from os import listdir

from os.path import isfile, join

import string

my\_path = '20\_newsgroups'

#creating a list of folder names to make valid pathnames later

folders = [f for f in listdir(my\_path)]

folders

#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)])

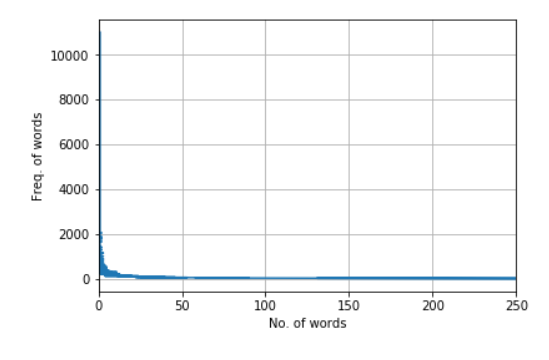


Fig: Plot of number of words and frequency words

#checking total no. of files gathered

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

#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)

#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)

#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)

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

#function to convert a document into list of words

def tokenize(path):

#load document as a list of lines

f = open(path, 'r', encoding = "latin-1")

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

#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

len(folders)

list\_of\_words = []

for document in doc\_train:

list\_of\_words.append(flatten(tokenize(document)))

len(list\_of\_words)

len(flatten(list\_of\_words))

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)

#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()

#deciding the no. of words to use as feature

n = 5000

features = wrds[0:n]

#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()

#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)

#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)

len(X\_train)

len(Y\_train)

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

list\_of\_words\_test = []

for document in doc\_test:

list\_of\_words\_test.append(flatten(tokenize(document)))

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)

len(X\_test)

Y\_test = np.asarray(Y\_test)

len(Y\_test)

#performing Text Classification using sklearn's Multinomial Naive Bayes

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB()

clf.fit(X\_train, Y\_train)

Y\_predict = clf.predict(X\_test)

#testing scores

clf.score(X\_test, Y\_test)

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

print(classification\_report(Y\_test, Y\_predict))

#training scores

Y\_predict\_tr = clf.predict(X\_train)

clf.score(X\_train, Y\_train)

print(classification\_report(Y\_train, Y\_predict\_tr))

#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)

print(classification\_report(Y\_test, my\_predictions))