VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Programme	:	B.Tech	**Semester**	:	Win Sem 21-22
Course	:	Web Mining	**Code**	:	CSE3024
Faculty	:	Dr. Bhuvaneswari A.	**Slot**	:	L7+L8
Date	:	25-02-2022	**Marks**	:	10 Points

Lab Submission-6

Tanmay Mahajan

19BCE1735

Q1:

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.

Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

Labelled Training Dataset

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

test_data 1= Serving good Food absolutely perfect Restaurant
Test_data 2= pathetic food ever had

CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('data1.txt', sep='\t',
                 names=['sentiment', 'document'])

df
length = len(df)  # length of the datframe
pos_count = len(df[df['sentiment'] == 1])  # positive_sentiment count
neg_count = len(df[df['sentiment'] == 0])  # negative_sentiment count
print (
    'length=',
    length,
    '\npos_count',
    pos_count,
    '\nneg_count',
    neg_count,
    )
sns.countplot(y="sentiment",data=df)#plotting the data
plt.show()
```

```
def sentence_to_words(sentence):

    l = sentence.lower()  # convert sentence to lowercase
    l = l.split()  # split sentence into individual word
    p = ""
    word_list = []

    for word in l:

        p = ""

        for letter in word:

            if ord(letter) >= 67 and ord(letter) <= 122:
                p = p + letter
        word_list.append(p)

    return word_list  # return the word list of the sentence devoid of special characters
and numericals


def naive_bayes_train(X, Y, a=0.000001):
    n_length = len(X)
    n_class_pos = len(Y[Y == 1])
    n_class_neg = len(Y[Y == 0])
    prior_pos = n_class_pos / n_length  # prior probability for  class
    prior_neg = n_class_neg / n_length  #prior probability for class
    (n, p, bag) = bag_of_words_maker(X, Y)

    pr = {}

    for i in range(len(bag)):   #evaluating the likelihood prob for each word given a
class
        p_pos = (bag['count_pos'][i] + a) / (p + len(bag) * a)

        p_neg = (bag['count_neg'][i] + a) / (n + len(bag) * a)

        pr[bag['index'][i]] = [p_pos, p_neg]
    pr = pd.DataFrame(pr).T
    pr.columns = ['sent=positive', 'sent=negative']
    pr = pr.reset_index()

    return (prior_pos, prior_neg, pr)


def naive_bayes_predict(
    X,
    pr,
    prior_pos,
    prior_neg,
    ):
```

```
    Y = []

    for i in range(len(X)):
        k_pos = 1
        k_neg = 1
        p = sentence_to_words(X[i])

        for k in range(len(pr)):

            for word in p:

                if word == pr['index'][k]:
                    k_pos = k_pos * pr['sent=positive'][k] #pdt of likelihood prob given
the word is present in vocabulary
                    k_neg = k_neg * pr['sent=negative'][k]

        nb = [prior_neg * k_neg, prior_pos * k_pos] # multiply each likelihood prob with
the prior prob
        Y.append(np.argmax(nb))

    return Y


def bag_of_words_maker(X, Y):

    bag_dict_binary_NB_pos = {} #keeping track of the positive class words
    bag_dict_binary_NB_neg = {} #keeping track of the negative class words
    stop_words = [
        'the',
        'is',
        'a',
        'was',
        'it',
        'food',
        'This',
        'and',
        '',
        'i',
        'I',
        'am',
        'of',
        'that',
        ]
    for i in range(len(X)):
        p = sentence_to_words(X[i])
        sent = Y[i]
        x_pos = {}
        x_neg = {} #we intialize the dict every iteration so that it does not consider
repititions .(Binary NB)

        if sent == 1:
            for word in p:

                if word in x_pos.keys():
```

```
                    x_pos[word] = [x_pos[word][0] + 1, x_pos[word][1]]  #word is the key
and value stored is [count, sentiment]
                else:
                    x_pos[word] = [1, sent]

            for key in x_pos.keys():

                if key in bag_dict_binary_NB_pos.keys():
                    bag_dict_binary_NB_pos[key] = \
                        [bag_dict_binary_NB_pos[key][0] + 1,
                         bag_dict_binary_NB_pos[key][1]]
                else:

                    bag_dict_binary_NB_pos[key] = [1, sent]  #storing it in the final dict

        if sent == 0:

            for word in p:
                if word in x_neg.keys():
                    x_neg[word] = [x_neg[word][0] + 1, x_neg[word][1]]
                else:
                    x_neg[word] = [1, sent]
            for key in x_neg.keys():
                if key in bag_dict_binary_NB_neg.keys():
                    bag_dict_binary_NB_neg[key] = \
                        [bag_dict_binary_NB_neg[key][0] + 1,
                         bag_dict_binary_NB_neg[key][1]]
                else:

                    bag_dict_binary_NB_neg[key] = [1, sent]

    # print(bag_dict_multi.keys())
    # returns the dataframe containg word count in each sentiment
    neg_bag = pd.DataFrame(bag_dict_binary_NB_neg).T
    pos_bag = pd.DataFrame(bag_dict_binary_NB_pos).T

    neg_bag.columns = ["count_neg", "sentiment_neg"]
    pos_bag.columns = ["count_pos", "sentiment_pos"]
    neg_bag = neg_bag.drop(stop_words)
    pos_bag = pos_bag.drop(stop_words)
    neg_bag = neg_bag.reset_index()
    pos_bag = pos_bag.reset_index()
    n = len(neg_bag)
    p = len(pos_bag)
    bag_of_words = pd.merge(neg_bag, pos_bag, on=["index"], how="outer")
    bag_of_words["count_neg"] = bag_of_words["count_neg"].fillna(0)
    bag_of_words["count_pos"] = bag_of_words["count_pos"].fillna(0)
    bag_of_words["sentiment_neg"] = bag_of_words["sentiment_neg"
            ].fillna(0)
    bag_of_words["sentiment_pos"] = bag_of_words["sentiment_pos"
            ].fillna(1)

    return (n, p, bag_of_words)
x = df["document"]
```

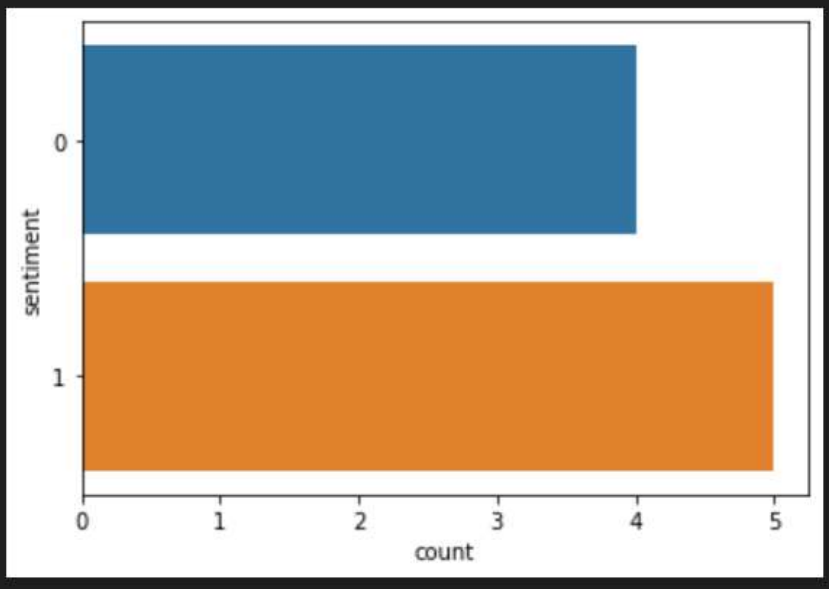
```
y = df["sentiment"]
(n, p, bag_of_words) = bag_of_words_maker(x, y)
print (n, " ", p)
bag_of_words.head(5)
prior_pos,prior_neg,table = naive_bayes_train(x,y)
table.head(5)
from sklearn.model_selection import train_test_split

X = df["document"]
Y = df["sentiment"]
x_train, x_test, y_train, y_test = train_test_split(X,Y,)

x_train = x_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
x_test = x_test.reset_index(drop=True)
a,b,bag = naive_bayes_train(x_train,y_train)
y_predicted = naive_bayes_predict(x_test,bag,a,b)
y_predicted
from sklearn.metrics import accuracy_score, confusion_matrix
print ("Accuracy=", accuracy_score(y_test, np.array(y_predicted)))

(tn, fp, fn, tp) = confusion_matrix(y_test,
                                    np.array(y_predicted)).ravel()

print ("precsion=", tp / (tp + fp))
print ("recall=", tp / (tp + fn))
```

OUTPUT:

	sentiment	document
0	1	simply Loved it
1	0	most disgusting food I have ever had
2	0	stay away, very disgusting food
3	1	menu is absolutely perfect, loved it!
4	1	a really good value for money
5	1	this is a very good restaurant
6	0	terrible experience!
7	1	this place has best food
8	0	this place has most pathetic serving food!

```
length= 9
pos_count 5
neg_count 4
```

```
sns.countplot(y="sentiment",data=df)#plotting the data
plt.show()
```




```
Accuracy= 0.9720570749108205
precsion= 0.9807073954983923
recall= 0.9692796610169492
```

Q2:

2. The dataset . Create your own dataset as [Data.csv](./Data.csv) file

TDP	Nifty	Sidhu	BJP	Sensex	Sixer	Congress	Century	Category
4	0	3	5	1	0	6	0	Politics
0	5	0	2	6	0	1	0	Business
0	0	6	1	0	4	1	2	Sports
4	1	0	1	1	0	6	0	Politics
0	0	0	0	0	5	0	6	Sports
0	4	0	2	6	0	0	1	Business
5	0	0	3	0	0	5	0	Politics

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 numpy as np
import pandas as pd
from csv import reader
```

✓ 1.2s

```
data = pd.read_csv(r'lab6.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   TDP       7 non-null      int64
 1   Nifty     7 non-null      int64
 2   Sidhu     7 non-null      int64
 3   BJP       7 non-null      int64
 4   Sensex    7 non-null      int64
 5   Sixer     7 non-null      int64
 6   Congress  7 non-null      int64
 7   Century   7 non-null      int64
```

```
Index(['TDP', 'Nifty', 'Sidhu', 'BJP', 'Sensex', 'Sixer', 'Congress',
       'Century', 'Category'],
      dtype='object')
```

0	4	0	3	5	1	0	6	0	Politics
2	0	0	6	1	0	4	1	2	Sports
4	0	0	0	0	0	5	0	6	Sports
6	5	0	0	3	0	0	5	0	Politics

```
'Po11t1c s' 'Bus1 nes s' 'Sport s']
['TDP', 'Nifty', 'Sidhu', 'BJP', 'Sensex', 'Sixer', 'Congress', 'Century']
7
```

```
conditional_probability = {}
probability = {}
```

```
for outputClass in outputlabels:
    temp_dataframe = data.loc[data['Category']==outputClass]
    probability[outputClass]=(temp_dataframe.shape[0]/numtraindocuments)
```

```
print(probability)
```

```
{'Politics': 0.42857142857142855, 'Business': 0.2857142857142857, 'Sports': 0.2857142857142857}
```

```
ALPHA = 1
```

```
for outputClass in outputlabels:
    temp_dataframe = data.loc[data['Category']==outputClass]
    total_word_count_in_category =0
    for i in range(temp_dataframe.shape[0]):
        for word in words:
            total_word_count_in_category += temp_dataframe.iloc[i][word]
    for word in words:
        current_word_count_in_category =0
        for i in range(temp_dataframe.shape[0]):
            current_word_count_in_category += temp_dataframe.iloc[i][word]
            cur_prob = (current_word_count_in_category + ALPHA) / (total_word_count_in_category)
            conditional_probability[(word, outputClass)] = cur_prob
```

```
print("Conditional probability after applying smoothing\n")
conditional_probability
```

```
(('TDP',  'Po1it ics'): 0.3111111111111111,
 ('Nifty'   'Po1it:ic s'): 0.044444444444444446
 ('Sidhu',  'Po1it:1c s'): 0. 08B88888888B8B889,
 ('B3P',  'Po1it i cs'): 0.2222222222222222,
 ('Sensex', 'Politics'): 0.06666666666666667,
 ('Sixer',  'Po1 it:1c s'): 0. 022222222222222223
 ('Congress',  'Po11t ie s ): 0.4,
 ('Century  ,  'Po11ti cs'): 0. 022222222222222223,
 ('TDP',  'Bus iness'): 0. 037037037037037035
 ('Nifty',  'Bu siness'): 0. 37037057637037035,
 ('sidhu',  'Bu siness'): 0. 037037037037037035
 ('B3P',  'Bus iness'): 0.1851B51851851B517,
 ('Sensex', 'Business'): 0.4B14814814B148145,
 ('Sixer',  'Bu siness'): 0. 037037037037037035
 ('Congress',  'Bus1nes s ): 0. 07407407407407407,
 ('Century',  'Buslnes s'): 0. 07407467407407467,
('TDP',  'Sports'): 0. 04,
 ('Nifty', 'Sports'): 0.04,
 ('sidhu',  'Sport:s'): 0.28,
 ('BJP', 'Sports'): 0.08,
 ('Sensex',  'Sport s'): 6. 04,
 ('Sixer',  'Sport:s'): 6. 4,
 ('Congress',  'Sport s  ): 6. 68,
 ('Century  ,  'Sports'): 0. 36}
```

FINAL OUTPUT:

```
    i=1
    for categorical_result_probability in result_probability:
        result_category = max(categorical_result_probability, key=categorical_result_probability.get)
        result_score = categorical_result_probability[result_category]
        print(f"The query {i} entered belongs to the category : {result_category}")
        i=i+1
```

```
The query 1 entered belongs to the category : Politics
The query 2 entered belongs to the category : Sports
The query 3 entered belongs to the category : Sports
```