Name : Maitri Nilesh Patel

UTA ID: 1001716017

Assignment 03

Importing Dataset for Movie Review

Here I have imported yelp Dataset from Kaggle Sentiment labelled Dataset

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
    Drive already mounted at /content/gdrive; to attempt to forcibly remount, call d
```

```
import pandas as pd
Data = pd.read_csv('/content/gdrive/My Drive/Dataset/yelp_labelled.txt', delimiter='\t
Data.head()
```

	Reviews	Sentiments
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

```
Data.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 1000 entries, 0 to 999
    Data columns (total 2 columns):
     #   Column      Non-Null Count  Dtype
    ---  ------      --------------  -----
     0   Reviews     1000 non-null   object
     1   Sentiments  1000 non-null   int64
    dtypes: int64(1), object(1)
    memory usage: 15.8+ KB
```

```
Data.isnull().sum()
```

```
    Reviews       0
    Sentiments    0
    dtype: int64
```

```
Data = pd.DataFrame(Data)
```

```
import os
import numpy as np
from pandas import Series
import re
from sys import path
from collections import Counter
```

```
Data.Sentiments.value_counts()
```

```
1    500
0    500
Name: Sentiments, dtype: int64
```

A) Divide the Dataset as Training and Development Data.

#Here I have divided data in the ratio of 60:40 as Train and Development.

For the Testing data I have imported them at the last as it not required for the time being, it will be required for the last sub-question

```
shuffle = Data.sample(frac=1).to_numpy()
Datasize = int(.60 * len(Data))
Training_Dataset = shuffle[:Datasize]
Dev_Dataset = shuffle[Datasize:]
print(Training_Dataset)
print(Dev_Dataset)
```

```
[['I checked out this place a couple years ago and was not impressed.' 0]
 ['Overall, I like there food and the service.' 1]
 ["Don't waste your time here." 0]
 ...
 ['We enjoy their pizza and brunch.' 1]
 ['We waited for thirty minutes to be seated (although there were 8 vacant table
  0]
 ['I ordered Albondigas soup - which was just warm - and tasted like tomato soup
  0]]
[['Pretty awesome place.' 1]
 ['Then our food came out, disappointment ensued.' 0]
 ['And it was way to expensive.' 0]
 ["I've never been more insulted or felt disrespected." 0]
 ['The best place in Vegas for breakfast (just check out a Sat, or Sun.'
  1]
 ['My side Greek salad with the Greek dressing was so tasty, and the pita and hu
  1]
 ['It was absolutely amazing.' 1]
 ['And the chef was generous with his time (even came around twice so we can tak
  1]
 ["Now the burgers aren't as good, the pizza which used to be amazing is doughy 
  0]
 ['I took back my money and got outta there.' 0]
 ['Needless to say, we will never be back here again.' 0]
 ['My husband and I ate lunch here and were very disappointed with the food and 
  0]
 ['Thoroughly disappointed!' 0]
 ['This place is not worth your time, let alone Vegas.' 0]
 ["We thought you'd have to venture further away to get good sushi, but this pla
  1]
 ['This place should honestly be blown up.' 0]
 ["We'll never go again." 0]
 ['It was attached to a gas station, and that is rarely a good sign.' 0]
 ["Don't bother coming here." 0]
 ['The staff was very attentive.' 1]
 ["At least 40min passed in between us ordering and the food arriving, and it wa
  0]
 ['Hands down my favorite Italian restaurant!' 1]
 ['This place is overpriced, not consistent with their boba, and it really is OV
  0]
 ['Which are small and not worth the price.' 0]
 ['My sashimi was poor quality being soggy and tasteless.' 0]
 ['Things that went wrong: - They burned the saganaki.' 0]
 ['This place is amazing!' 1]
 ['Very Very Disappointed ordered the $35 Big Bay Plater.' 0]
 ['My husband said she was very rude... did not even apologize for the bad food 
  0]
 ['Check it out.' 1]
 ['The atmosphere is modern and hip, while maintaining a touch of coziness.'
  1]
 ['By this time our side of the restaurant was almost empty so there was no excu
  0]
 ['Although I very much liked the look and sound of this place, the actual exper
  0]
 ['It was packed!!' 0]
 ['Total letdown, I would much rather just go to the Camelback Flower Shop and C
  0]
 ['Ordered an appetizer and took 40 minutes and then the pizza another 10 minute
  0]
```

```
print("Shape of Training_Dataset",{Training_Dataset.shape})
print("Shape of Dev_Dataset",{Dev_Dataset.shape})
```

```
    Shape of Training_Dataset {(600, 2)}
    Shape of Dev_Dataset {(400, 2)}
```

```
x_Train = Training_Dataset[:,0]
y_Train = Training_Dataset[:,-1]

x_Dev = Dev_Dataset[:,0]
x_Dev = Dev_Dataset[:,0]
y_Dev = Dev_Dataset[:,-1]

x_Train
y_Train
```

```
array([0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 0], dtype=object)
```

B) Building a Vocabulary list as below

```
V = {}
V_List = []
def build_V_List(Data):
  V = counting_number_of_unique_words(Data)
  for i in V.keys():
    V_List.append(i)
    return V_List


Count = {}
def counting_number_of_unique_words(Data):
  for L in Data:
    for W in L.split():
      if W in Count:
        Count[W] = Count[W] + 1
      else:
          Count[W] = 1
  return Count
```

```
Count = counting_number_of_unique_words(x_Train)
print("Unique words in Dataset before Removing less Occuring Words : ", len(Count))
```

```
Unique words in Dataset before Removing less Occuring Words :  2061
```

```
print (Count)
```

```
{'I': 175, 'checked': 2, 'out': 13, 'this': 57, 'place': 43, 'a': 135, 'couple':
```

```
Repeat_W = []
for i in Count.keys():
  if (Count[i]<7):
    Repeat_W.append(i)
for i in Repeat_W:
  del Count[i]
len (Count)
```

```
150
```

```
def Remove_less_occuring_word(Data, Count):
  for i, Reviews in enumerate(Data):
    Cleaning = []
    for W in Reviews.split():
      if W in Count.keys():
         Cleaning.append(W)
    Cleaning = ' '.join(Cleaning)
    Data[i] = Cleaning
  return Data
```

```
x_Train = Remove_less_occuring_word(x_Train, Count)
print("Total unique words after Removing less occuring Words : ", len(x_Train))
print(Count)
print('')
```

```
Total unique words after Removing less occuring Words :  600
{'I': 175, 'out': 13, 'this': 57, 'place': 43, 'a': 135, 'and': 234, 'was': 157,
```

```
Reviews_0_Train = Training_Dataset[y_Train == 0]
Reviews_1_Train = Training_Dataset[y_Train == 1]
```

```
print("Negative Reviews in Training Dataset Count : ", len((Reviews_0_Train)))
```

```
    Negative Reviews in Training Dataset Count :  301
```

```
print("Positive Reviews in Training Dataset Count : ", len((Reviews_1_Train)))
```

```
    Positive Reviews in Training Dataset Count :  299
```

Making Directory

```
def Total_Words(Data, Count):
  New_Directory = {}
  for W in Count.keys():
    Count1 = 0
    for Reviews in Data:
      if W in Reviews:
        Count1 = Count1 + 1
    New_Directory [W] = Count1
  return New_Directory
```

```
Occur_Word = Total_Words(x_Train, Count)
print("The count in dataset", Occur_Word['the'])
```

```
    The count in dataset 218
```

C) Finding Basic Probabilities

```
def basic_probability(y_Train):
  Probability = {}
  for a in np.unique(y_Train):
    Count1 = sum(y_Train == a)
    Probability[a] = Count1 / y_Train.size
  return Probability
```

```
Probability = basic_probability(y_Train)
print("Negative probabiity",Probability[0])
print("Positive probability",Probability[1])
```

```
    Negative probabiity 0.5016666666666667
    Positive probability 0.49833333333333335
```

# Probability of Occurance of word "the"

```
def Probability_of_the_in_Dataset (Data,W):
```

```
def Probability_of_the_in_Dataset (Data,W):
  Count1 = 0
  if W not in Occur_Word.keys():
    Count1 = 0
  else:
    Count1 = Occur_Word[W]
  return Count1 / len(Data)
```

```
The_Probability = Probability_of_the_in_Dataset(x_Train,'the')
```

```
print("Probability of the in dataset P[the]:",The_Probability * 100)
```

```
Probability of the in dataset P[the]: 36.333333333333336
```

Conditional Probability based on the Sentiments

```
def Positive_Words (Data, label, Count):
  P_Words = {}
  for W in Count.keys():
    P = 0
    for index, Reviews in enumerate(Data):
      if W in Reviews.split() and label[index] == 1:
        P = P + 1
    P_Words[W] = P
  return P_Words
```

```
def Negative_Words (Data, label, Count):
 N_Words = {}
 for W in Count.keys():
    N = 0
    for index, Reviews in enumerate(Data):
      if W in Reviews.split() and label[index] == 1:
        N = N + 1
    N_Words[W] = N
 return N_Words
```

```
P_Words = Positive_Words(x_Train, y_Train, Count)
N_Words = Negative_Words(x_Train, y_Train, Count)
```

```
P_Words['the']
```

```
94
```

```
N_Words['the']
```

```
94
```

```
def docu_positive (Data):
  P_doc = 0
  for i in range(len(Data)):
    if (Data[i] == 1):
      P_doc = P_doc + 1
  return P_doc



def docu_negative (Data):
  N_doc = 0
  for i in range(len(Data)):
    if (Data[i] == 0):
      N_doc = N_doc + 1
  return N_doc


def docu_count_positive_negative (Data, attribute):
  if attribute == "positive":
    C = docu_positive(Data)
  else:
    C = docu_negative(Data)
  return C



P_doc = docu_count_positive_negative (y_Train, attribute= "positive")
N_doc = docu_count_positive_negative (y_Train, attribute= "negative")



print(P_doc)
print(N_doc)
```

```
299
301
```

```
def Number_Words_in_Doc(W, Count, attribute):
  Count1 = 0
  if attribute == "positive":
    if W in P_Words.keys():
      Count1 = P_Words[W]
    else:
      Count1 = 0

  elif attribute == "negative":
    if W in N_Words.keys():
      Count1 = N_Words[W]
    else:
      Count1 = 0
  return Count1



n = Number_Words_in_Doc('the', Count, attribute= "positive")
m = n / P_doc
print("p[the|positive]: " +str(m*100)+"%")
```

```
p[the|positive]: 31.438127090301005%
```

D) Calulating Accuracy using the Development Dataset

```
#probability for p[positive|the] without NAIVE BAYES
probability1 = (m*(P_doc/len(x_Train)))/ The_Probability
print("Probability of given word in data",probability1*100)
```

```
Probability of given word in data 43.11926605504588
```

```
def Naive_Bayes (Data, label,W, Count, attribute):
  n = Number_Words_in_Doc(W, Count, attribute)
  D = Probability_of_the_in_Dataset(Data, W)
  PN = docu_count_positive_negaive(label,attribute)
  if D == 0:
    D = 0.1
  else:
    D = D
  Result = (n/PN)*(PN/len(Data))/D
  return Result


def Naive_Bayes_Data (Data, label, Count, data1):
  Prediction_P = 1
  Prediction_N = 1

  pred = []
  for index, Reviews in enumerate(data1):
    for W in Reviews.split(" "):
      NB = Naive_Bayes(Data, label, W, Count, attribute= "negative")
      if NB == 0:
        NB = 0.1
      else:
        NB = NB

      Prediction_N = Prediction_N * NB


      NB1 = Naive_Bayes(Data, label, W, Count, attribute= "positive")
      if NB1 == 0:
        NB1 = 0.1
      else:
        NB1 = NB1

      Prediction_P = Prediction_P * NB1

    if Prediction_P > Prediction_N:
      pred.append(1)
    else:
      pred.append(0)

  return pred


pred = Naive_Bayes_Data(x_Train,y_Train,Count,x_Dev)


def Accuracy(label, Prediction):
  pred1 = 0
  for i in range(len(label)):
    if(label[i]== Prediction[i]):
      pred1 = pred1 + 1
    Z = float(pred1/len(label))
  return Z



Accuracy1 = Accuracy(y_Dev, pred)
print("Accuracy Without Smoothing Data: "+str(Accuracy1*100)+"%")
```

```
Accuracy Without Smoothing Data: 45.5%
```

E) Comparing the effect of Smoothing

DATA Smoothing

```
def Naive_Bayes_Smooth(Data, label, W,Count, attribute):
    n = Number_Words_in_Doc(W, Count, attribute)
    D = Probability_of_the_in_Dataset(Data, W)
    PN = docu_count_positive_negaive(label,attribute)
    if D == 0:
        D = 0.1
    else:
        D = D
    Result = (((n/PN)*(PN/len(Data)))+1)/(D+2)
    return Result
def Naive_Bayes_for_Smoothing_data(Data,label,Count,data1):
    Prediction_P = 1
    Prediction_N = 1
    pred = []
    for index,Reviews in enumerate(data1):
        for W in Reviews.split(" "):
            NB = Naive_Bayes_Smooth(Data, label, W,Count, attribute = "negative")
            if NB == 0:
                NB = 0.1
            else:
                NB = NB
            Prediction_N = Prediction_N * NB
            #print("n",negative_pred)
            NB1 = Naive_Bayes_Smooth(Data, label, W,Count, attribute = "positive")
            if NB1 == 0:
                NB1 = 0.1
            else:
                NB1 = NB1
            Prediction_P = Prediction_P * NB1
            #print("p",positive_pred)
        if Prediction_P > Prediction_N:
            pred.append(1)
        else:
            pred.append(0)
    return pred
```

```
pred = Naive_Bayes_for_Smoothing_data(x_Train,y_Train, Count, x_Dev)
```

```
Accuracy1 = Accuracy(y_Dev, pred)
print("Accuracy Without Smoothing Data: "+str(Accuracy1*100)+"%")
```

```
Accuracy Without Smoothing Data: 51.24999999999999%
```

Derive Top ten words that predicts Positive and Negative class

```
import operator, itertools
Positive_top = {}
for W in Count.keys():
  Z = Naive_Bayes(x_Train, y_Train, W, Count, attribute= "positive")
  Positive_top[W] = Z

top_reversed = dict(sorted(Positive_top.items(), key= operator.itemgetter(1),reverse=
Top_ten_Positive = dict(itertools.islice(top_reversed.items(), 10))
```

```
print("Top Ten Words that Predict the Positive :",Top_ten_Positive.keys())
```

```
    Top Ten Words that Predict the Positive : dict_keys(['great', 'Great', 'love', '
```

```
Negative_top = {}
for W in Count.keys():
  Z = Naive_Bayes(x_Train, y_Train, W, Count, attribute= "negative")
  Negative_top[W] = Z

top_reversed = dict(sorted(Negative_top.items(), key= operator.itemgetter(1),reverse=T
Top_ten_Negative = dict(itertools.islice(top_reversed.items(), 10))


print("Top Ten Words that Predict the Negative :",Top_ten_Negative.keys())
```

```
    Top Ten Words that Predict the Negative : dict_keys(['I', 'out', 'this', 'place'
```

F) Using the Test Dataset and calulating the Final Accuracy

```
S = Data.sample(frac=1).to_numpy()
```

Shuffling the data to get the Random Values in Training set and Development set

```
size = int(0.40*len(Data))
Test = S[:size]
print("Size of the Test set is : ",len(Test))
print("Shape of the Test set is : ",{Test.shape})
```

```
    Size of the Test set is :  400
    Shape of the Test set is :  {(400, 2)}
```

```
x_Test = Test[:,0]
y_Test = Test[:,-1]
```

Accuracies Without and With Smoothing

Accuracy Without Smoothing

```
pred = Naive_Bayes_Data(x_Train, y_Train,Count, x_Test)
```

```
Accuracy1 = Accuracy(y_Test,pred)
print("Accuracy of the Data without Smoothing :"+str(Accuracy1*100)+"%")
```

```
    Accuracy of the Data without Smoothing :46.75%
```

Accuracy after Smoothing

```
pred = Naive_Bayes_for_Smoothing_data(x_Train, y_Train,Count, x_Test)
```

```
Accuracy1 = Accuracy(y_Test,pred)
print("Accuracy of the Data with Smoothing :"+str(Accuracy1*100)+"%")
```

```
    Accuracy of the Data with Smoothing :48.5%
```

✓ 0s completed at 9:15 PM ● ✕