CSE3024 WEB MINING

NAME: Aryaman Mishra

REGISTRATION NUMBER: 19BCE1027

LAB EXERCISE 5

Problem-1: Write a Naïve Bayes Classifier in python without using any package for the following dataset.

The following gives a Term frequency of some of the documents for the given keywords and the last column gives the category of the document.

Document	TDP	Nifty	Sidhu	BJP	Sensex	Sixer	Congress	Century	Category
D1	4	0	3	5	1	0	6	0	Politics
D2	0	5	0	2	6	0	1	0	Business
D3	0	0	6	1	0	4	1	2	Sports
D4	4	1	0	1	1	0	6	0	Politics
D5	0	0	0	0	0	5	0	6	Sports
D6	0	4	0	2	6	0	0	1	Business
D7	5	0	0	3	0	0	5	0	Politics

Predict which Category the following document will fall into

Testdoc	0	3	0	2	6	0	2	1	?

Proposed Algorithm/Pseudocode:

1. Compute the term document matrix for each category.
2. Find frequency of word in each category.
3. Calculate the probability of each word in every category.
4. Calculate the total count of each feature in the training set.
5. Find the probability of all the words in a text document.
6. Lowest priority is given to the document.

Data Structure Proposed: 2D-Arrays, Dictionaries.

Libraries Used: Pandas, CountVectorizer

IMPLEMENTATION CODE AND RESULTS:

In [3]:
```
import pandas as pd

columns = ['sent', 'class']
rows = []

rows = [['TDP TDP TDP TDP Sidhu Sidhu Sidhu BJP BJP BJP BJP BJP Congress Congress Congress Congress Congress Congress ', 'Politi
        ['Nifty Nifty Nifty Nifty Nifty BJP BJP Sensex Sensex Sensex Sensex Sensex Sensex Congress ', 'Business'],
        ['Sidhu Sidhu Sidhu Sidhu Sidhu Sidhu BJP Sixer Sixer Sixer Sixer Congress Century Century ', 'Sports'],
        ['TDP TDP TDP TDP Nifty BJP Sensex Congress Congress Congress Congress Congress Congress ', 'Politics'],
        ['Sixer Sixer Sixer Sixer Sixer Century Century Century Century Century Century ', 'Sports'],
        ['Nifty Nifty Nifty Nifty Nifty BJP BJP Sensex Sensex Sensex Sensex Sensex Sensex Century ', 'Business'],
        ['TDP TDP TDP TDP TDP BJP BJP BJP Congress Congress Congress Congress Congress ', 'Politics']]
```

In [4]:
```
training_data=pd.DataFrame(rows, columns=columns)
training_data
```

Out[4]:

	sent	class
0	TDP TDP TDP TDP Sidhu Sidhu Sidhu BJP BJP BJP ...	Politics
1	Nifty Nifty Nifty Nifty Nifty BJP BJP Sensex S...	Business
2	Sidhu Sidhu Sidhu Sidhu Sidhu Sidhu BJP Sixer ...	Sports
3	TDP TDP TDP TDP Nifty BJP Sensex Congress Cong...	Politics
4	Sixer Sixer Sixer Sixer Sixer Century Century ...	Sports
5	Nifty Nifty Nifty Nifty BJP BJP Sensex Sensex ...	Business
6	TDP TDP TDP TDP TDP BJP BJP BJP Congress Congr...	Politics

In [13]:
```
from sklearn.feature_extraction.text import CountVectorizer

pol_docs = [row['sent'] for index,row in training_data.iterrows() if row['class'] == 'Politics']
bus_docs = [row['sent'] for index,row in training_data.iterrows() if row['class'] == 'Business']
sports_docs = [row['sent'] for index,row in training_data.iterrows() if row['class'] == 'Sports']
```

In [14]:
```
vec_p = CountVectorizer()
X_p = vec_p.fit_transform(pol_docs)
tdm_p = pd.DataFrame(X_p.toarray(), columns=vec_p.get_feature_names())

tdm_p
```

Out[14]:

	bjp	congress	nifty	sensex	sidhu	tdp
0	5	6	0	0	3	4
1	1	6	1	1	0	4
2	3	5	0	0	0	5

In [15]:
```
vec_b = CountVectorizer()
X_b = vec_b.fit_transform(bus_docs)
tdm_b = pd.DataFrame(X_b.toarray(), columns=vec_b.get_feature_names())

tdm_b
```

Out[15]:

	bjp	century	congress	nifty	sensex
0	2	0	1	5	6
1	2	1	0	4	6

In [16]:
```
vec_s = CountVectorizer()
X_s = vec_s.fit_transform(sports_docs)
tdm_s = pd.DataFrame(X_s.toarray(), columns=vec_s.get_feature_names())

tdm_s
```

Out[16]:

	bjp	century	congress	sidhu	sixer
0	1	2	1	6	4
1	0	6	0	0	5

In [17]:
```
word_list_p = vec_p.get_feature_names();
count_list_p = X_p.toarray().sum(axis=0)
freq_p = dict(zip(word_list_p,count_list_p))
freq_p
```

Out[17]: {'bjp': 9, 'congress': 17, 'nifty': 1, 'sensex': 1, 'sidhu': 3, 'tdp': 13}

In [19]:
```
word_list_b = vec_b.get_feature_names();
count_list_b = X_b.toarray().sum(axis=0)
freq_b = dict(zip(word_list_b,count_list_b))
freq_b
```

Out[19]: {'bjp': 4, 'century': 1, 'congress': 1, 'nifty': 9, 'sensex': 12}

In [20]:
```
word_list_s = vec_s.get_feature_names();
count_list_s = X_s.toarray().sum(axis=0)
freq_s = dict(zip(word_list_s,count_list_s))
freq_s
```

Out[20]: {'bjp': 1, 'century': 8, 'congress': 1, 'sidhu': 6, 'sixer': 9}

In [21]:
```
from sklearn.feature_extraction.text import CountVectorizer

docs = [row['sent'] for index,row in training_data.iterrows()]

vec = CountVectorizer()
X = vec.fit_transform(docs)

total_features = len(vec.get_feature_names())
total_features
```

Out[21]: 8

In [24]:
```
total_cnts_features_p = count_list_p.sum(axis=0)
total_cnts_features_b = count_list_b.sum(axis=0)
total_cnts_features_s = count_list_s.sum(axis=0)
```

In [28]:
```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

new_sentence = 'Nifty Nifty Nifty BJP BJP Sensex Sensex Sensex Sensex Sensex Sensex Congress Congress Century'
new_word_list = word_tokenize(new_sentence)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

In [34]:
```
prob_s_with_ls = []
p1=1
for word in new_word_list:
    if word in freq_s.keys():
        count = freq_s[word]
    else:
        count = 0
    prob_s_with_ls.append((count + 1)/(total_cnts_features_s + total_features))
    p1=p1*((count + 1)/(total_cnts_features_s + total_features))
dict(zip(new_word_list,prob_s_with_ls))
p1
```

Out[34]: 5.505602363429078e-22

In [36]:
```
prob_p_with_ls = []
p2=1
for word in new_word_list:
    if word in freq_p.keys():
        count = freq_p[word]
    else:
        count = 0
    prob_p_with_ls.append((count + 1)/(total_cnts_features_p + total_features))
    p2=p2*((count + 1)/(total_cnts_features_p + total_features))
dict(zip(new_word_list,prob_p_with_ls))
p2
```

Out[36]: 9.46135175695248e-25

In [38]:
```
prob_b_with_ls = []
p3=1
for word in new_word_list:
    if word in freq_b.keys():
        count = freq_b[word]
    else:
        count = 0
    prob_b_with_ls.append((count + 1)/(total_cnts_features_b + total_features))
    p3=p3*((count + 1)/(total_cnts_features_b + total_features))
dict(zip(new_word_list,prob_b_with_ls))
p3
```

Out[38]: 2.415724362071022e-22

Business has least value;Document belongs to Business Category

RESULTS: Test Doc belongs to ‘Business’ Category.

Challenging Exercise 1- Take any large text corpora, do the necessary preprocessing and built a Naïve Bayes Classifier and draw the interferences.

Proposed Algorithm:

1. First, we will load all the necessary libraries required for our classifier.
2. Next, we load the data (training and test sets):
3. We calculate the total number of classes and samples we have.
4. The next step consists of building the Naive Bayes classifier and finally training the model. We will convert the collection of text documents (train and test sets) into a matrix of token counts.
5. To implement that text transformation we will use the make_pipeline function. This will internally transform the text data and then the model will be fitted using the transformed data**.**
6. The last line of code predicts the labels of the test set**.**
7. Finally, we build the multi-class confusion matrix to see if the model is good or if the model predicts correctly only specific text categories.
8. Hence, we classify whatever sentence we choose.

Pseudocode:

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
   'This is the first document.',
   'This document is the second document.',
   'And this is the third one.',
   'Is this the first document?',
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
```

```
['and', 'document', 'first', 'is', 'one', 'second', '
the', 'third', 'this']
```

```
print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

Data Structure Proposed: Arrays

Libraries Used: Pandas, Seaborn, Matplotlib, pyplot, sklearn.

IMPLEMENTATION CODE AND RESULTS:

In [3]:
```
import numpy as np, pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
sns.set() # use seaborn plotting style
```

In [4]:
```
# Load the dataset
data = fetch_20newsgroups()
# Get the text categories
text_categories = data.target_names
# define the training set
train_data = fetch_20newsgroups(subset="train", categories=text_categories)
# define the test set
test_data = fetch_20newsgroups(subset="test", categories=text_categories)
```

In [5]:
```
print("We have {} unique classes".format(len(text_categories)))
print("We have {} training samples".format(len(train_data.data)))
print("We have {} test samples".format(len(test_data.data)))
```

```
We have 20 unique classes
We have 11314 training samples
We have 7532 test samples
```

In [6]:
```
# let's have a look as some training data
print(test_data.data[5])
```

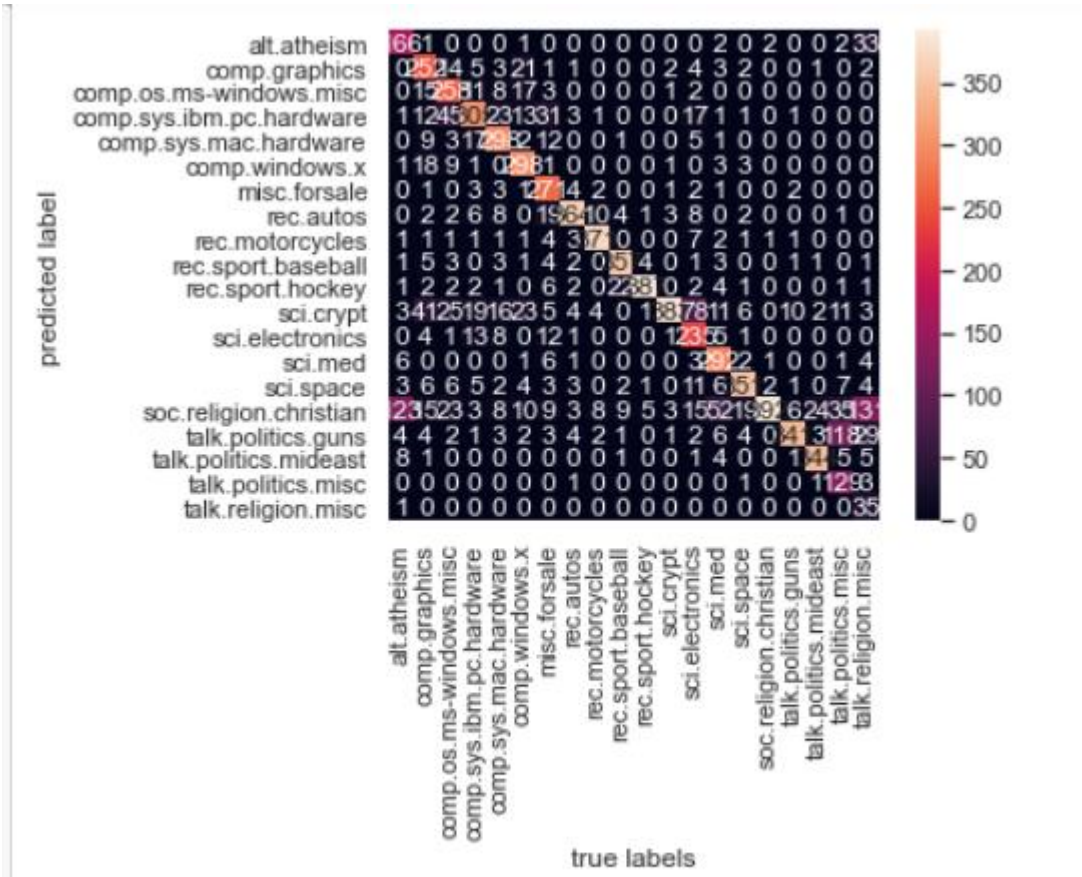
```
From: banschbach@vms.ocom.okstate.edu
Subject: Re: Candida(yeast) Bloom, Fact or Fiction
Organization: OSU College of Osteopathic Medicine
Lines: 91
Nntp-Posting-Host: vms.ocom.okstate.edu

In article <1rp8p1$2d3@usenet.INS.CWRU.Edu>, esd3@po.CWRU.Edu (Elisabeth S. Davidson) writes:
>
> In a previous article, banschbach@vms.ocom.okstate.edu () says:
>>least a few "enlightened" physicians practicing in the U.S.  It's really
>>too bad that most U.S. medical schools don't cover nutrition because if
>>they did, candida would not be viewed as a non-disease by so many in the
>>medical profession.
>
> Case Western Reserve Med School teaches nutrition in its own section as
> well as covering it in other sections as they apply (i.e. B12
> deficiency in neuro as a cause of neuropathy, B12 deficiency in
> hematology as a cause of megaloblastic anemia), yet I sill
> hold the viewpoint of mainstream medicine:  candida can cause
> mucocutaneous candidiasis, and, in already very sick patients
> with damaged immune systems like AIDS and cancer patients,
> systemic candida infection.  I think "The Yeast Connection" is
> a bunch of hooey.  What does this have to do with how well
> nutrition is taught, anyway?

Elisabeth, let's set the record straight for the nth time, I have not read
"The Yeast Connection".  So anything that I say is not due to brainwashing
by this "hated" book.  It's okay I guess to hate the book, by why hate me?
Elisabeth, I'm going to quote from Zinsser's Microbiology, 20th Edition.
A book that you should be familiar with and not "hate". "Candida species
colonize the mucosal surfaces of all humans during birth or shortly
thereafter.  The risk of endogenous infection is clearly ever present.
Indeed, candidiasis occurs worldwide and is the most common systemic
mycosis."  Neutrophils play the main role in preventing a systemic
infection(candidiasis) so you would have to have a low neutrophil count or
"sick" neutrophils to see a systemic infection.  Poor diet and persistent
parasitic infestation set many third world residents up for candidiasis.
Your assessment of candidiasis in the U.S. is correct and I do not dispute
it.
```

In [7]:
```
# Build the model
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
# Train the model using the training data
model.fit(train_data.data, train_data.target)
# Predict the categories of the test data
predicted_categories = model.predict(test_data.data)
```

In [8]:
```
print(np.array(test_data.target_names)[predicted_categories])
```

```
['rec.autos' 'sci.crypt' 'alt.atheism' ... 'rec.sport.baseball'
 'comp.sys.ibm.pc.hardware' 'soc.religion.christian']
```

In [9]:
```
# plot the confusion matrix
mat = confusion_matrix(test_data.target, predicted_categories)
sns.heatmap(mat.T, square = True, annot=True, fmt = "d", xticklabels=train_data.target_names,yticklabels=train_data.target_names)
plt.xlabel("true labels")
plt.ylabel("predicted label")
plt.show()
print("The accuracy is {}".format(accuracy_score(test_data.target, predicted_categories)))
```




```
The accuracy is 0.7738980350504514
```

In [11]:
```
# custom function to find results
def my_predictions(my_sentence, model):
    all_categories_names = np.array(data.target_names)
    prediction = model.predict([my_sentence])
    return all_categories_names[prediction]

my_sentence = "jesus"
print(my_predictions(my_sentence, model))
['soc.religion.christian']
my_sentence = "Are you an atheist?"
print(my_predictions(my_sentence, model))
```

```
['soc.religion.christian']
['alt.atheism']
```

RESULTS: Hence we are able to infer our results and find out the confusion matrix and the value of accuracy which is around 0.77 or 77%.