

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | **B.Tech** | Semester | : | **Win Sem 21-22** |
| Course | : | **Web Mining Lab** | Code | : | **CSE3024** |
| Faculty | : | **Dr.Bhuvaneswari A** | Slot | : | **L7+L8** |
| Date | : | **22-02-2022** | Marks | : | **10 Points** |

# Name: RAHUL GARG Register No: 19BCE1431

**Exercise 6: Web Documents Classification - Naïve Bayes Classifier**

Write a Naïve Bayes Classifier in python without using any direct ML package for the following datasets (1) and (2).

1. Dataset of Restaurant customer reviews.



Identify the label -> Positive or Negative of the following query by applying NB classifer with Laplace smoothing

test\_data = Serving good Food and absolutely perfect Restaurant

# CODE:

from enum import unique import pandas as pd import numpy as np import re

all\_words = [] word\_size\_class = {} all\_sents = [] word\_frequency\_label = {}

def load\_data(filename):

data = pd.read\_excel(filename) test\_index = data['Class'] == '?' training\_data = data[-test\_index] testing\_data = data[test\_index]

testing\_data.reset\_index(inplace = True, drop = True)

return (training\_data,testing\_data)

def find\_vocab\_size(training\_data):

for index,sent in enumerate(training\_data['URL']): ext\_words = re.findall(r"([a-z0-9]+)",sent) label = training\_data['Class'][index]

word\_size\_class[label] = word\_size\_class.get(label,0) + len(ext

\_words)

all\_words.extend(ext\_words) all\_sents.append(ext\_words)

unique\_words\_count = len(set(all\_words)) all\_words\_count = len(all\_words)

global unique\_words

unique\_words = list(set(all\_words))

return (unique\_words\_count,all\_words\_count)

def find\_prior\_probabilties(training\_data):

class\_prior = {}

labels = training\_data['Class'].unique()

total = len(training\_data) for l in labels:

class\_prior[l] = sum(training\_data['Class'] == l) / total

return class\_prior

def find\_word\_frequency\_class(training\_data):

for word in unique\_words:

for index,sent\_vec in enumerate(all\_sents): if word in sent\_vec:

if word not in word\_frequency\_label: word\_frequency\_label[word] = {}

label = training\_data['Class'][index]

word\_frequency\_label[word][label] = word\_frequency\_labe l[word].get(label,0) + sent\_vec.count(word)

def display\_conditional\_prob(vocab\_size,labels):

i=0

for word in word\_frequency\_label:

for label in labels:

num = word\_frequency\_label[word].get(label,0) + 1 denom = word\_size\_class[label] + vocab\_size space = " "

print(f"P({word}/{label}) = {num}/{denom} {space\*(14- len(word))}",end="\t")

print()

def display\_test\_results(data,labels,vocab\_size,class\_prior):

for i,sent in enumerate(data['URL']): ext\_words = re.findall(r"([a-z0-9]+)",sent)

probs = []

for label in labels: prob = 1

for word in ext\_words:

class\_dict = word\_frequency\_label.get(word)

num = 0

denom = (word\_size\_class[label]+vocab\_size)

if class\_dict == None: num = 1

else:

num = class\_dict.get(label,0) + 1

prob \*= (num/denom)

prior = class\_prior[label] probs.append(prior\*prob)

probs = np.array(probs, dtype=np.float32)

index = np.argmax(probs)

print(f"\n{sent} ===> {labels[index]} {probs}") data['Class'][i] = labels[index]

print("\nFinal Result is as follows :") print(f"{data}\n")

def main():

filename = "./naive\_bayes\_data.xlsx"

train\_data,test\_data = load\_data(filename)

print("Training Data:") print(train\_data)

labels = train\_data['Class'].unique()

class\_prior = find\_prior\_probabilties(train\_data) print(f"\nPrior Probabilities: {class\_prior}\n")

vocab\_size, total\_word\_count = find\_vocab\_size(train\_data) print(f"Vocabulary size: {vocab\_size}")

print(f"Total words in train data: {total\_word\_count}\n")

find\_word\_frequency\_class(train\_data)

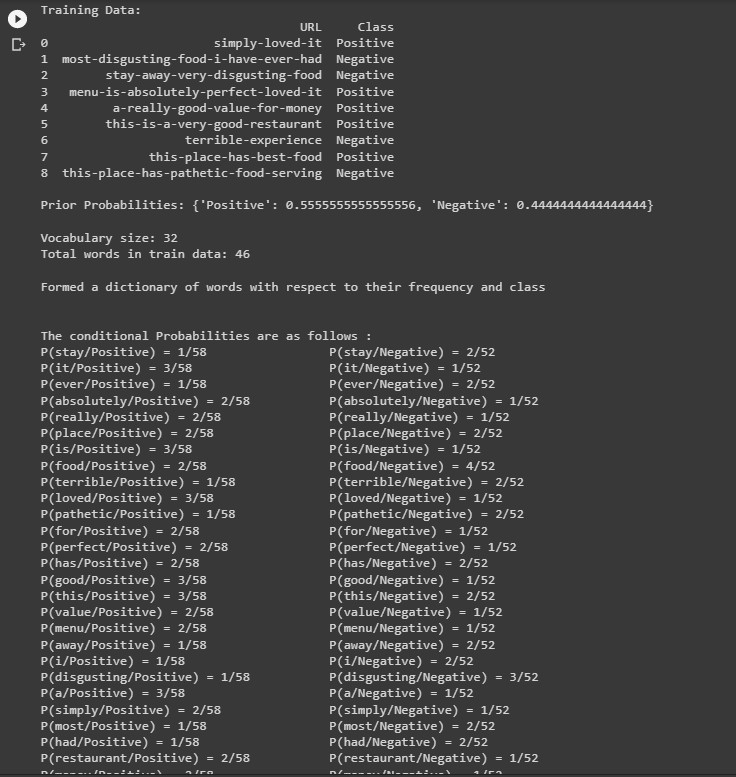
print("Formed a dictionary of words with respect to their frequency and class\n")

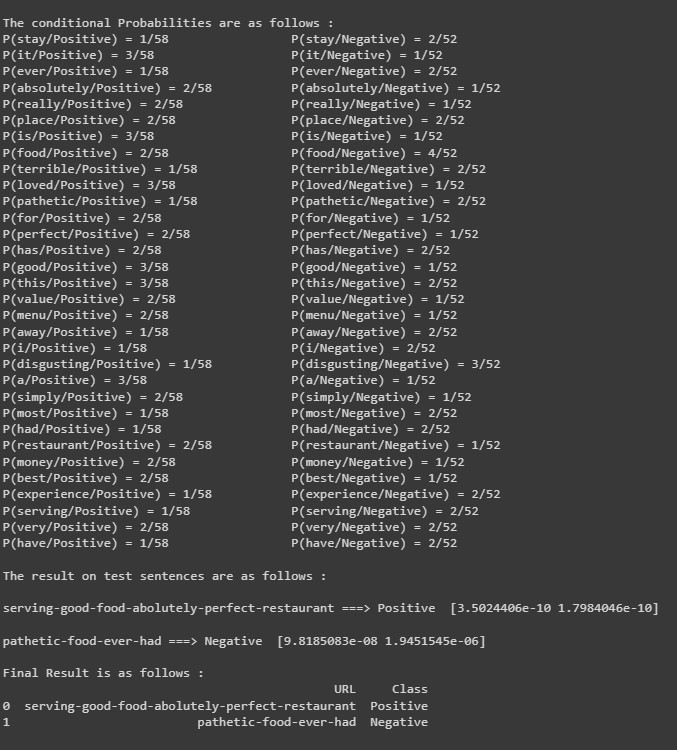
print('\nThe conditional Probabilities are as follows :') display\_conditional\_prob(vocab\_size,labels)

print("\nThe result on test sentences are as follows :") display\_test\_results(test\_data,labels,vocab\_size,class\_prior)

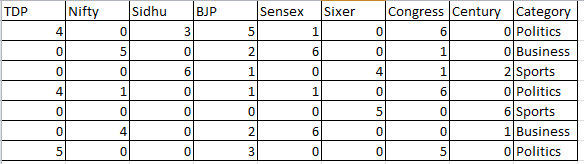
if \_name == " main ": main()

# OUTPUT:





1. The dataset is provided in the [Data.csv](./Data.csv) file



Identify the class/category  Politics or Business or Sports - of the following query by applying NB classifer with Laplace smoothing

(i) query\_data = [4,0,2,0,1,0,6,0]

(ii) query\_data = [0,0,2,0,0 ,9,0,9]

(iii) query\_data = [5,0,2,5,0 ,9,0,9]

# CODE:

import csv

def strings(row,count): str1=""

result\_arr = ['Politics','Business','Sports','Politics','Sports','Business','Politics','?', '?','?']

arr = []

fields = ['TDP','Nifty','Sidhu','BJP','Sensex','Sixer','Congress','Century'] for i in range(0,8):

temp = row[i]; #print(temp) temp1 = int(temp)

for j in range(0,temp1):

str1 = str1 + " " + fields[i] arr.append(str1) arr.append(result\_arr[count]) return arr

filename = "./naive\_bayes\_data - Copy.csv" fields = []

class\_arr = ['Politics','Business','Sports','Politics','Sports','Business','Politics','?', '?','?']

rows = [] mini\_data = []

with open(filename, 'r') as csvfile:

# creating a csv reader object

csvreader = csv.reader(csvfile) fields = next(csvreader)

for k,row in enumerate(csvreader): rows.append(row)

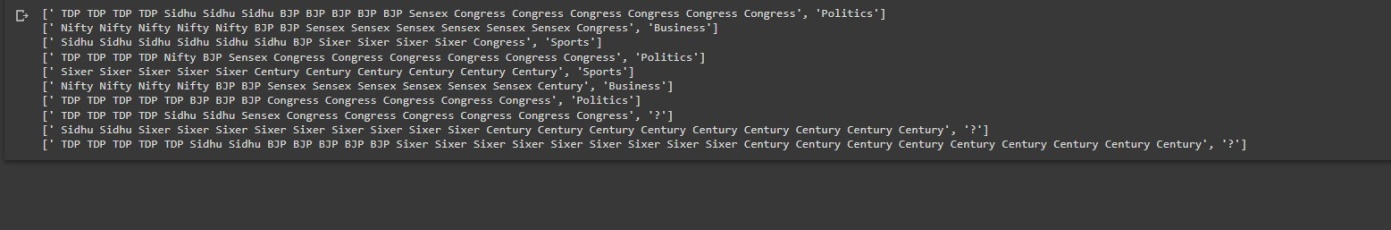
#print(row)

tempo = strings(row,k) mini\_data.append(tempo)

for i in mini\_data:

print(i)

**OUTPUT:**



Procedure to be followed:

The following dataset gives the Term Frequencies of a few documents for the given keywords, and the last column gives the category of the document.

## Steps

We first enter the given data into a pandas data frame and save it as a CSV file for easier access.

Now we load the data stored earlier from the CSV file, then we get the list of all unique output labels, all the words provided to us in the dataset which are available in the document, and also get the total number of documents provided to us.

In the Naive Bayes process we use the Bayes rule which goes as follows :

`P(A/B) = P(B/A)P(A) / P(B)`

In our current case we have three output classes, so we would need to find :

`P(Politics / query), P(Sports / query), P(Business / query)`

And based on these three values, we choose the class with the maximum value and choose that as the appropriate category for the provided query.

Based on the above two formulae, we can infer that we would need to find the Probability of the occurrence of each output class.

`P(Output Class) = Number of documents that belong to the output class / Total number of document`

Now to the next part of the numerator `P(Query / Output Class)` , this can be calculated as the product of the Probabilities of the each word of the query belonging to that particular output class.

`P(Query / Output Class) = ∏ P(Word / Output Class)`

Now to find the probability of a word in the class :

P(Word / Output Class) = Number of times the word belongs to the output class / Total number of words that belong to the output class

We can now use all these formulas to calculate the numerator, in our present scenario we are more interested to know more about the class that the document belongs to, rather than the actual probability score for the document belonging to each class. So we will be ignoring the denominator for comparison reasons.

We can notice here that the other probabilities are zero, though in our current case we have the correct answer, but in some cases we might get a zero probability in all the classes if we encounter an unseen word, so we can apply many techniques and algorithms to solve this, but for now we will be applying Smoothing, and mainly Laplace smoothing in our current case.

This is the formula for `Laplace smoothing`, and we apply this when we find the

`P(Word / Output Class)`. This solves the problem of zeros in the conditional probability as we add a ALPHA value to the numerator, and to prevent the Probabilities from going greater than 1, we also ass a normalising factor to the denominator. When we assume the value of `ALPHA` as 1, we consider this as the Laplace Smoothing.

**GOOGLE COLAB LINK:**

[**https://colab.research.google.com/drive/13kAreyYCHMQVK1wPDpoOWKyCC7EVVb**](https://colab.research.google.com/drive/13kAreyYCHMQVK1wPDpoOWKyCC7EVVbGK?usp=sharing)[**GK?usp=sharing**](https://colab.research.google.com/drive/13kAreyYCHMQVK1wPDpoOWKyCC7EVVbGK?usp=sharing)