 8478, 8479, 8480, 8481, 8482, 8483, 8484, 8485, 8486, 8487, 8488, 8489, 8490, 8491, 8492, 8493,
8494, 8495, 8496, 8497, 8498, 8499, 8500, 8501, 8502, 8503, 8504, 8505, 8506, 8507, 8508, 8509, 8510, 8511, 8512, 8513, 8514,
8515, 8516, 8517, 8518, 8519, 8520, 8521, 8522, 8523, 8524, 8525, 8526, 8527, 8528, 8529, 8530, 8531, 8532, 8533, 8534, 8535,
8536, 8537, 8538, 8539, 8540, 8541, 8542, 8543, 8544, 8545, 8546, 8547, 8548, 8549, 8550, 8551, 8552, 8553, 8554, 8555, 8556,
8557, 8558, 8559, 8560, 8561, 8562, 8563, 8564, 8565, 8566, 8567, 8568, 8569, 8570, 8571, 8572, 8573, 8574, 8575, 8576, 8577,
8578, 8579, 8580, 8581, 8582, 8583, 8584, 8585, 8586, 8587, 8588, 8589, 8590, 8591, 8592, 8593, 8594, 8595, 8596, 8597, 8598,
8599, 8600, 8601, 8602, 8603, 8604, 8605, 8606, 8607, 8608, 8609, 8610, 8611, 8612, 8613, 8614, 8615, 8616, 8617, 8618, 8619,
8620, 8621, 8622, 8623, 8624, 8625, 8626, 8627, 8628, 8629, 8630, 8631, 8632, 8633, 8634, 8635, 8636, 8637, 8638, 8639, 8640,
8641, 8642, 8643, 8644, 8645, 8646, 8647, 8648, 8649, 8650, 8651, 8652, 8653, 8654, 8655, 8656, 8657, 8658, 8659, 8660, 8661,
8662, 8663, 8664, 8665, 8666, 8667, 8668, 8669, 8670, 8671, 8672, 8673, 8674, 8675, 8676, 8677, 8678, 8679, 8680, 8681, 8682,
8683, 8684, 8685, 8686, 8687, 8688, 8689, 8690, 8691, 8692, 8693, 8694, 8695, 8696, 8697, 8698, 8699, 8700, 8701, 8702, 8703,
8704, 8705, 8706, 8707, 8708, 8709, 8710, 8711, 8712, 8713, 8714, 8715, 8716, 8717, 8718, 8719, 8720, 8721, 8722, 8723, 8724,
8725, 8726, 8727, 8728, 8729, 8730, 8731, 8732, 8733, 8734, 8735, 8736, 8737, 8738, 8739, 8740, 8741, 8742, 8743, 8744, 8745,
8746, 8747, 8748, 8749, 8750, 8751, 8752, 8753, 8754, 8755, 8756, 8757, 8758, 8759, 8760, 8761, 8762, 8763, 8764, 8765, 8766,
8767, 8768, 8769, 8770, 8771, 8772, 8773, 8774, 8775, 8776, 8777, 8778, 8779, 8780, 8781, 8782, 8783, 8784, 8785, 8786, 8787,
```

```python
#now we make a 2D array having the frequency of each word of our feature set in each individual documents

X_train = []
for k in dictionary.keys():
    row = []
    for f in features:
        if(f in dictionary[k].keys()):
            #if word f is present in the dictionary of the document as a key, its value is copied
            #this gives us no. of occurences
            row.append(dictionary[k][f])
        else:
            #if not present, the no. of occurences is zero
            row.append(0)
    X_train.append(row)
```

```python
#we convert the X and Y into np array for concatenation and conversion into dataframe

X_train = np.asarray(X_train)
Y_train = np.asarray(Y_train)
```

```python
len(X_train)
```

```
15005
```

```python
len(Y_train)
```

```
15005
```

## we'll make our test data by performing the same operations as we did for train data

```python
list_of_words_test = []

for document in doc_test:
        list_of_words_test.append(flatten(tokenize(document)))
```

```
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
Detected encoding: ascii
```

```
dictionary_test = {}
doc_num = 1
for doc_words in list_of_words_test:
    #print(doc_words)
    np_doc_words = np.asarray(doc_words)
    w, c = np.unique(np_doc_words, return_counts=True)
    dictionary_test[doc_num] = {}
    for i in range(len(w)):
        dictionary_test[doc_num][w[i]] = c[i]
    doc_num = doc_num + 1
```

```
#now we make a 2D array having the frequency of each word of our feature set in each individual documents

X_test = []
for k in dictionary_test.keys():
    row = []
    for f in features:
        if(f in dictionary_test[k].keys()):
            #if word f is present in the dictionary of the document as a key, its value is copied
            #this gives us no. of occurences
            row.append(dictionary_test[k][f])
        else:
            #if not present, the no. of occurences is zero
            row.append(0)
    X_test.append(row)
```

```
X_test = np.asarray(X_test)
Y_test = np.asarray(Y_test)
```

```
len(X_test)
```

```
5002
```

```
len(Y_test)
```

```
5002
```

## Text Classification

## performing Text Classification using sklearn's Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X_train, Y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
Y_predict = clf.predict(X_test)
```

## testing scores

```
clf.score(X_test, Y_test)
```

```
0.7752898840463814
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(classification_report(Y_test, Y_predict))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.57      0.73      0.64       236
           comp.graphics       0.67      0.69      0.68       253
 comp.os.ms-windows.misc       0.73      0.70      0.71       233
comp.sys.ibm.pc.hardware       0.66      0.70      0.68       249
   comp.sys.mac.hardware       0.71      0.75      0.73       249
          comp.windows.x       0.80      0.78      0.79       246
            misc.forsale       0.84      0.79      0.81       240
               rec.autos       0.81      0.87      0.83       268
         rec.motorcycles       0.82      0.90      0.85       249
      rec.sport.baseball       0.91      0.94      0.92       255
        rec.sport.hockey       0.97      0.94      0.95       257
               sci.crypt       0.92      0.86      0.89       248
         sci.electronics       0.70      0.69      0.69       231
                 sci.med       0.89      0.85      0.87       233
               sci.space       0.90      0.86      0.88       284
  soc.religion.christian       0.74      0.83      0.78       248
      talk.politics.guns       0.69      0.81      0.74       240
   talk.politics.mideast       0.93      0.87      0.90       243
      talk.politics.misc       0.66      0.66      0.66       243
      talk.religion.misc       0.62      0.35      0.44       297

                accuracy                           0.78      5002
               macro avg       0.78      0.78      0.77      5002
            weighted avg       0.78      0.78      0.77      5002
```

## training scores

```
Y_predict_tr = clf.predict(X_train)
```

```
clf.score(X_train, Y_train)
```

```
0.8313895368210596
```

```
print(classification_report(Y_train, Y_predict_tr))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.73      0.86      0.79       764
           comp.graphics       0.73      0.74      0.74       747
 comp.os.ms-windows.misc       0.80      0.77      0.79       767
comp.sys.ibm.pc.hardware       0.76      0.79      0.78       751
   comp.sys.mac.hardware       0.80      0.86      0.83       751
          comp.windows.x       0.85      0.82      0.83       754
            misc.forsale       0.82      0.85      0.83       760
               rec.autos       0.87      0.89      0.88       732
         rec.motorcycles       0.89      0.94      0.91       751
      rec.sport.baseball       0.91      0.94      0.93       745
        rec.sport.hockey       0.96      0.95      0.95       753
               sci.crypt       0.93      0.89      0.91       752
         sci.electronics       0.80      0.81      0.80       769
                 sci.med       0.94      0.88      0.91       767
               sci.space       0.93      0.87      0.90       716
  soc.religion.christian       0.86      0.89      0.88       749
      talk.politics.guns       0.73      0.88      0.80       760
```

```
       talk.politics.mideast       0.90      0.85      0.88       757
          talk.politics.misc       0.73      0.68      0.70       757
          talk.religion.misc       0.71      0.45      0.55       703

                    accuracy                           0.83     15005
                   macro avg       0.83      0.83      0.83     15005
                weighted avg       0.83      0.83      0.83     15005
```

## performing Text Classification using my implementation of Multinomial Naive Bayes

### functions for my implementation

```
#function to create a training dictionary out of the text files for training set, consisiting the frequency of
#words in our feature set (vocabulary) in each class or label of the 20 newsgroup

def fit(X_train, Y_train):
    result = {}
    classes, counts = np.unique(Y_train, return_counts=True)

    for i in range(len(classes)):
        curr_class = classes[i]

        result["TOTAL_DATA"] = len(Y_train)
        result[curr_class] = {}

        X_tr_curr = X_train[Y_train == curr_class]

        num_features = n

        for j in range(num_features):
            result[curr_class][features[j]] = X_tr_curr[:,j].sum()

        result[curr_class]["TOTAL_COUNT"] = counts[i]

    return result


#function for calculating naive bayesian log probablity for each test document being in a particular class

def log_probablity(dictionary_train, x, curr_class):
    output = np.log(dictionary_train[curr_class]["TOTAL_COUNT"]) - np.log(dictionary_train["TOTAL_DATA"])
    num_words = len(x)
    for j in range(num_words):
        if(x[j] in dictionary_train[curr_class].keys()):
            xj = x[j]
            count_curr_class_equal_xj = dictionary_train[curr_class][xj] + 1
            count_curr_class = dictionary_train[curr_class]["TOTAL_COUNT"] + len(dictionary_train[curr_class].keys())
            curr_xj_prob = np.log(count_curr_class_equal_xj) - np.log(count_curr_class)
            output = output + curr_xj_prob
        else:
            continue

    return output


#helper function for the predict() function that predicts the class or label for one test document at a time

def predictSinglePoint(dictionary_train, x):
    classes = dictionary_train.keys()
    best_p = -10000
    best_class = -1
    for curr_class in classes:
        if(curr_class == "TOTAL_DATA"):
            continue
        p_curr_class = log_probablity(dictionary_train, x, curr_class)
        if(p_curr_class > best_p):
            best_p = p_curr_class
            best_class = curr_class

    return best_class


#predict function that predicts the class or label of test documents using train dictionary made using the fit() function

def predict(dictionary_train, X_test):
    Y_pred = []
    for x in X_test:
        y_predicted = predictSinglePoint(dictionary_train, x)
        Y_pred.append(y_predicted)
```

```
    #print(Y_pred)
    return Y_pred
```

## performing the implementation

```
train_dictionary = fit(X_train, Y_train)
```

```
X_test = []

for key in dictionary_test.keys():
    X_test.append(list(dictionary_test[key].keys()))
```

```
my_predictions = predict(train_dictionary, X_test)
```

```
my_predictions = np.asarray(my_predictions)
```

```
accuracy_score(Y_test, my_predictions)
```

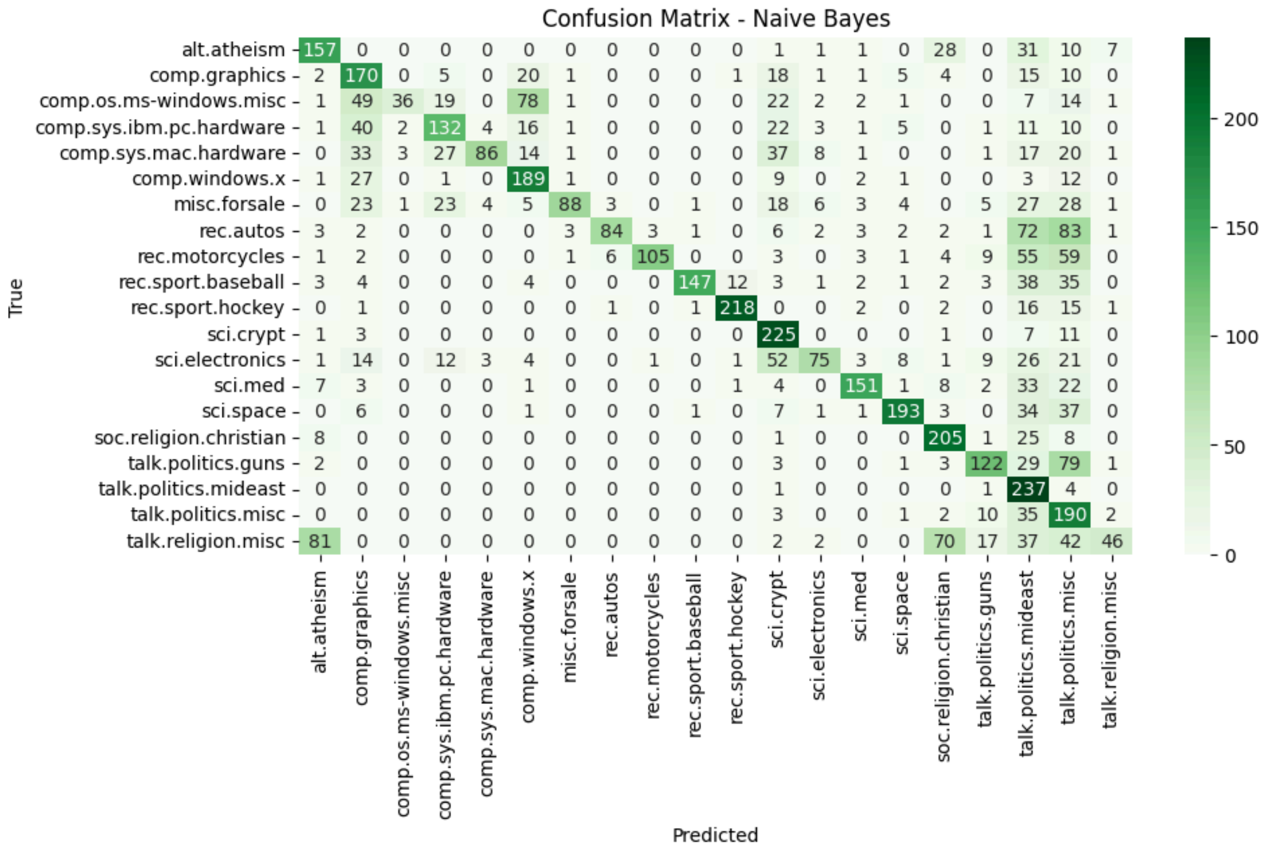
```
0.5709716113554578
```

```
print("\nClassification Report for Naive Bayes:\n")
print(classification_report(Y_test, my_predictions))
```

```
Classification Report for Naive Bayes:

                          precision    recall  f1-score   support

             alt.atheism       0.58      0.67      0.62       236
           comp.graphics       0.45      0.67      0.54       253
 comp.os.ms-windows.misc       0.86      0.15      0.26       233
comp.sys.ibm.pc.hardware       0.60      0.53      0.56       249
   comp.sys.mac.hardware       0.89      0.35      0.50       249
          comp.windows.x       0.57      0.77      0.65       246
            misc.forsale       0.91      0.37      0.52       240
               rec.autos       0.89      0.31      0.46       268
         rec.motorcycles       0.96      0.42      0.59       249
      rec.sport.baseball       0.97      0.58      0.72       255
        rec.sport.hockey       0.94      0.85      0.89       257
               sci.crypt       0.51      0.91      0.66       248
         sci.electronics       0.74      0.32      0.45       231
                 sci.med       0.86      0.65      0.74       233
               sci.space       0.86      0.68      0.76       284
  soc.religion.christian       0.61      0.83      0.70       248
      talk.politics.guns       0.67      0.51      0.58       240
   talk.politics.mideast       0.31      0.98      0.47       243
      talk.politics.misc       0.27      0.78      0.40       243
      talk.religion.misc       0.75      0.15      0.26       297

                accuracy                           0.57      5002
               macro avg       0.71      0.57      0.57      5002
            weighted avg       0.71      0.57      0.57      5002
```

```
# Import libraries
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import nltk
from nltk.corpus import stopwords
import re
import matplotlib.pyplot as plt
import seaborn as sns

# Fetch the 20 newsgroups dataset
newsgroups = fetch_20newsgroups()

# ... (rest of your code)

conf_matrix_nb = confusion_matrix(Y_test, my_predictions)
plt.figure(figsize=(10, 5))
# Use newsgroups.target_names instead of folders.target_names
sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap='Greens', xticklabels=newsgroups.target_names, yticklabels=newsgroups.target_names
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
plt.title('Confusion Matrix - Naive Bayes')
plt.show()
accuracy = accuracy_score(Y_test, Y_predict)
precision = precision_score(Y_test, Y_predict, average='weighted')
recall = recall_score(Y_test, Y_predict, average='weighted')
f1 = f1_score(Y_test, Y_predict, average='weighted')

print("\nClassification Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```




```
Classification Metrics:
Accuracy: 0.7753
Precision: 0.7768
Recall: 0.7753
```