TEXT AND WEB INTELLIGENCE ANALYTICS

LAB MANUAL

SUBMITTED BY	**MAHIMA MUNJAL**
ROLL NUMBER	**17CSU098**
CLASS	**CSE-VIII-B**
GROUP	**C3**
SESSION	**2020-2021**
FACULTY	**DR. VAISHALI KALRA**

Department of Computer Science and Engineering
The Northcap University, Sector-23, Gurugram, Haryana

INDEX

COURSE-CURRICULUM EXPERIMENTS				
S.NO.	**Experiment**	**Date**	**Grade**	**Signature**
1	Frequency Distribution	29-01-2021		
2	Stopwords	05/02/2021		
3	Exploring Names Corpus	12/02/2021		
4	Similarity metrices	16/02/2021		
5	Lesk Algorithm for Word Sense Disambiguation	24/02/2021		
6	Implement LDA with BOW and TF-IDF features and compare the results	01/04/2021		
7	Implementation of KNN, Naive Bayes and Multinominal Naive Bayes.	08/04/2021		
8	Implementation of K means, K Medoids and Hierarchical Clustering algorithms.	02/05/2021		
9	Implement homophily for social media analysis.	09/05/2021		
VALUE ADDED EXPERIMENTS				
1	Chunking, Chinking and Named Entity Recognition	03/03/2021		
2	Tokenization and Exploring Gutenberg Corpus	22-01-2021		

EXPERIMENT NO. 1

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_1_17CSU098.ipynb
Date: 29 January 2021
Faculty Signature:
Grade:

Objectives:

1. Import the corpus Shakespeare and find the frequency of each word in the file dream.xml.
2. Find 5 most frequently occurring words from the file dream.xml.
3. Import wordnet corpus from the available nltk corpus list and find out the sysnset of word bank. Also find the definition and example of first sysnset in the list.

Background Study:

nltk.FreqDist( )

A frequency distribution records the number of times each outcome of an experiment has occurred. For example, a frequency distribution could be used to record the frequency of each word type in a document. Formally, a frequency distribution can be defined as a function mapping from each sample to the number of times that sample occurred as an outcome.

Wordnet

Wordnet is a lexical database of semantic relations between words in more than 200 languages. WordNet links words into semantic relations including synonyms, hyponyms, and meronym.

Outcome: Students will be able to learn the concepts of nltk.freqdist(), collections in python and sysnets in wordnet library.

Problem Statement:

1. **Import the corpus Shakespeare and find the frequency of each word in the file dream.xml.**

   CODE AND OUTPUT:

```
import nltk
from nltk.corpus import shakespeare
```

```
[ ] nltk.download('shakespeare')
```

```
[nltk_data] Downloading package shakespeare to /root/nltk_data...
[nltk_data]   Package shakespeare is already up-to-date!
True
```

```
shakespeare.fileids()
```

```
['a_and_c.xml',
 'dream.xml',
 'hamlet.xml',
 'j_caesar.xml',
 'macbeth.xml',
 'merchant.xml',
 'othello.xml',
 'r_and_j.xml']
```

```
[ ] dream= nltk.corpus.shakespeare.words('dream.xml')
    dream
```

```
['A',
 'Midsummer',
 'Night',
 "'",
 's',
 'Dream',
 'Dramatis',
 'Personae',
 'THESEUS',
```

```
[ ] len(dream)
```

```
21538
```

```
[ ] #cleaning data of punctuations
    import string
    l=string.punctuation.split()
    no_punct_dream=[words for words in dream if words not in string.punctuation]
    no_punct_dream
```

```
['A',
 'Midsummer',
 'Night',
 's',
 'Dream',
 'Dramatis',
 'Personae',
 'THESEUS',
 'Duke',
 'of',
 'Athens',
 'EGEUS',
 'father',
 'to',
 'Hermia',
 'LYSANDER',
 'DEMETRIUS',
 'in',
 'love',
 'with',
 'Hermia',
 'PHILOSTRATE',
 'master',
 'of',
```

```
[ ]  #Finding frequency
     fdist=nltk.FreqDist(w.lower() for w in no_punct_dream)
     fdist
```

```
FreqDist({'a': 273,
          'midsummer': 2,
          'night': 52,
          's': 133,
          'dream': 16,
          'dramatis': 1,
          'personae': 1,
          'theseus': 67,
          'duke': 14,
          'of': 272,
          'athens': 27,
          'egeus': 17,
          'father': 14,
          'to': 340,
          'hermia': 103,
          'lysander': 103,
          'demetrius': 101,
          'in': 243,
          'love': 117,
          'with': 177,
          'philostrate': 14,
          'master': 8,
          'the': 563,
          'revels': 5,
          'quince': 55,
          'carpenter': 1,
          'snug': 10,
          'joiner': 4,
          'bottom': 69,
          'weaver': 3,
          'flute': 19,
          'bellows': 3,
```

2. **Find 5 most frequently occurring words from the file dream.xml.**

   CODE AND OUTPUT:

Q2. Finding most frequently occuring words from the file dream.xml.

```
[ ]  Dictionary=dict(fdist)
```

```
[ ]   from collections import Counter
      dict(Counter(Dictionary).most_common(5))
```

```
{'and': 574, 'i': 470, 'the': 563, 'to': 340, 'you': 274}
```

3. **Import wordnet corpus from the available nltk corpus list and find out the sysnset of word bank. Also find the definition and example of first sysnset in the list.**

   CODE AND OUTPUT:

```
nltk.download('wordnet')
from nltk.corpus import wordnet
syns=wordnet.synsets("Bank")
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
print("Definition of the word Bank:")
print(syns[0].definition())
print("\nExamples of the word Bank:")
print(syns[0].examples())
```

```
Definition of the word Bank:
sloping land (especially the slope beside a body of water)

Examples of the word Bank:
['they pulled the canoe up on the bank', 'he sat on the bank of the river and watched the currents']
```

EXPERIMENT NO. 2

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_2_17CSU098.ipynb
Date: 5 February 2021
Faculty Signature:
Grade:

Objective:
1. Print all the Arabic Stopwords.
2. Omit a given list of stop words from the total stopwords list of English language.

Background Study:

Stopwords

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.We would not want these words to take up space in our database or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that are considered stopping words.

NLTK (Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages.

Outcome: Students will be able to understand the concept of stopwords in nltk and list comprehensions in python.

Problem Statement:

1. **Print all the Arabic Stopwords.**
   CODE AND OUTPUT:

```
[1]  import nltk
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
Arabic_Stopwords = set(nltk.corpus.stopwords.words("arabic"))
Arabic_Stopwords
```

```
{'آه',
 'آها',
 'آي',
 'أف',
 'أقل',
 'أكثر',
 'ألا',
 'أم',
 'أما',
 'أن',
```

2. Omit a given list of stop words from the total stopwords list of English language.

CODE AND OUTPUT:

```
English_Stopwords = list(nltk.corpus.stopwords.words("english"))
English_Stopwords
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
```

```
l=['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves','Mahima','Munjal']
print(l)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'Mahima', 'Munjal']
```

```
for i in l:
  if i in English_Stopwords:
    English_Stopwords.remove(i)
English_Stopwords
```

```
['you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers'
```

EXPERIMENT NO. 3

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_3_17CSU098_.ipynb
Date: 12 February 2021
Faculty Signature:
Grade:

Objectives:

1. Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names.
2. From the names corpus, combine all the labelled male and female names and print any 20.
3. Print the definition and examples of any one English language word using WordNet corpus.

Background Study:

Text Corpus

A text corpus is a large and structured set of texts (nowadays usually electronically stored and processed). Text corpora are used to do statistical analysis and hypothesis testing, checking occurrences, or validating linguistic rules within a specific language territory.

Wordnet Corpus

Wordnet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (sysnets), each expressing a distinct concept. Sysnets are interlinked by means of conceptual-semantic and lexical relations.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings.
Outcome: Students will be able to explore names corpus, understand the concept of labelling the data and learn about sysnets in Wordnet corpus.

Problem Statement:

1. **Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names.**

   CODE AND OUTPUT:

```
[ ]  nltk.download('names')
     from nltk.corpus import names
     print("\nNumber of male names:")
     print (len(names.words('male.txt')))
     print("Number of female names:")
     print (len(names.words('female.txt')))
```

```
[nltk_data] Downloading package names to /root/nltk_data...
[nltk_data]   Package names is already up-to-date!

Number of male names:
2943
Number of female names:
5001
```

2. **From the names corpus, combine all the labelled male and female names and print any 20.**

   CODE AND OUTPUT:

```
[ ]  Male_names = names.words('male.txt')
     Female_names = names.words('female.txt')
     print("\nFirst 15 male names:")
     print (male_names[0:15])
     print("\nFirst 15 female names:")
     print (female_names[0:15])
```

```
First 15 male names:
['Aamir', 'Aaron', 'Abbey', 'Abbie', 'Abbot', 'Abbott', 'Abby', 'Abdel', 'Abdul', 'Abdulkarim', 'Abdullah', 'Abe', 'Abel', 'Abelard', 'Abner']

First 15 female names:
['Abagael', 'Abagail', 'Abbe', 'Abbey', 'Abbi', 'Abbie', 'Abby', 'Abigael', 'Abigail', 'Abigale', 'Abra', 'Acacia', 'Ada', 'Adah', 'Adaline']
```

```
Male= names.words('male.txt')
Female = names.words('female.txt')

Label_Male= [(str(name), 'male') for name in Male]
Label_Female = [(str(name), 'female') for name in Female]

print("Male  :",Label_Male)
print("Female  :",Label_Female)
```

```
Male  : [('Aamir', 'male'), ('Aaron', 'male'), ('Abbey', 'male'), ('Abbie', 'male'), ('Abbot', 'male'), ('Abbott', 'male'), ('Abby', 'male'), ('Abdel', 'male'),
Female  : [('Abagael', 'female'), ('Abagail', 'female'), ('Abbe', 'female'), ('Abbey', 'female'), ('Abbi', 'female'), ('Abbie', 'female'), ('Abby', 'female'), ('
```

```
[ ]  import random
     Label_All = Label_Male + Label_Female
     random.shuffle(Label_All)
     print("First 20 random labeled combined names:")
     Label_All[:20]
```

```
First 20 random labeled combined names:
[('Carlota', 'female'),
 ('Joell', 'female'),
 ('Erinna', 'female'),
 ('Norm', 'male'),
 ('Lysandra', 'female'),
 ('Shirley', 'female'),
 ('Blondell', 'female'),
 ('Dosi', 'female'),
 ('Chicky', 'female'),
 ('Gloriane', 'female'),
 ('Mead', 'male'),
 ('Konstance', 'female'),
 ('Raina', 'female'),
 ('Sissie', 'female'),
 ('Constancia', 'female'),
 ('Kurtis', 'male'),
 ('Tedi', 'female'),
 ('Mickie', 'female'),
 ('Evangelin', 'female'),
 ('Ibby', 'female')]
```

3. Print the definition and examples of any one English language word using WordNet corpus.

CODE AND OUTPUT:

```
nltk.download('wordnet')
from nltk.corpus import wordnet
syns=wordnet.synsets("Telephone")
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
print("Definition of the word Telephone:")
print(syns[0].definition())
print("\nExamples of the word Telephone:")
print(syns[0].examples())
```

```
Definition of the word Telephone:
electronic equipment that converts sound into electrical signals that can be transmitted over distances and then converts received signals back into sounds

Examples of the word Telephone:
['I talked to him on the telephone']
```

EXPERIMENT NO. 4

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_4_17CSU098.ipynb
Date: 16 February 2021
Faculty Signature:
Grade:

Objective: To implement Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer.

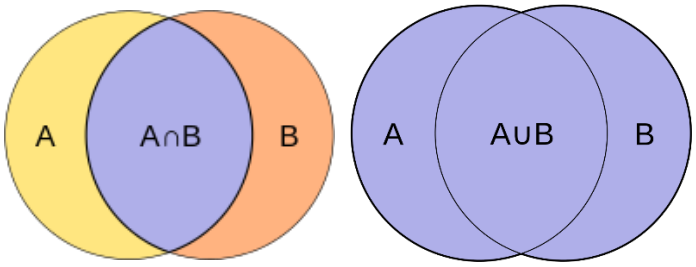
Background Study:

Levenshtein distance

The Levenshtein distance is a string metric for measuring difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

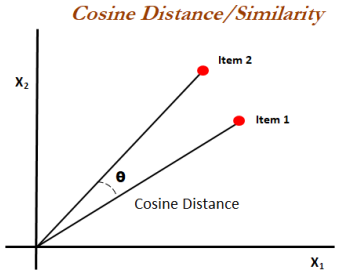
Jaccard similarity

The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.



Cosine similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.



TF-IDF Vectorizer

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

TF-IDF for a word in a document is calculated by multiplying two different metrics:

- The **term frequency** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.

- The **inverse document frequency** of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.

- So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

CountVectorizer

CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

	about	all	cent	cents	money	new	old	one	two
doc	1	1	3	1	1	1	1	1	1

In theory (a)

Index	0	1	2	3	4	5	6	7	8
doc	1	1	3	1	1	1	1	1	1

In practice (b)

Outcome: Students will be able to demonstrate Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer.

Problem Statement:

1. **Demonstrate the computation of Similarity Metrics such as Jaccard, Levenshtein and Cosine.**

CODE AND OUTPUT :

```
#Jaccard Similarity - Method 1

def jaccard_similarity(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection)/union
```

```
data1=input()
data2=input()
list1 = data1.split(" ")
list2 = data2.split(" ")
print("List 1 ",list1)
print("List 2 ",list2)
```

```
Mahima Munjal is a good girl
Mahima Munjal studies at NCU
List 1  ['Mahima', 'Munjal', 'is', 'a', 'good', 'girl']
List 2  ['Mahima', 'Munjal', 'studies', 'at', 'NCU']
```

```
jaccard_similarity(list1, list2)
```

```
0.2222222222222222
```

```
#Jaccard Similarity - Method 2

def jaccard_similarities(list1, list2):
    s1 = set(list1)
    s2 = set(list2)
    return float(len(s1.intersection(s2)) / len(s1.union(s2)))
```

```
jaccard_similarities(list1, list2)
```

```
0.2222222222222222
```

```
[24] #Jaccard Similarity -Method 3

     import numpy as np
     from sklearn.metrics import jaccard_score

     jaccard_score([1,1,0,0],[0,1,0,1])
```

```
     0.3333333333333333
```

```
[11] pip install jaccard-index
```

```
     Collecting jaccard-index
       Downloading https://files.pythonhosted.org/packages/e7/66/a066229192ef1323b5a36
     Installing collected packages: jaccard-index
     Successfully installed jaccard-index-0.0.3
```

```
[15] #Jaccard Index

     rom jaccard_index.jaccard import jaccard_index
     jaccard_index("Mahima","Mahima Munjal")
```

```
     0.5
```

LEVENSHTEIN DISTANCE

```
[7]  #Levenshtein Distance

     import enchant

     data1 = "Mahima Munjal is a good girl."
     data2 = "Mahima Munjal"

     enchant.utils.levenshtein(data1,data2)
```

```
16
```

EDIT DISTANCE

```
#Edit Distance

import nltk
from nltk.metrics import *

edit_distance("Mahima Munjal is a good girl.","Mahima Munjal")
```

```
16
```

COSINE SIMILARITY

```
[8]  import nltk
     nltk.download('punkt')
     nltk.download('stopwords')
```

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

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
I = input("Enter first string: ").lower()
II = input("Enter second string: ").lower()
```

```
Enter first string: Mahima Munjal is a good girl
Enter second string: Mahima Munjal studies at NCU and works at Nagarro
```

```
# tokenization
I_list = word_tokenize(I)
II_list = word_tokenize(II)
print('First List', I_list)
print('Second List', II_list)
```

```
First List ['mahima', 'munjal', 'is', 'a', 'good', 'girl']
Second List ['mahima', 'munjal', 'studies', 'at', 'ncu', 'and', 'works', 'at', 'nagarro']
```

```
#removing stopwords

sw = stopwords.words('english')
l1 =[];l2 =[]
I_set = {w for w in I_list if not w in sw}
II_set = {w for w in II_list if not w in sw}
print('First Set', I_set)
print('Second Set', II_set)
```

```
First Set {'good', 'mahima', 'girl', 'munjal'}
Second Set {'nagarro', 'ncu', 'studies', 'munjal', 'works', 'mahima'}
```

```
# Forming a set containing keywords of both strings

rvector = I_set.union(II_set)
for w in rvector:
    if w in I_set: l1.append(1)
    else: l1.append(0)
    if w in II_set: l2.append(1)
    else: l2.append(0)
c = 0
```

```
# cosine formula
for i in range(len(rvector)):
        c+= l1[i]*l2[i]
cosine = c / float((sum(l1)*sum(l2))**0.5)

print('Cosine Similarity Value :',cosine)
```

```
Cosine Similarity Value : 0.4082482904638631
```

COSINE SIMILARITY MATRIX

```
[ ]  I = input("Enter first string: ").lower()
     II = input("Enter second string: ").lower()
```

```
Enter first string: Mahima Munjal is a student of Text and Web Analytics
Enter second string: Mahima Munjal is a student of The Northcap University
```

```
[ ]  documents = [I,II]
```

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer
     import pandas as pd
```

```
[ ]  count_vectorizer = CountVectorizer(stop_words='english')
     count_vectorizer = CountVectorizer()
     sparse_matrix = count_vectorizer.fit_transform(documents)


     doc_term_matrix = sparse_matrix.todense()
     df = pd.DataFrame(doc_term_matrix,
                       columns=count_vectorizer.get_feature_names(),
                       index=['I', 'II'])
     df
```

	analytics	and	is	mahima	munjal	northcap	of	student	text	the	university	web
I	1	1	1	1	1	0	1	1	1	0	0	1
II	0	0	1	1	1	1	1	1	0	1	1	0

```
[ ]  from sklearn.metrics.pairwise import cosine_similarity
     print(cosine_similarity(df, df))
```

```
[[1.         0.58925565]
 [0.58925565 1.        ]]
```

2. Calculate the TF-IDF vectorizer on 2 documents.

CODE AND OUTPUT:

Q2. CALCULATE THE TFIDF VECTORIZOR ON 2 DOCUMENTS.

```
[ ]  I = input("Enter first string: ").lower()
     II = input("Enter second string: ").lower()
```

```
Enter first string: Mahima Munjal loves to watch movies
Enter second string: Mahima Munjal loves to do yoga
```

```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer
     corpus = [I,II]
     vectorizer = TfidfVectorizer()
     X = vectorizer.fit_transform(corpus)
     print('Vectorizer Features :',vectorizer.get_feature_names())
     print('Vectorizer Shape: ',X.shape)
```

```
Vectorizer Features : ['do', 'loves', 'mahima', 'movies', 'munjal', 'to', 'watch', 'yoga']
Vectorizer Shape:  (2, 8)
```

```
[28] print("TF-IDF Scores")
     print(X)
```

```
TF-IDF Scores
  (0, 3)        0.49844627974580596
  (0, 6)        0.49844627974580596
  (0, 5)        0.35464863330313684
  (0, 1)        0.35464863330313684
  (0, 4)        0.35464863330313684
  (0, 2)        0.35464863330313684
  (1, 7)        0.49844627974580596
  (1, 0)        0.49844627974580596
  (1, 5)        0.35464863330313684
  (1, 1)        0.35464863330313684
  (1, 4)        0.35464863330313684
  (1, 2)        0.35464863330313684
```

```
[42] idf=vectorizer.idf_
     idf
```

```
array([1.40546511, 1.        , 1.        , 1.40546511, 1.        ,
       1.        , 1.40546511, 1.40546511])
```

```
     dict(zip(vectorizer.get_feature_names(),idf))
```

```
{'do': 1.4054651081081644,
 'loves': 1.0,
 'mahima': 1.0,
 'movies': 1.4054651081081644,
 'munjal': 1.0,
 'to': 1.0,
 'watch': 1.4054651081081644,
 'yoga': 1.4054651081081644}
```

3. Apply the max-df, min-df param in the TF-IDF function.

CODE AND OUTPUT:

Q3. Apply the max-df, min-df param in the TF-IDF function.

```
[33] from sklearn.feature_extraction.text import CountVectorizer
```

```
[34] data = [I,II]
     count_vec = CountVectorizer(stop_words="english", analyzer='word',ngram_range=(1, 1), max_df=0.50, min_df=1, max_features=None)

     count_train = count_vec.fit(data)
     bag_of_words = count_vec.transform(data)

     print('Features :',count_vec.get_feature_names())
```

```
Features : ['movies', 'watch', 'yoga']
```

```
print("Count Vectorizer Scores")
print(bag_of_words)
```

```
Count Vectorizer Scores
  (0, 0)        1
  (0, 1)        1
  (1, 2)        1
```

4. Compute Cosine Similarity using both TF-IDF and Count vectorizer

CODE AND OUTPUT:

```
[26] from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.feature_extraction.text import TfidfTransformer
     from nltk.corpus import stopwords
     import numpy as np
     import numpy.linalg as LA
     import nltk
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
[29] train_set = ["Mahima Munjal is a good girl", "Mahima Munjal studies at The Northcap University."]    # Documents
     test_set = ["A good girl named Mahima Munjal studies at the Northcap University."]    # Query
     stopWords = stopwords.words('english')
```

```
vectorizer = CountVectorizer(stop_words = stopWords)
print('Vectorizer : ',vectorizer)
transformer = TfidfTransformer()
print('\n\nTF-IDF Transformer : ',transformer)
```

```
Vectorizer :  CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None,
                stop_words=['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
                            'ourselves', 'you', "you're", "you've", "you'll",
                            "you'd", 'your', 'yours', 'yourself', 'yourselves',
                            'he', 'him', 'his', 'himself', 'she', "she's",
                            'her', 'hers', 'herself', 'it', "it's", 'its',
                            'itself', ...],
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)


TF-IDF Transformer :  TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=False, use_idf=True)
```

```
#Using Count Vectorizer

trainVectorizerArray = vectorizer.fit_transform(train_set).toarray()
testVectorizerArray = vectorizer.transform(test_set).toarray()
print ('Fit Vectorizer to train set \n', trainVectorizerArray)
print ('\n\nTransform Vectorizer to test set\n', testVectorizerArray)
```

```
Fit Vectorizer to train set
 [[1 1 1 1 0 0 0]
 [0 0 1 1 1 1 1]]


Transform Vectorizer to test set
 [[1 1 1 1 1 1 1]]
```

```
#Using TF-IDF

transformer.fit(trainVectorizerArray)
print('Fit transformer to train set \n',transformer.transform(trainVectorizerArray).toarray())
transformer.fit(testVectorizerArray)
tfidf = transformer.transform(testVectorizerArray)
print('\n\nFit transformer to test set\n',tfidf.todense())
```

```
Fit transformer to train set
 [[0.57615236 0.57615236 0.40993715 0.40993715 0.         0.
  0.        ]
 [0.         0.         0.35520009 0.35520009 0.49922133 0.49922133
  0.49922133]]


Fit transformer to test set
 [[0.37796447 0.37796447 0.37796447 0.37796447 0.37796447 0.37796447
  0.37796447]]
```

EXPERIMENT NO. 5

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_5_17CSU098.ipynb
Date: 24 February 2021
Faculty Signature:
Grade:

Objective: Implementation of the Lesk algorithm for Word Sense Disambiguation.

Background Study:

Lesk Algorithm

The Lesk algorithm assumes that words in each "neighbourhood" (section of text) will tend to share a common topic. A simplified version of the Lesk algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighbourhood.

An implementation might look like this:

1. for every sense of the word being disambiguated one should count the number of words that are in both neighbourhood of that word and in the dictionary definition of that sense
2. the sense that is to be chosen is the sense which has the biggest number of this count.

Outcome: Students will be able to demonstrate how to Lesk algorithm works.

Problem Statement: Implement Lesk algorithm for Word Sense Disambiguation.

CODE AND OUTPUT:

LESK WORKS CORRECTLY

```
[13] # Context 1 - Financial institution

     print ("Context-1:", sentences[0])
     answer = simple_lesk(sentences[0],'bank')
     print ("Sense:", answer)
     print ("Definition : ", answer.definition())

     # Correct Output - Financial Institution printed
     # No disambiguity
```

```
Context-1: I went to the bank to deposit my money
Sense: Synset('depository_financial_institution.n.01')
Definition :  a financial institution that accepts deposits and channels the money into lending activities
```

```
# Context 2 - River Bank

print ("Context-2:", sentences[1])
answer = simple_lesk(sentences[1],'bank')
print ("Sense:", answer)
print ("Definition : ", answer.definition())


# Correct Output - River Bank or sloping land printed
# No disambiguity
```

```
Context-2: The river bank had a lot of fishes and crocodiles.
Sense: Synset('bank.n.01')
Definition :  sloping land (especially the slope beside a body of water)
```

LESK WORKS INCORRECTLY

```
[17] new_sentences = ['The workers at the plant were overworked','The plant was no longer bearing flowers','The workers at the industrial plant were overworked']

     #ambiguous word - Plant
```

```
[18] # Context 1 - Industrial plant

     print ("Context-1:", new_sentences[0])
     answer = simple_lesk(new_sentences[0],'plant')
     print ("Sense:", answer)
     print ("Definition : ", answer.definition())

     # Incorrect output - Industrial plant not printed
     # Disambiguity occured
```

```
Context-1: The workers at the plant were overworked
Sense: Synset('plant.v.06')
Definition :  put firmly in the mind
```

[19]
```
# Context 2 - Tree/Seedling/Sapling

print ("Context-2:", new_sentences[1])
answer = simple_lesk(new_sentences[1],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())

# Correct output - Plant bearing flower sense printed
# No disambiguity
```

```
Context-2: The plant was no longer bearing flowers
Sense: Synset('plant.v.01')
Definition :  put or set (seeds, seedlings, or plants) into the ground
```

```
# Context 3 - Industrial plant (Added the word industrial before plant in
# Context 1)

print ("Context-3:", new_sentences[2])
answer = simple_lesk(new_sentences[2],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())

# Correct output - Industrial plant printed
# (Disambiguity resolved by adding the word "industrial" in the sentence.)
```

```
Context-3: The workers at the industrial plant were overworked
Sense: Synset('plant.n.01')
Definition :  buildings for carrying on industrial labor
```

EXPERIMENT NO. 6

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_6_17CSU098.ipynb
Date: 01 April 2021
Faculty Signature:
Grade:

Objectives:

1. Implement LDA with Bag of Words
2. Implement LDA with TF-IDF
3. Compare the accuracy using Score values make the analysis.

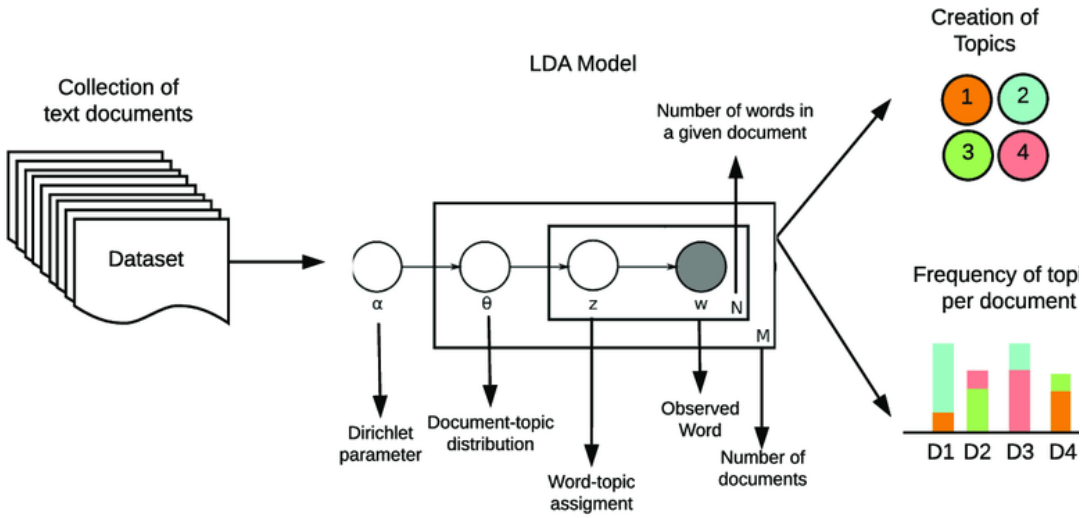
Dataset: https://www.kaggle.com/therohk/million-headlines

Background Study:

Latent Dirichlet Allocation (LDA)

It is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.

For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a topic model and belongs to the machine learning toolbox and in wider sense to the artificial intelligence toolbox.



Outcome: Students will be able to demonstrate how LDA works using Bag of Words and using TF-IDF.

Problem Statement: Implement LDA using BOW and TF-IDF on Million News Dataset

CODE AND OUTPUT:

```
import numpy as np
import pandas as pd
```

```
[3] from google.colab import files
    uploaded = files.upload()
```

Choose Files abcnews-date-text.csv

- **abcnews-date-text.csv**(application/vnd.ms-excel) - 62726972 bytes, last modified: 5/14/2021 - 100% done

```
Saving abcnews-date-text.csv to abcnews-date-text (2).csv
```

```
[4] import io
    data = pd.read_csv(io.BytesIO(uploaded['abcnews-date-text.csv']),error_bad_lines=False)
    data
```

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers
...	...	...
1226253	20201231	what abc readers learned from 2020 looking bac...
1226254	20201231	what are the south african and uk variants of ...
1226255	20201231	what victorias coronavirus restrictions mean f...
1226256	20201231	whats life like as an american doctor during c...
1226257	20201231	womens shed canberra reskilling unemployed pan...

```
data_text = data[['headline_text']]
data_text['index'] = data_text.index
documents = data_text
documents
```

	headline_text	index
0	aba decides against community broadcasting lic...	0
1	act fire witnesses must be aware of defamation	1
2	a g calls for infrastructure protection summit	2
3	air nz staff in aust strike for pay rise	3
4	air nz strike to affect australian travellers	4
...	...	...
1226253	what abc readers learned from 2020 looking bac...	1226253
1226254	what are the south african and uk variants of ...	1226254
1226255	what victorias coronavirus restrictions mean f...	1226255
1226256	whats life like as an american doctor during c...	1226256
1226257	womens shed canberra reskilling unemployed pan...	1226257

1226258 rows × 2 columns

```
print(len(documents))
```

```
1226258
```

```
[7] documents.isnull().sum()
```

```
headline_text    0
index            0
dtype: int64
```

Data Preprocessing

```
[8] import gensim
    from gensim.utils import simple_preprocess
    from gensim.parsing.preprocessing import STOPWORDS
    from nltk.stem import WordNetLemmatizer, SnowballStemmer
    from nltk.stem.porter import *
    import numpy as np
    np.random.seed(2018)
    import nltk
    nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
[13] stemmer = SnowballStemmer('english')
     def lemmatize_stemming(text):
         return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))
     def preprocess(text):
         result = []
         for token in gensim.utils.simple_preprocess(text):
             if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
                 result.append(lemmatize_stemming(token))
         return result
```

```
doc_sample = documents[documents['index'] == 5000].values[0][0]
print('Original document: ')
words = []
for word in doc_sample.split(' '):
    words.append(word)
print(words)
print('\n\n Tokenized and Lemmatized document: ')
print(preprocess(doc_sample))
```

```
Original document:
['staffing', 'rail', 'stations', 'would', 'mean', 'ticket', 'hike']


 Tokenized and Lemmatized document:
['staff', 'rail', 'station', 'mean', 'ticket', 'hike']
```

```
[11] processed_docs = documents['headline_text'].map(preprocess)
     processed_docs
```

```
0                          [decid, commun, broadcast, licenc]
1                                          [wit, awar, defam]
2                       [call, infrastructur, protect, summit]
3                                [staff, aust, strike, rise]
4                        [strike, affect, australian, travel]
                                         ...
1226253                          [reader, learn, look, year]
1226254                    [south, african, variant, covid]
1226255          [victoria, coronaviru, restrict, mean, year]
1226256           [what, life, like, american, doctor, covid]
1226257    [women, shed, canberra, reskil, unemploy, pandem]
Name: headline_text, Length: 1226258, dtype: object
```

Bag of Words

```
[15] dictionary = gensim.corpora.Dictionary(processed_docs)
     count = 0
     for k, v in dictionary.iteritems():
         print(k, v)
         count += 1
         if count > 10:
             break
```

```
0 broadcast
1 commun
2 decid
3 licenc
4 awar
5 defam
6 wit
7 call
8 infrastructur
9 protect
10 summit
```

```
[16] dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)
```

```
[17] bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
     bow_corpus[5000]
```

```
[(13, 1), (237, 1), (1375, 1), (1405, 1), (1430, 1), (1596, 1)]
```

```
[19] bow_doc = bow_corpus[5000]
     for i in range(len(bow_doc)):
         print("Word {} (\"{}\") appears {} time.".format(bow_doc[i][0],dictionary[bow_doc[i][0]], bow_doc[i][1]))
```

```
Word 13 ("staff") appears 1 time.
Word 237 ("station") appears 1 time.
Word 1375 ("ticket") appears 1 time.
Word 1405 ("rail") appears 1 time.
Word 1430 ("mean") appears 1 time.
Word 1596 ("hike") appears 1 time.
```

TF-IDF

```
[20] from gensim import corpora,models
     tfidf=models.TfidfModel(bow_corpus)
```

```
[21] corpus_tfidf=tfidf[bow_corpus]
```

```
[22] from pprint import pprint
     for doc in corpus_tfidf:
         pprint(doc)
         break
```

```
[(0, 0.5852942020878993),
 (1, 0.38405854933668493),
 (2, 0.5017732999224691),
 (3, 0.5080878695349914)]
```

Running LDA using Bag of Words

```
[23] lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=10, id2word=dictionary, passes=2, workers=2)
```

```
[25] for index, score in sorted(lda_model[bow_corpus[5000]], key=lambda tup: -1*tup[1]):
         print("\nScore: {}\t \nTopic: {}".format(score, lda_model.print_topic(index, 10)))
```

```
Score: 0.44285327196121216
Topic: 0.028*"govern" + 0.018*"health" + 0.017*"state" + 0.016*"chang" + 0.016*"coronaviru" + 0.013*"hospit" + 0.013*"canberra" + 0.010*"say" + 0.010*"minist" + 0.010*"care"

Score: 0.29999372363090515
Topic: 0.059*"polic" + 0.032*"death" + 0.027*"sydney" + 0.016*"shoot" + 0.015*"miss" + 0.015*"investig" + 0.015*"arrest" + 0.014*"melbourn" + 0.013*"royal" + 0.013*"scott"

Score: 0.15714284777641296
Topic: 0.034*"elect" + 0.021*"nation" + 0.018*"tasmania" + 0.015*"say" + 0.012*"tasmanian" + 0.010*"darwin" + 0.010*"presid" + 0.009*"vote" + 0.009*"labor" + 0.009*"beach"

Score: 0.014291977509856224
Topic: 0.025*"coronaviru" + 0.025*"china" + 0.022*"live" + 0.021*"news" + 0.020*"market" + 0.014*"busi" + 0.011*"australian" + 0.011*"power" + 0.010*"price" + 0.010*"drum"

Score: 0.014289580285549164
Topic: 0.015*"worker" + 0.013*"countri" + 0.012*"help" + 0.012*"announc" + 0.011*"commun" + 0.011*"indigen" + 0.010*"work" + 0.010*"pandem" + 0.010*"polit" + 0.010*"find"

Score: 0.014285714365541935
Topic: 0.027*"donald" + 0.015*"attack" + 0.014*"kill" + 0.013*"school" + 0.013*"interview" + 0.012*"children" + 0.012*"andrew" + 0.011*"releas" + 0.011*"border" + 0.011*"australian"

Score: 0.014285714365541935
Topic: 0.038*"trump" + 0.034*"queensland" + 0.033*"coronaviru" + 0.027*"victoria" + 0.020*"south" + 0.018*"bushfir" + 0.017*"coast" + 0.014*"hous" + 0.013*"covid" + 0.012*"crash"

Score: 0.014285714365541935
Topic: 0.027*"case" + 0.027*"court" + 0.024*"covid" + 0.018*"face" + 0.018*"charg" + 0.017*"coronaviru" + 0.016*"murder" + 0.015*"restrict" + 0.014*"jail" + 0.014*"trial"
```

Running LDA using TF-IDF

```
[26] lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf, num_topics=10, id2word = dictionary, passes = 2, workers=4)
```

```
[28] for index, score in sorted(lda_model_tfidf[bow_corpus[4310]], key=lambda tup: -1*tup[1]):
         print("\nScore: {}\t \nTopic: {}".format(score, lda_model_tfidf.print_topic(index, 10)))
```

```
Score: 0.5632221698760986
Topic: 0.009*"interview" + 0.007*"andrew" + 0.007*"elect" + 0.007*"violenc" + 0.006*"updat" + 0.006*"financ" + 0.005*"liber" + 0.005*"coronaviru" + 0.005*"domest" + 0.005*"inquest"

Score: 0.19738005101680756
Topic: 0.012*"countri" + 0.012*"rural" + 0.009*"hour" + 0.008*"govern" + 0.008*"news" + 0.007*"nation" + 0.006*"climat" + 0.006*"stori" + 0.006*"chang" + 0.006*"monday"

Score: 0.15188860893249512
Topic: 0.014*"drum" + 0.013*"polic" + 0.009*"charg" + 0.009*"murder" + 0.008*"fatal" + 0.008*"shoot" + 0.007*"driver" + 0.007*"crash" + 0.007*"sport" + 0.007*"arrest"

Score: 0.012503090314567089
Topic: 0.014*"coronaviru" + 0.013*"market" + 0.007*"australian" + 0.006*"rise" + 0.006*"price" + 0.006*"australia" + 0.006*"busi" + 0.006*"share" + 0.006*"queensland" + 0.006*"michael"

Score: 0.012501217424869537
Topic: 0.009*"guilti" + 0.009*"charg" + 0.009*"assault" + 0.008*"tuesday" + 0.008*"court" + 0.007*"street" + 0.007*"sexual" + 0.007*"abus" + 0.007*"woman" + 0.007*"sentenc"

Score: 0.01250113733112812
Topic: 0.009*"kill" + 0.007*"northern" + 0.006*"crash" + 0.006*"dead" + 0.005*"social" + 0.005*"protest" + 0.005*"australia" + 0.005*"march" + 0.005*"june" + 0.005*"korea"

Score: 0.012501135468482971
Topic: 0.008*"thursday" + 0.008*"christma" + 0.007*"age" + 0.007*"peter" + 0.007*"care" + 0.006*"polic" + 0.006*"brief" + 0.006*"peopl" + 0.006*"daniel" + 0.006*"novemb"

Score: 0.012501110322773457
Topic: 0.011*"scott" + 0.010*"morrison" + 0.010*"wednesday" + 0.009*"news" + 0.006*"grand" + 0.006*"april" + 0.006*"quiz" + 0.006*"search" + 0.005*"miss" + 0.005*"ash"
```

EXPERIMENT NO. 7

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_7_17CSU098.ipynb
Date: 4 April 2021
Faculty Signature:
Grade:

Objectives:

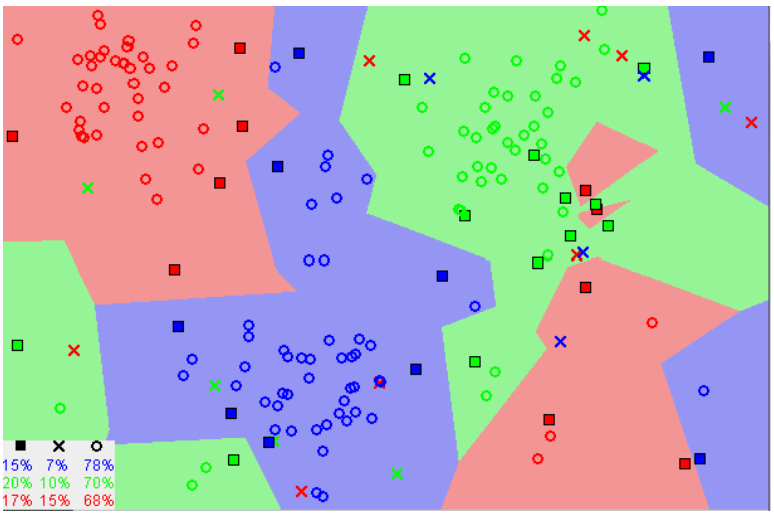
1. **Perform KNN, Naïve Bayes and Multinomial Naïve Bayes on 20 Newsgroup Dataset.**

Dataset: https://www.kaggle.com/crawford/20-newsgroups

Background Study:

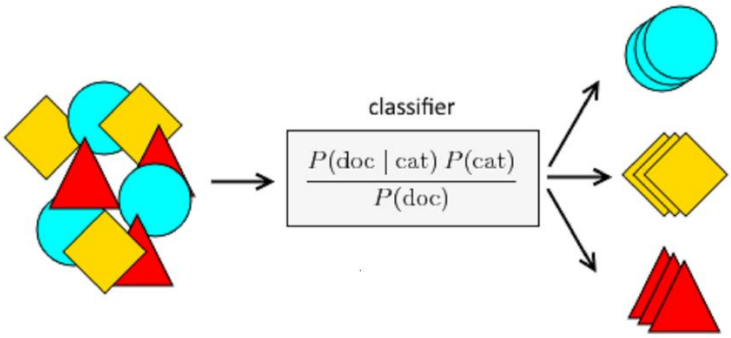
K- Nearest Neighbours(KNN)

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.



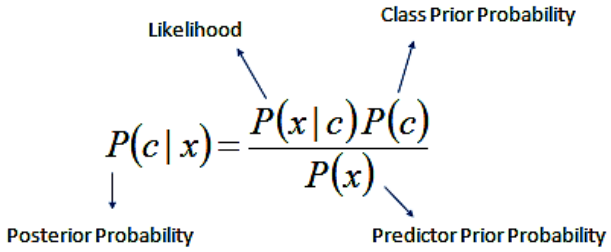
Naïve Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.



Multinomial Naïve Bayes

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Outcome: Students will be able to understand the difference between the implementations of KNN, Naïve Bayes and Multinomial Naïve Bayes.

Problem Statement: Implement KNN, Naïve Bayes and Multinomial Naïve Bayes on20 Newsgroup datset and compare their accuracy scores.

CODE AND OUTPUT:

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

```
categories = ['rec.motorcycles', 'sci.electronics',
              'comp.graphics', 'sci.med']


train_data = fetch_20newsgroups(subset='train',
                                categories=categories, shuffle=True, random_state=42)

print(train_data.target_names)
```

```
print("\n".join(train_data.data[0].split("\n")[:3]))
print(train_data.target_names[train_data.target[0]])


for t in train_data.target[:10]:
    print(train_data.target_names[t])
```

```
['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']
From: kreyling@lds.loral.com (Ed Kreyling 6966)
Subject: Sun-os and 8bit ASCII graphics
Organization: Loral Data Systems
comp.graphics
comp.graphics
comp.graphics
rec.motorcycles
comp.graphics
sci.med
sci.electronics
sci.electronics
comp.graphics
rec.motorcycles
sci.electronics
```

COUNT VECTORIZER AND TFIDF VECTORIZER

```
[ ]  count_vect = CountVectorizer()
     X_train_counts = count_vect.fit_transform(train_data.data)

     tfidf_transformer = TfidfTransformer()
     X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

K-Nearest Neighbour

```
[ ]  knn = KNeighborsClassifier(n_neighbors=7)
     clf = knn.fit(X_train_tfidf, train_data.target)
     docs_new = ['I want to buy a computer.', 'I have a television at my home.']

     X_new_counts = count_vect.transform(docs_new)
     X_new_tfidf = tfidf_transformer.transform(X_new_counts)
```

```
predicted = clf.predict(X_new_tfidf)

for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, train_data.target_names[category]))
```

```
'I want to buy a computer.' => sci.electronics
'I have a television at my home.' => comp.graphics
```

```
text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', knn),
])

# Fitting our train data to the pipeline
text_clf.fit(train_data.data, train_data.target)

# Test data
test_data = fetch_20newsgroups(subset='test',
                               categories=categories, shuffle=True, random_state=42)
docs_test = test_data.data
# Predicting our test data
predicted = text_clf.predict(docs_test)
print('Accuracy :',np.mean(predicted == test_data.target)*100,'%')
```

```
Accuracy : 82.67766497461929 %
```

Naive bayes

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_train_tfidf, train_data.target, test_size=0.30, random_state=42)

print('Training Data Shape:', X_train.shape)
print('Testing Data Shape: ', X_test.shape)
```

```
Training Data Shape: (1656, 35653)
Testing Data Shape:  (711, 35653)
```

```
y1=X_train.todense()
y1
```

```
matrix([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ],
        ...,
        [0.        , 0.00508825, 0.        , ..., 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ]])
```

```
from sklearn.naive_bayes import GaussianNB
lr_model = GaussianNB()
lr_model.fit(y1, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
y2=X_test.todense()
y2
```

```
matrix([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]])
```

```
predictions = lr_model.predict(y2)
predictions
```

```
array([2, 0, 2, 1, 2, 3, 0, 1, 1, 3, 3, 2, 3, 1, 3, 3, 2, 0, 0, 3, 0, 3,
       0, 2, 3, 1, 2, 1, 0, 0, 3, 3, 1, 1, 3, 2, 0, 0, 3, 3, 1, 1, 0, 3,
       0, 1, 3, 0, 1, 3, 3, 0, 2, 0, 3, 3, 3, 0, 3, 1, 1, 1, 2, 0, 2, 2,
       3, 2, 1, 1, 2, 0, 0, 0, 2, 2, 0, 3, 2, 1, 2, 2, 1, 3, 3, 3, 3, 1,
       1, 0, 3, 1, 1, 1, 3, 1, 0, 1, 3, 0, 3, 1, 0, 0, 0, 0, 3, 1, 0, 1,
       2, 0, 3, 1, 2, 0, 1, 3, 1, 0, 1, 1, 2, 0, 3, 0, 3, 3, 1, 0, 1, 2,
       3, 1, 3, 2, 2, 1, 2, 3, 0, 1, 1, 0, 2, 0, 1, 3, 3, 3, 0, 1, 1, 1,
       2, 1, 1, 0, 1, 3, 0, 2, 2, 2, 0, 1, 3, 1, 2, 1, 1, 2, 1, 2, 3, 3,
       1, 3, 2, 0, 1, 1, 2, 1, 3, 3, 0, 2, 3, 2, 2, 1, 0, 3, 3, 1, 3, 1,
       1, 0, 0, 3, 2, 3, 1, 0, 0, 0, 0, 0, 0, 3, 2, 3, 1, 3, 2, 2, 3, 1,
       2, 1, 0, 1, 3, 0, 1, 0, 3, 2, 1, 0, 2, 1, 3, 1, 1, 0, 0, 2, 3, 3,
       2, 0, 1, 1, 1, 1, 0, 1, 1, 2, 2, 2, 3, 3, 3, 1, 1, 0, 1, 1, 0, 3,
```

CONFUSION MATRIX

```
[ ]  from sklearn import metrics
     print(metrics.confusion_matrix(y_test,predictions))
```

```
    [[150   1   9   7]
     [  4 177   3   5]
     [ 18   4 144   6]
     [  1   2   8 172]]
```

CLASSIFICATION SUMMARY

```
[ ]  print(metrics.classification_report(y_test,predictions))
```

```
                  precision    recall  f1-score   support

               0       0.95      0.96      0.96       167
               1       0.97      0.99      0.98       189
               2       0.95      0.95      0.95       172
               3       0.98      0.96      0.97       183

        accuracy                           0.97       711
       macro avg       0.97      0.97      0.97       711
    weighted avg       0.97      0.97      0.97       711
```

```
[ ]  print('Accuracy :',metrics.accuracy_score(y_test,predictions)*100,'%')
```

```
    Accuracy : 90.43600562587905 %
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_train_tfidf, train_data.target, test_size=0.30, random_state=42)

print('Training Data Shape:', X_train.shape)
print('Testing Data Shape: ', X_test.shape)
```

```
Training Data Shape: (1656, 35653)
Testing Data Shape:  (711, 35653)
```

```
from sklearn.naive_bayes import MultinomialNB
lr_model = MultinomialNB()
lr_model.fit(X_train, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
predictions = lr_model.predict(X_test)
predictions
```

```
array([2, 2, 2, 1, 2, 3, 0, 1, 1, 3, 3, 2, 3, 1, 3, 3, 2, 0, 0, 3, 0, 1,
       0, 2, 3, 1, 3, 1, 0, 0, 3, 3, 1, 1, 3, 2, 0, 0, 3, 3, 1, 1, 2, 3,
       0, 1, 3, 0, 1, 3, 3, 0, 2, 0, 1, 3, 3, 0, 3, 1, 1, 1, 2, 0, 2, 3,
       2, 2, 1, 1, 2, 0, 0, 0, 2, 2, 0, 3, 2, 1, 2, 2, 1, 3, 3, 3, 1, 1,
       1, 0, 3, 1, 1, 1, 3, 1, 0, 1, 3, 0, 3, 2, 0, 0, 0, 0, 3, 1, 0, 1,
       2, 0, 3, 1, 2, 2, 1, 3, 1, 0, 1, 1, 2, 0, 3, 3, 1, 3, 1, 0, 1, 2,
       3, 1, 3, 2, 3, 1, 2, 3, 0, 1, 1, 0, 2, 0, 3, 2, 3, 3, 2, 1, 1, 1,
       2, 1, 1, 0, 1, 3, 0, 2, 2, 2, 0, 1, 3, 1, 2, 1, 1, 2, 1, 2, 3, 3,
       1, 3, 2, 0, 1, 1, 2, 1, 0, 3, 0, 2, 3, 2, 2, 1, 0, 3, 0, 1, 3, 1,
       1, 2, 3, 3, 2, 3, 1, 0, 0, 0, 0, 2, 0, 3, 2, 3, 1, 3, 2, 2, 3, 1,
       2, 1, 0, 1, 3, 0, 1, 0, 3, 2, 1, 0, 1, 1, 3, 1, 1, 0, 0, 2, 3, 3,
       2, 0, 0, 1, 1, 1, 0, 1, 1, 2, 2, 3, 3, 3, 3, 1, 1, 0, 1, 1, 0, 3,
       1, 0, 0, 2, 3, 0, 2, 3, 0, 1, 2, 0, 0, 1, 3, 0, 3, 1, 2, 2, 2, 1,
```

CONFUSION MATRIX

```
[ ]  from sklearn import metrics
     print(metrics.confusion_matrix(y_test,predictions))
```

```
[[161   0   4   2]
 [  0 187   1   1]
 [  5   3 164   0]
 [  3   2   3 175]]
```

CLASSIFICATION SUMMARY

```
[ ]  print(metrics.classification_report(y_test,predictions))
```

```
                precision    recall  f1-score   support

             0       0.95      0.96      0.96       167
             1       0.97      0.99      0.98       189
             2       0.95      0.95      0.95       172
             3       0.98      0.96      0.97       183

      accuracy                           0.97       711
     macro avg       0.97      0.97      0.97       711
  weighted avg       0.97      0.97      0.97       711
```

```
[ ]  print("Accuracy:  ",metrics.accuracy_score(y_test,predictions)*100,'%')
```

```
Accuracy:   96.62447257383965 %
```

Result: Multinomial Naïve Bayes (96 %) >Naïve Bayes (90%) > KNN(82%)

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: **https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_8_17CSU098.ipynb**
Date: 2 May 2021
Faculty Signature:
Grade:

Objectives:

1. Implementation of K Means, K Medoids and Hierarchical Clustering algorithms on text data.

Background Study:

K-Means

K-Means Clustering is an unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters in such a way that each dataset belongs to the group which has similar properties. Here K defines the number of pre-defined clusters that need to be created in the process.

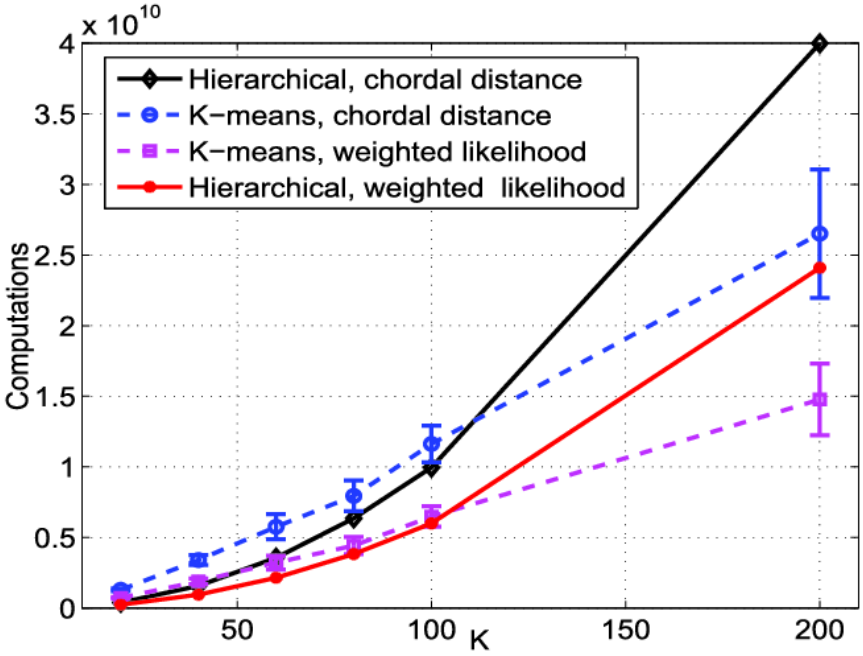
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

K-Medoids

K-medoids chooses actual data points as centers (medoids or exemplars), and thereby allows for greater interpretability of the cluster centers than in *k*-means, where the center of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster). Furthermore, *k*-medoids can be used with arbitrary dissimilarity measures, whereas *k*-means generally requires Euclidean distance for efficient solutions.

Hierarchical Clustering

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.



Outcome: Student will be able to differentiate between the three clustering algorithms and compare their accuracies.
Problem Statement: Implement K-Means, K-Medoids and Hierarchical Clustering Algorithms on text data and plot visualizations for the same.

CODE AND OUTPUT:

Importing Libraries

```
[ ]  pip install scikit-learn-extra
```

```
    Collecting scikit-learn-extra
      Downloading https://files.pythonhosted.org/packages/82/1e/79cd1a2b60a6cd6a7a9c4719f947e264ae391644adce3ddecf73afd5233d/scikit_learn_extra-0.2.0-cp37-cp37m-manylinux2010_x86_64.whl (1.7MB)
         |████████████████████████████████| 1.7MB 5.3MB/s
    Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.19.5)
    Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.4.1)
    Collecting scikit-learn>=0.23.0
      Downloading https://files.pythonhosted.org/packages/a8/eb/a48f25c967526b66d5f1fa7a984594f0bf0a5afafa94a8c4dbc317744620/scikit_learn-0.24.2-cp37-cp37m-manylinux2010_x86_64.whl (22.3MB)
         |████████████████████████████████| 22.3MB 2.2MB/s
    Collecting threadpoolctl>=2.0.0
      Downloading https://files.pythonhosted.org/packages/f7/12/ec3f2e203afa394a149911729357aa48affc59c20e2c1c8297a60f33f133/threadpoolctl-2.1.0-py3-none-any.whl
    Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.0.1)
    Installing collected packages: threadpoolctl, scikit-learn, scikit-learn-extra
      Found existing installation: scikit-learn 0.22.2.post1
        Uninstalling scikit-learn-0.22.2.post1:
          Successfully uninstalled scikit-learn-0.22.2.post1
    Successfully installed scikit-learn-0.24.2 scikit-learn-extra-0.2.0 threadpoolctl-2.1.0
```

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import metrics
from sklearn_extra.cluster import KMedoids
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
from sklearn.preprocessing import normalize
```

```
categories = ['rec.motorcycles', 'sci.electronics', 'comp.graphics', 'sci.med']

# sklearn provides us with subset data for training and testing
train_data = fetch_20newsgroups(subset='train',
                                categories=categories, shuffle=True, random_state=

print(train_data.target_names)

print("\n".join(train_data.data[0].split("\n")[:3]))
print(train_data.target_names[train_data.target[0]])

# Let's look at categories of our first ten training data
for t in train_data.target[:10]:
    print(train_data.target_names[t])
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']
From: kreyling@lds.loral.com (Ed Kreyling 6966)
Subject: Sun-os and 8bit ASCII graphics
Organization: Loral Data Systems
comp.graphics
comp.graphics
comp.graphics
rec.motorcycles
comp.graphics
sci.med
sci.electronics
sci.electronics
comp.graphics
rec.motorcycles
sci.electronics
```

K-Means

```
[ ] dataset = fetch_20newsgroups(subset='all', categories=categories,shuffle=True, random_state=42)
    data=dataset.data
    labels = dataset.target
    true_k=10
```

```
[ ] print(data[400])
```

```
From: morley@suncad.camosun.bc.ca (Mark Morley)
Subject: Medical Images via Gopher?
Nntp-Posting-Host: suncad.camosun.bc.ca
Organization: Camosun College, Victoria B.C, Canada
X-Newsreader: Tin 1.1 PL4
Lines: 11

A few days back someone posted info on a gopher site where you could
search for medical graphics, etc.  Could someone please repost or mail me
a copy?  I'd greatly appreciate it.  Thanks!

Mark

======================================================================
 Mark Morley, UNIX/SUN Manager           NET: morley@camosun.bc.ca
 Camosun College - Interurban Campus     TEL: (604) 370-4601
 4461 Interurban Road  Room 143-Tech     FAX: (604) 370-3660
 Victoria, B.C.  Canada   V8X 3X1
```

```
[ ] labels
```

```
array([3, 3, 0, ..., 0, 0, 3])
```

```
[ ] vectorizer = TfidfVectorizer(max_df=0.5,
                                 min_df=2,
                                 stop_words='english')
    X = vectorizer.fit_transform(dataset.data)
```

```
[ ] km = KMeans(n_clusters=20, init='k-means++', max_iter=100, n_init=1)
    km.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=100,
       n_clusters=20, n_init=1, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
[ ] km.inertia_
```

```
3743.787347394425
```

```
[ ] km.cluster_centers_
```

```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.25277450e-04, 7.00221765e-04, 5.21675406e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [9.97370748e-04, 3.79988747e-04, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.30898235e-03, 6.34550100e-04, 0.00000000e+00, ...,
```

```
[ ] km.n_iter_
```

```
32
```

```
[ ] order_centroids = km.cluster_centers_.argsort()[:, ::-1]
    terms = vectorizer.get_feature_names()
    for i in range(true_k):
      print("cluster ", i+1)
      for ind in order_centroids[i,:20]:
          print('%s' % terms[ind])
      print()
```

```
cluster  1
information
nz
keith
eis
zealand
calstate
com
university
thanks
new
syndrome
cs
spock
cri
pat
know
host
nntp
posting
```

```
cluster  2
list
cview
joe
cactus
copy
rider
org
mailing
registration
simms
disks
senner
urc
program
bmw
tue
temp
user
nl
protection

cluster  3
ca
bnr
duke
infante
hydro
bc
acpub
tony
writes
morgan
dod
bike
jody
```

```
cluster  4
geb
banks
pitt
gordon
cs
surrender
shameful
cadre
dsl
chastity
n3jxp
intellect
skepticism
pittsburgh
soon
univ
science
computer
reply
lyme

cluster  5
sun
east
green
ed
egreen
com
microsystems
ninjaite
cruncher
biker
nc
8302
460
```

```
cluster  6
com
audio
amp
use
power
circuit
output
ir
noise
radio
current
used
like
bellcore
input
high
data
voltage
cooling
dtmedin

cluster  7
bike
behanna
nec
com
dod
nj
syl
chris
bikes
honda
advice
ride
riding
zx
```

```
cluster  8
ai
uga
georgia
mcovingt
covington
athens
aisun3
michael
programs
542
706
n4tmi
7415
0358
30602
intelligence
artificial
associate
amateur
scientist

cluster  9
ibm
photography
com
krillean
3do
kirlian
pictures
article
austin
vnet
carl
writes
rchland
```

```
cluster  10
image
format
file
files
gif
images
tiff
bit
color
jpeg
program
24
graphics
convert
display
ftp
pov
formats
xv
university
```

OUTPUT RESULTS

```
[ ]  print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels, km.labels_))
     print("Completeness: %0.3f" % metrics.completeness_score(labels, km.labels_))
     print("V-measure: %0.3f" % metrics.v_measure_score(labels, km.labels_))
     print("Silhouette Coefficient: %0.3f"
           % metrics.silhouette_score(X, km.labels_, sample_size=1000))
```

```
Homogeneity: 0.416
Completeness: 0.231
V-measure: 0.297
Silhouette Coefficient: 0.009
```

K-Medoids

```
[ ]  myiris = datasets.load_iris()
     x = myiris.data
     y = myiris.target
```

```
[ ]  print(myiris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
```

Scaling and Fitting KMedoids

```
[ ]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler().fit(x)
     x_scaled = scaler.transform(x)
     kMedoids = KMedoids(n_clusters = 3, random_state = 0)
     kMedoids.fit(x_scaled)
     y_kmed = kMedoids.fit_predict(x_scaled)
```

```
[ ]  y_kmed
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

```
[ ]  kMedoids.inertia_
```

```
131.87877332824286
```

```
[ ]  kMedoids.cluster_centers_
```

```
array([[-1.02184904,  0.78880759, -1.2833891 , -1.3154443 ],
       [-0.17367395, -0.59237301,  0.42173371,  0.13250973],
       [ 1.2803405 ,  0.09821729,  0.93327055,  1.18556721]])
```

```
[ ] kMedoids.n_iter_
```

```
2
```

Evaluating clusters

```
[ ] from sklearn.metrics import silhouette_samples, silhouette_score
    kMedoids = KMedoids(n_clusters = 3, random_state = 0)
    kMedoids.fit(x_scaled)
    y_kmed = kMedoids.fit_predict(x_scaled)
    silhouette_avg = silhouette_score(x_scaled, y_kmed)
    print(silhouette_avg)
```

```
0.4590416105554613
```

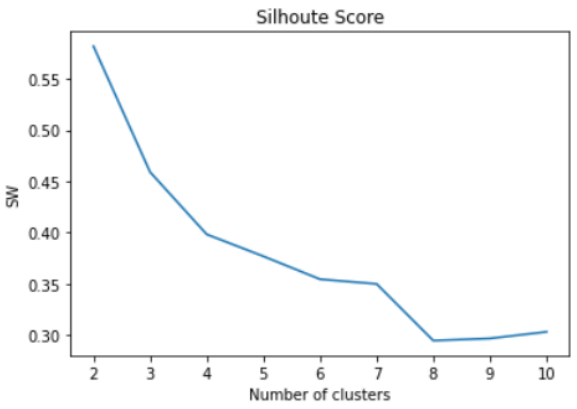
```
[ ] sample_silhouette_values = silhouette_samples(x_scaled, y_kmed)
    for i in range(3):
        ith_cluster_silhouette_values = sample_silhouette_values[y_kmed == i]
        print(np.mean(ith_cluster_silhouette_values))
```

```
0.636330614585637
0.3768888598233938
0.36213487963471125
```

Finding Number of Clusters

```
[ ] sw = []

    for i in range(2, 11):
        kMedoids = KMedoids(n_clusters = i, random_state = 0)
        kMedoids.fit(x_scaled)
        y_kmed = kMedoids.fit_predict(x_scaled)
        silhouette_avg = silhouette_score(x_scaled, y_kmed)
        sw.append(silhouette_avg)
```

```
[ ] plt.plot(range(2, 11), sw)
    plt.title('Silhoute Score')
    plt.xlabel('Number of clusters')
    plt.ylabel('SW')
    plt.show()
```



Purity Score

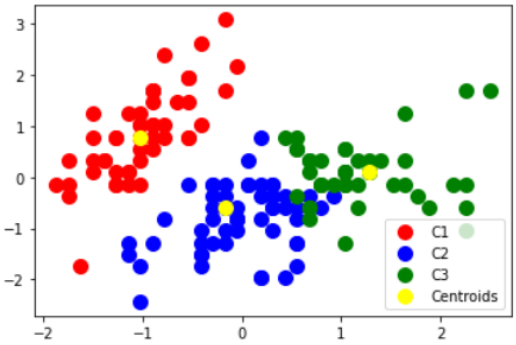
```
def purity_score(y_true, y_pred):

    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)
```

```
kmedoids = KMedoids(n_clusters=3, random_state=0).fit(x_scaled)
y_kmed = kmedoids.fit_predict(x_scaled)
purity_score(y,y_kmed)
```

```
0.84
```

```
plt.scatter(x_scaled[y_kmed == 0, 0], x_scaled[y_kmed == 0, 1], s = 100, c = 'red', label = 'C1')
plt.scatter(x_scaled[y_kmed == 1, 0], x_scaled[y_kmed == 1, 1], s = 100, c = 'blue', label = 'C2')
plt.scatter(x_scaled[y_kmed == 2, 0], x_scaled[y_kmed == 2, 1], s = 100, c = 'green', label = 'C3')
plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbf3892b750>
```



Hierarchal Clustering

```
[ ] from google.colab import files
    uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

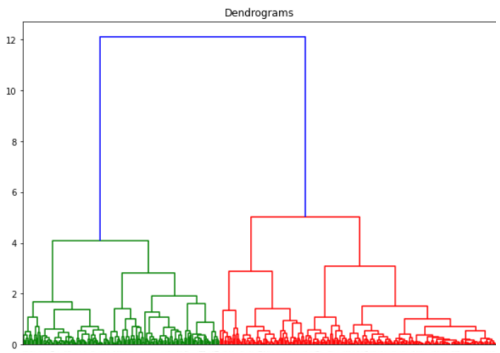
```
Saving Wholesale customers data.csv to Wholesale customers data.csv
```

```
[ ] import io
    data = pd.read_csv(io.BytesIO(uploaded['Wholesale customers data.csv']))
```

Data Normalization

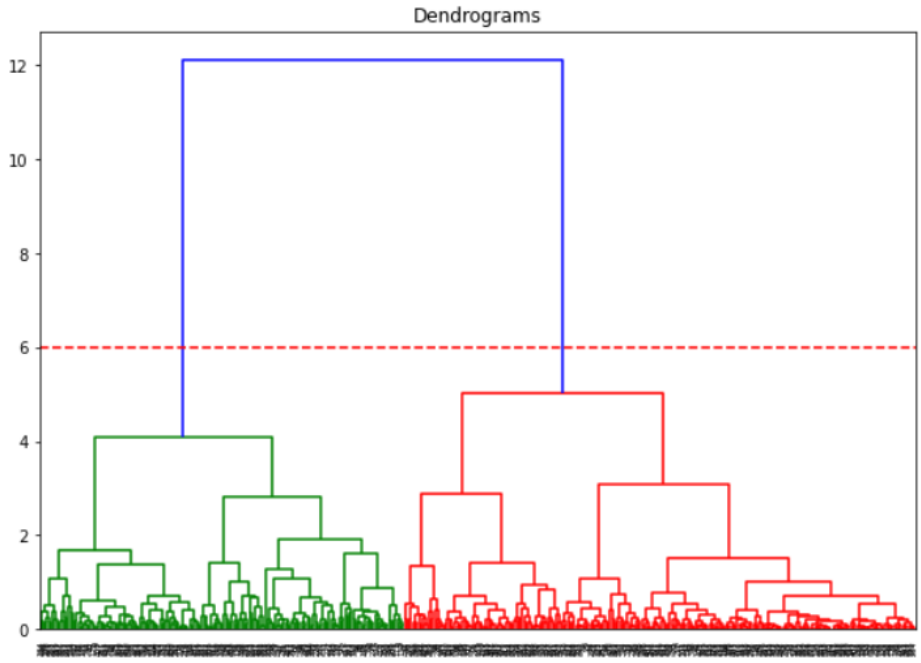
```
[ ] data_scaled = normalize(data)
    data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
```

```
[ ]
    plt.figure(figsize=(10, 7))
    plt.title("Dendrograms")
    dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
```



```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')
```

```
<matplotlib.lines.Line2D at 0x7fbf2f2e82d0>
```

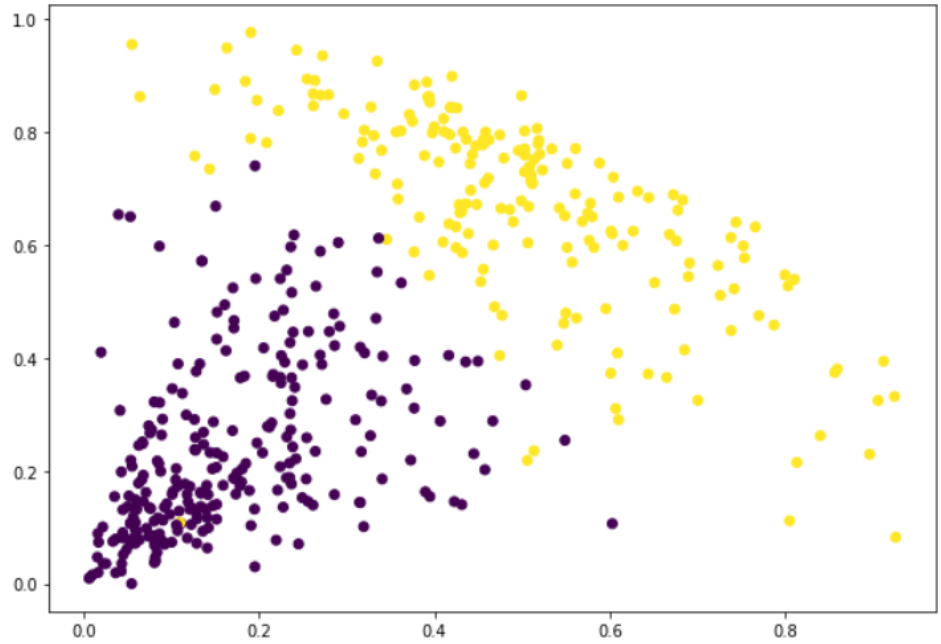



```
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(data_scaled)
```

```
array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1])
```

```
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
```

```
<matplotlib.collections.PathCollection at 0x7fbf2eeed690>
```



EXPERIMENT NO. 9

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/Lab_Practical_1_17CSU098.ipynb
Date: 9 May 2021
Faculty Signature:
Grade:

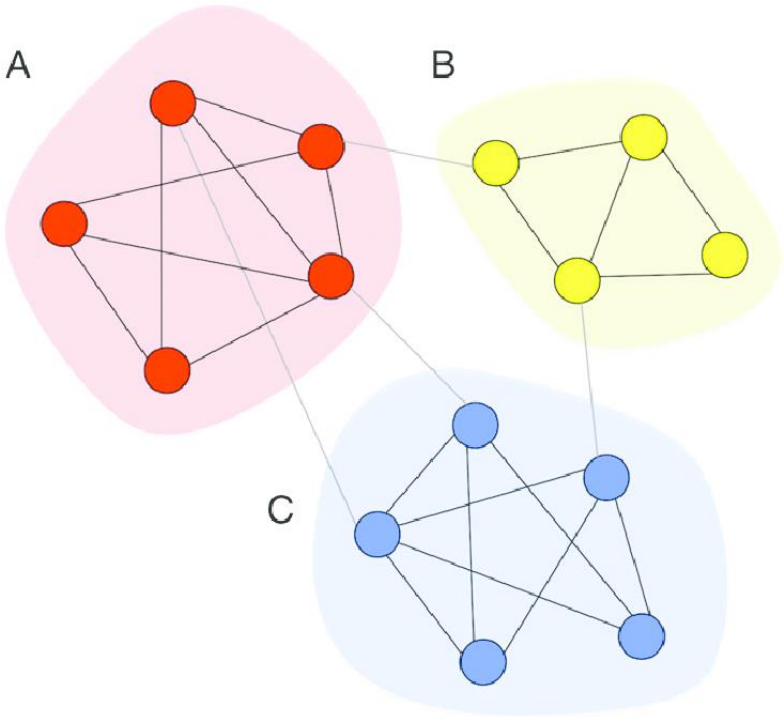
Objectives:

1. Implement homophily for social media analysis.

Background Study:

Homophily

Network homophily refers to the theory in network science which states that, based on node attributes, similar nodes may be more likely to attach to each other than dissimilar ones. The hypothesis is linked to the model of preferential attachment and it draws from the phenomenon of homophily in social sciences and much of the scientific analysis of the creation of social ties based on similarity comes from network science. In fact, empirical research seems to indicate the frequent occurrence of homophily in real networks.



Outcome: Students will be able to perform a case study on Social Network Analysis for a population.

Problem Statement:
In this case study, you will investigate homophily of several characteristics of individuals connected in social networks in rural India.

You will calculate the chance homophily for an arbitrary characteristic. Homophily is the proportion of edges in the network whose constituent nodes share that characteristic. How much homophily do we expect by chance? If characteristics are distributed completely randomly, the probability that two nodes x and y share characteristic a is the probability both nodes have characteristic a, which is the frequency of a squared. The total probability that nodes x and y share their characteristic is therefore the sum of the frequency of each characteristic in the network.

DATA CAMP CASE STUDY QUESTIONS AND ANSWERS

Q1. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will calculate the chance homophily for an arbitrary characteristic. Homophily is the proportion of edges in the network whose constituent nodes share that characteristic. How much homophily do we expect by chance? If characteristics are distributed completely randomly, the probability that two nodes `x` and `y` share characteristic `a` is the probability both nodes have characteristic `a`, which is the frequency of `a` squared. The total probability that nodes `x` and `y` share their characteristic is therefore the sum of the frequency of each characteristic in the network. For example, in the dictionary `favorite_colors` provided, the frequency of `red` and `blue` is 1/3 and 2/3 respectively, so the chance homophily is (1/3)^2+(2/3)^2 = 5/9.

100 XP

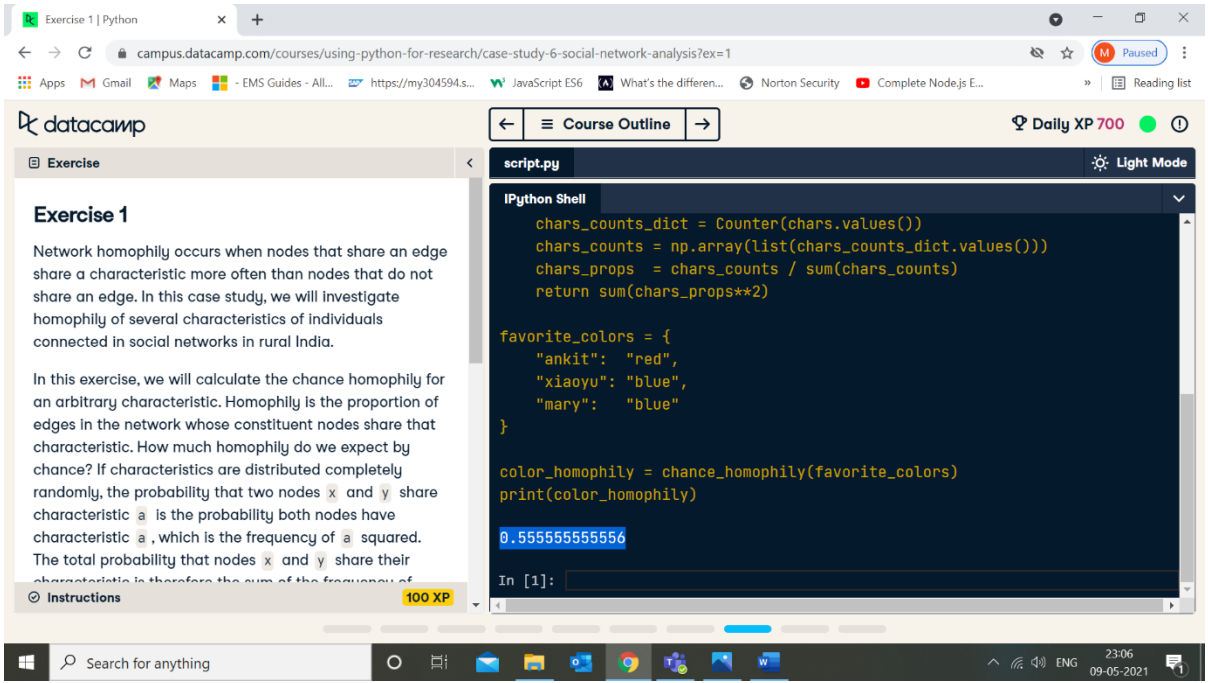
- Create a function that takes a dictionary `chars` with personal IDs as keys and characteristics as values, and returns a dictionary with characteristics as keys, and the frequency of their occurrence as values.
- Create a function `chance_homophily(chars)` that takes a dictionary `chars` defined as above and computes the chance homophily for that characteristic.
- A sample of three peoples' favorite colors is given in `favorite_colors`. Use your function to compute the chance homophily in this group, and store as `color_homophily`.
- Print `color_homophily`.

CODE:

```
from collections import Counter
def chance_homophily(chars):
    """
    Computes the chance homophily of a characteristic,
    specified as a dictionary, chars.
    """
    #enter your code here
    chars_counts_dict = Counter(chars.values())
    chars_counts = np.array(list(chars_counts_dict.values()))
    chars_props  = chars_counts / sum(chars_counts)
    return sum(chars_props**2)
```

```
favorite_colors = {
    "ankit":  "red",
    "xiaoyu": "blue",
    "mary":   "blue"
}

color_homophily = chance_homophily(favorite_colors)
print(color_homophily)
```

OUTPUT:



Q2. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In the remaining exercises, we will calculate and compare the actual homophily in these village to chance. In

this exercise, we subset the data into individual villages and store them.

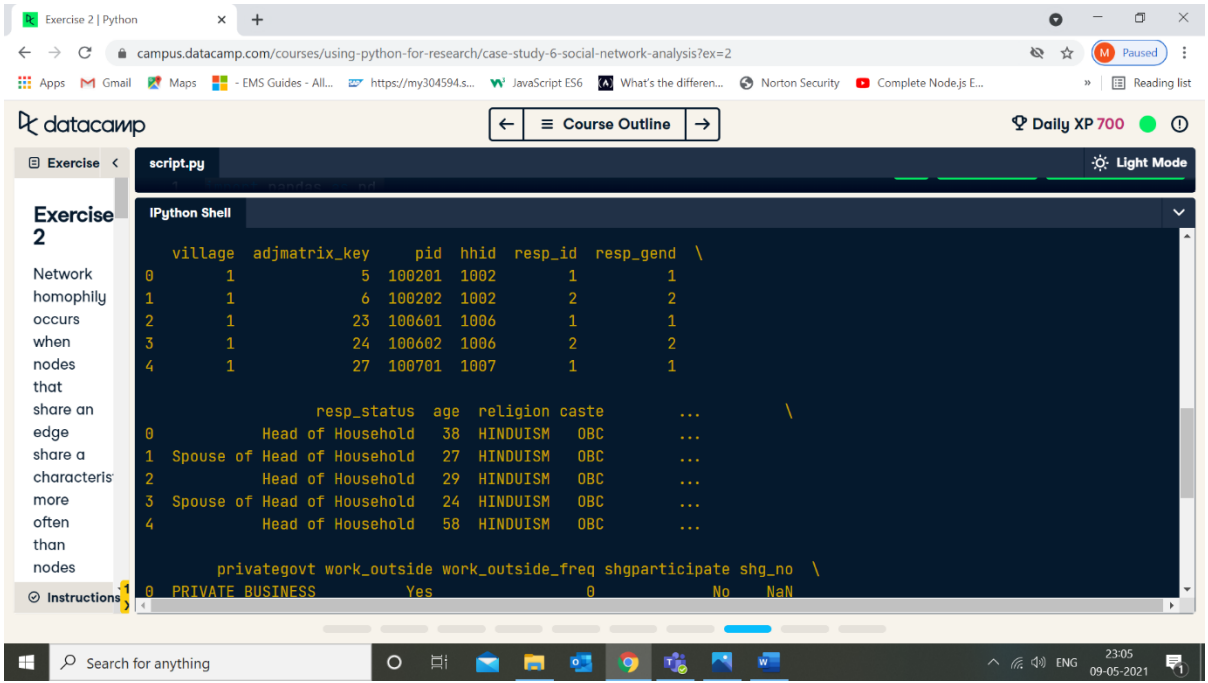
Instructions
100 XP

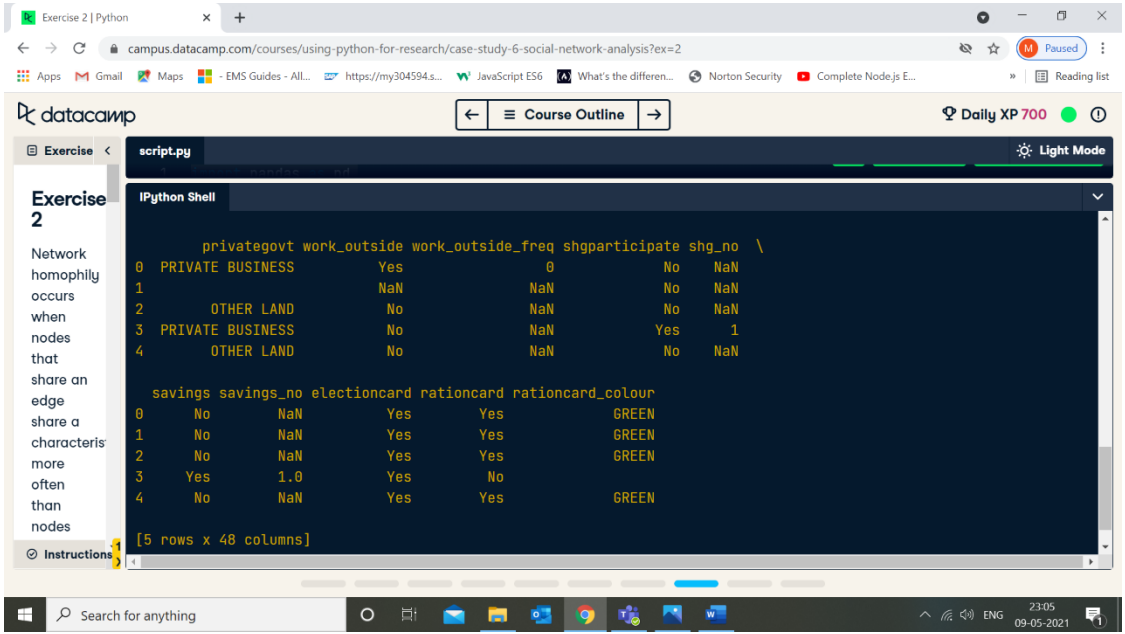
- `individual_characteristics.dta` contains several characteristics for each individual in the dataset such as age, religion, and caste. Use the `pandas` library to read in and store these characteristics as a dataframe called `df`.
- Store separate datasets for individuals belonging to Villages 1 and 2 as `df1` and `df2`, respectively.
  - Note that some attributes may be missing for some individuals. In this case study, we will ignore rows of data where some column information is missing.
- Use the `head` method to display the first few entries of `df1`.

CODE :

```
import pandas as pd
df  = pd.read_stata(data_filepath + "individual_characteristics.dta")
df1 = df[df["village"]==1]# Enter code here!
df2 = df[df["village"]==2]# Enter code here!

# Enter code here!
df1.head()
```

OUTPUT:


Exercise 2 | Python
campus.datacamp.com/courses/using-python-for-research/case-study-6-social-network-analysis?ex=2
Paused
Apps
Gmail
Maps
- EMS Guides - All...
https://my304594.s...
JavaScript ES6
What's the differen...
Norton Security
Complete Node.js E...
Reading list
datacamp
Course Outline
Daily XP 700
Exercise
script.py
Light Mode
Exercise 2
Network homophily occurs when nodes that share an edge share a characteris more often than nodes
Instructions
IPython Shell
village adjmatrix_key pid hhid resp_id resp_gend \
0 1 5 100201 1002 1 1
1 1 6 100202 1002 2 2
2 1 23 100601 1006 1 1
3 1 24 100602 1006 2 2
4 1 27 100701 1007 1 1
resp_status age religion caste ...
0 Head of Household 38 HINDUISM OBC ...
1 Spouse of Head of Household 27 HINDUISM OBC ...
2 Head of Household 29 HINDUISM OBC ...
3 Spouse of Head of Household 24 HINDUISM OBC ...
4 Head of Household 58 HINDUISM OBC ...
privategovt work_outside work_outside_freq shgparticipate shg_no \
0 PRIVATE BUSINESS Yes 0 No NaN
Search for anything
ENG
23:05
09-05-2021



Q3. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we define a few dictionaries that enable us to look up the sex, caste, and religion of members of each village by personal ID. For Villages 1 and 2, their personal IDs are stored as `pid`.

Instructions
100 XP

- Define dictionaries with personal IDs as keys and a given covariate for that individual as values. Complete this for the sex, caste, and religion covariates, for Villages 1 and 2.
- For Village 1, store these dictionaries into variables named `sex1`, `caste1`, and `religion1`.
- For Village 2, store these dictionaries into variables named `sex2`, `caste2`, and `religion2`.

CODE:

```
sex1      = df1.set_index("pid")["resp_gend"].to_dict()
caste1    = df1.set_index("pid")["caste"].to_dict()
religion1 = df1.set_index("pid")["religion"].to_dict()

sex2      = df2.set_index("pid")["resp_gend"].to_dict()
caste2    = df2.set_index("pid")["caste"].to_dict()
religion2 = df2.set_index("pid")["religion"].to_dict()
```

Q4. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will print the chance homophily of several characteristics of Villages 1 and 2. The function `chance_homophily` is still defined from Exercise 1.

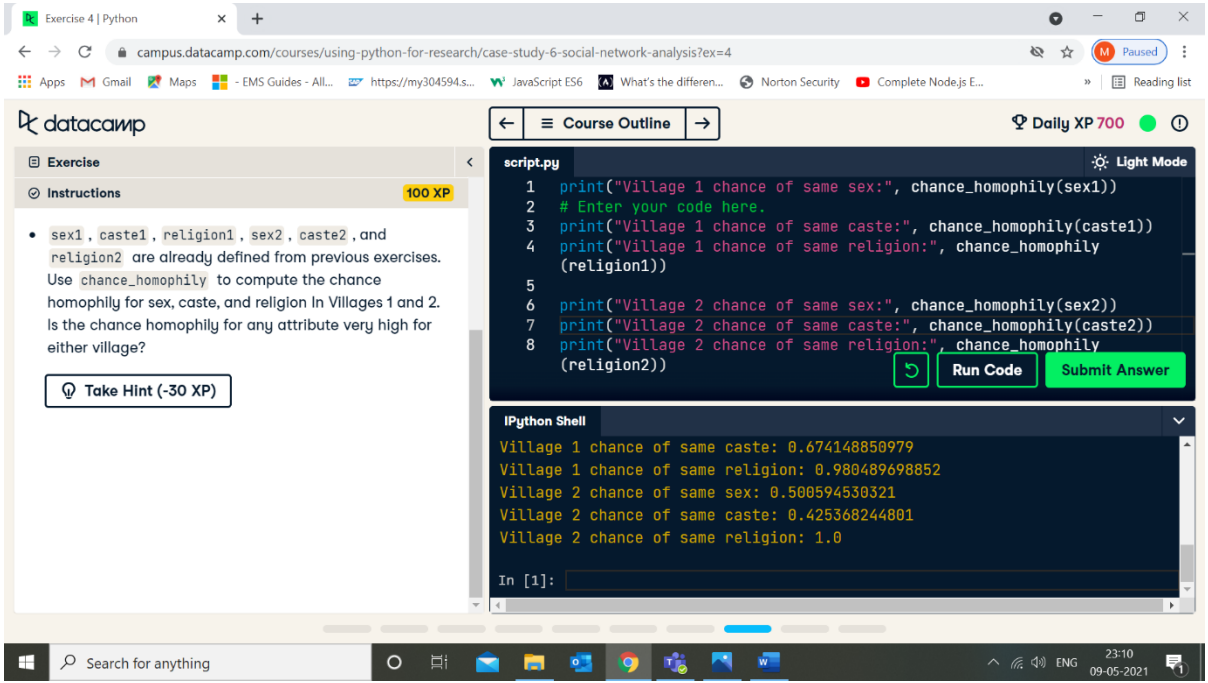
Instructions
100 XP

- `sex1`, `caste1`, `religion1`, `sex2`, `caste2`, and `religion2` are already defined from previous exercises.
  Use `chance_homophily` to compute the chance homophily for sex, caste, and religion In Villages 1 and 2. Is the chance homophily for any attribute very high for either village?

CODE :

```
print("Village 1 chance of same sex:", chance_homophily(sex1))
# Enter your code here.
print("Village 1 chance of same caste:", chance_homophily(caste1))
print("Village 1 chance of same religion:", chance_homophily(religion1))

print("Village 2 chance of same sex:", chance_homophily(sex2))
print("Village 2 chance of same caste:", chance_homophily(caste2))
print("Village 2 chance of same religion:", chance_homophily(religion2))
```

OUTPUT:


Exercise 4 | Python
campus.datacamp.com/courses/using-python-for-research/case-study-6-social-network-analysis?ex=4
Paused
Apps
Gmail
Maps
- EMS Guides - All...
https://my304594.s...
JavaScript ES6
What's the differen...
Norton Security
Complete Node.js E...
Reading list
datacamp
Course Outline
Daily XP 700
Exercise
Instructions
100 XP
sex1 , caste1 , religion1 , sex2 , caste2 , and religion2 are already defined from previous exercises. Use chance_homophily to compute the chance homophily for sex, caste, and religion In Villages 1 and 2. Is the chance homophily for any attribute very high for either village?
Take Hint (-30 XP)
script.py
Light Mode
1 print("Village 1 chance of same sex:", chance_homophily(sex1))
2 # Enter your code here.
3 print("Village 1 chance of same caste:", chance_homophily(caste1))
4 print("Village 1 chance of same religion:", chance_homophily (religion1))
5
6 print("Village 2 chance of same sex:", chance_homophily(sex2))
7 print("Village 2 chance of same caste:", chance_homophily(caste2))
8 print("Village 2 chance of same religion:", chance_homophily (religion2))
Run Code
Submit Answer
IPython Shell
Village 1 chance of same caste: 0.674148850979
Village 1 chance of same religion: 0.980489698852
Village 2 chance of same sex: 0.500594530321
Village 2 chance of same caste: 0.425368244801
Village 2 chance of same religion: 1.0
In [1]:
Search for anything
ENG
23:10
09-05-2021

Q5. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will create a function that computes the observed homophily given a village and characteristic.

Instructions
100 XP

- Complete the function `homophily()`, which takes a network `G`, a dictionary of characteristics `chars`, and node IDs `IDs`. For each node pair, determine whether a tie exists between them, as well as whether they share a characteristic. The total count of these is `num_same_ties` and `num_ties` respectively, and their ratio is the homophily of `chars` in `G`. Complete the function by choosing where to increment `num_same_ties` and `num_ties`.

CODE:

```
def homophily(G, chars, IDs):
    num_same_ties, num_ties = 0, 0
    for n1 in G.nodes():
        for n2 in G.nodes():
            if n1 > n2:   # do not double-count edges!
                if IDs[n1] in chars and IDs[n2] in chars:
```

```
                    if G.has_edge(n1, n2):
                        num_ties += 1# Should `num_ties` be incremented?  What about `num
_same_ties`?
                        if chars[IDs[n1]] == chars[IDs[n2]]:
                            num_same_ties += 1# Should `num_ties` be incremented?  What a
bout `num_same_ties`?
    return (num_same_ties / num_ties)
```

Q6. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will obtain the personal IDs for Villages 1 and 2. These will be used in the next exercise to calculate homophily for these villages.

Instructions
100 XP

- In this dataset, each individual has a personal ID, or PID, stored in `key_vilno_1.csv` and `key_vilno_2.csv` for villages 1 and 2, respectively. `data_filepath` contains the base URL to the datasets used in this exercise. Use `pd.read_csv` to read in and store `key_vilno_1.csv` and `key_vilno_2.csv` as `pid1` and `pid2` respectively. The `csv` files have no headers, so make sure to include the parameter `header = None`.

CODE:

```
pid1 = pd.read_csv(data_filepath + "key_vilno_1.csv", dtype=int, header = None)
pid2 = pd.read_csv(data_filepath + "key_vilno_2.csv", dtype=int, header = None)
```

Q7. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

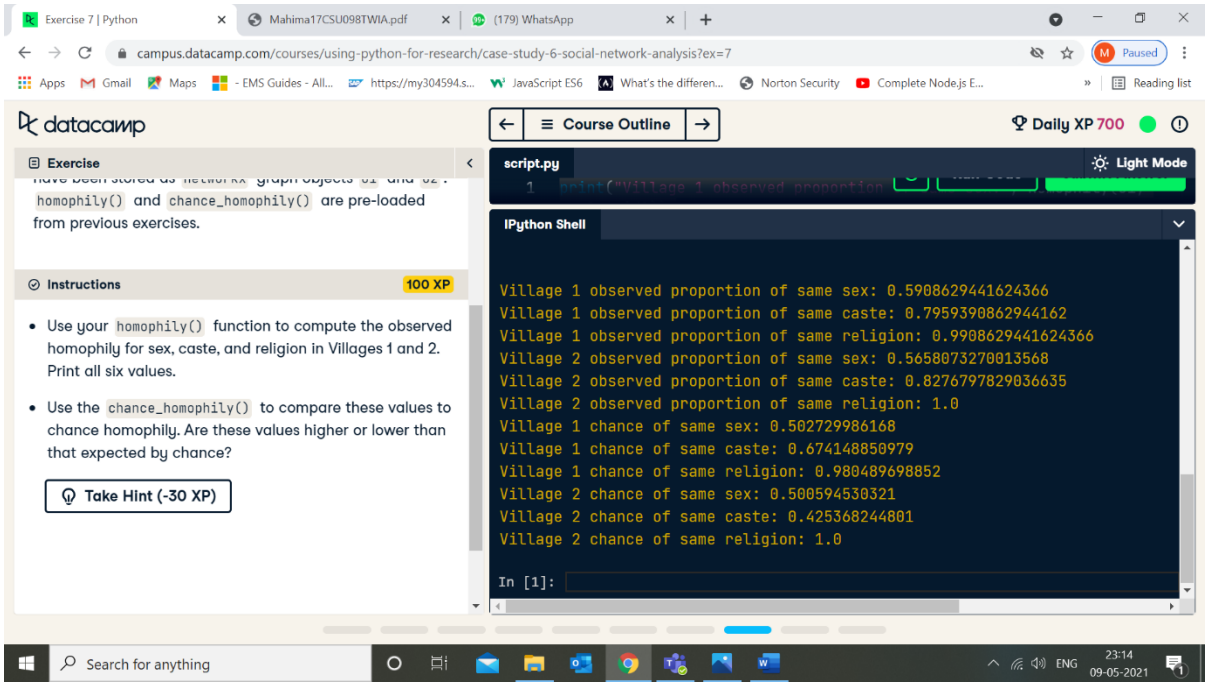
In this exercise, we will compute the homophily of several network characteristics for Villages 1 and 2, and compare this to chance homophily. The networks for these villages have been stored as `networkx` graph objects `G1` and `G2`. `homophily()` and `chance_homophily()` are pre-loaded from previous exercises.

CODE :

```
print("Village 1 observed proportion of same sex:", homophily(G1, sex1, pid1))
# Enter your code here!

print("Village 1 observed proportion of same caste:", homophily(G1, caste1, pid1))
print("Village 1 observed proportion of same religion:", homophily(G1, religion1, pid1))

print("Village 2 observed proportion of same sex:", homophily(G2, sex2, pid2))
print("Village 2 observed proportion of same caste:", homophily(G2, caste2, pid2))
print("Village 2 observed proportion of same religion:", homophily(G2, religion2, pid2))
```

```
print("Village 1 chance of same sex:", chance_homophily(sex1))

print("Village 1 chance of same caste:", chance_homophily(caste1))
print("Village 1 chance of same religion:", chance_homophily(religion1))

print("Village 2 chance of same sex:", chance_homophily(sex2))
print("Village 2 chance of same caste:", chance_homophily(caste2))
print("Village 2 chance of same religion:", chance_homophily(religion2))
```

OUTPUT:


Exercise 7 | Python
Mahima17CSU098TWIA.pdf
(179) WhatsApp
campus.datacamp.com/courses/using-python-for-research/case-study-6-social-network-analysis?ex=7
Paused
Apps
Gmail
Maps
- EMS Guides - All...
https://my304594.s...
JavaScript ES6
What's the differen...
Norton Security
Complete Node.js E...
Reading list
datacamp
Course Outline
Daily XP 700
Light Mode
Exercise
homophily() and chance_homophily() are pre-loaded from previous exercises.
Instructions
100 XP
Use your homophily() function to compute the observed homophily for sex, caste, and religion in Villages 1 and 2. Print all six values.
Use the chance_homophily() to compare these values to chance homophily. Are these values higher or lower than that expected by chance?
Take Hint (-30 XP)
script.py
IPython Shell
Village 1 observed proportion of same sex: 0.5908629441624366
Village 1 observed proportion of same caste: 0.7959390862944162
Village 1 observed proportion of same religion: 0.9908629441624366
Village 2 observed proportion of same sex: 0.5658073270013568
Village 2 observed proportion of same caste: 0.8276797829036635
Village 2 observed proportion of same religion: 1.0
Village 1 chance of same sex: 0.502729986168
Village 1 chance of same caste: 0.674148859979
Village 1 chance of same religion: 0.980489698852
Village 2 chance of same sex: 0.500594530321
Village 2 chance of same caste: 0.425368244801
Village 2 chance of same religion: 1.0
In [1]:
Search for anything
ENG
23:14
09-05-2021

VALUE ADDED
EXPERIMENT NO. 1

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_VA_17CSU098.ipynb
Date: 3 March 2021
Faculty Signature:
Grade:

Objective:

1. **Apply chunking on a sentence and explore text corpora.**
2. **Apply chinking on a piece of text.**
3. **Implementation of Named Entity recognition.**

Background Study:

Chunking

Chunking refers to the process of taking individual pieces of information and grouping them into larger units. By grouping each data point into a larger whole, we can improve the amount of information we can remember.

Chinking

Chinking is the process of removing a sequence of tokens from a chunk. If the matching sequence of tokens spans an entire chunk, then the whole chunk is removed; if the sequence of tokens appears in the middle of the chunk, these tokens are removed, leaving two chunks where there was only one before. If the sequence is at the periphery of the chunk, these tokens are removed, and a smaller chunk remains.

Named Entity

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on.

Named Entity Recognition

Named entity recognition (NER)is probably the first step towards information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

Outcome: Students will be able to understand the concept such as chunking, regex parser, chinking and tagging. They will also be able to explore the corpus brown and apply named entity recognition using ne_chunk() functions and Spacy library.

Problem Statement:

1. **Apply chunking on a sentence and explore text corpora.**

CODE AND OUTPUT:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[12] from pywsd.lesk import simple_lesk
```

```
[16] sentences = ['I went to the bank to deposit my money','The river bank had a lot of fishes and crocodiles.']

     #ambiguous word - Bank
```

Q1. (a) Chunking

```
[1]  import nltk
     from nltk.chunk import RegexpParser
     from nltk.tree import Tree
     from nltk import pos_tag
     nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
[2]  text ="My name is Mahima Munjal. I study at NCU.".split()
     print("\n\nSplitted text:",text)
     tokens_tag = pos_tag(text)
     print("\nAfter Token:",tokens_tag)
     patterns= """mychunk:{<NN.?>*<VBD.?>*<JJ.?>*<CC>?}"""
     chunker = RegexpParser(patterns)
     print("\nAfter Regex:",chunker)
     output = chunker.parse(tokens_tag)
     print("\nAfter Chunking",output)
```

```
After Token: [('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Mahima', 'NNP'), ('Munjal.', 'NNP'), ('I', 'PRP'), ('study', 'VBP'), ('at', 'IN'), ('NCU.', 'NNP')]

After Regex: chunk.RegexpParser with 1 stages:
RegexpChunkParser with 1 rules:
       <ChunkRule: '<NN.?>*<VBD.?>*<JJ.?>*<CC>?'>

After Chunking (S
  My/PRP$
  (mychunk name/NN)
  is/VBZ
  (mychunk Mahima/NNP Munjal./NNP)
  I/PRP
  study/VBP
  at/IN
  (mychunk NCU./NNP))
```

Q1. (b) Exploring Text Corpora

```
nltk.download('brown')
my_item = nltk.RegexpParser('CHUNK: {<V.*> <TO> <V.*>}')
shakespeare = nltk.corpus.brown
for sent in shakespeare.tagged_sents():
  tree = my_item.parse(sent)
  for subtree in tree.subtrees():
    if subtree.label() == 'CHUNK':
      print(subtree)
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
(CHUNK combined/VBN to/TO achieve/VB)
(CHUNK continue/VB to/TO place/VB)
(CHUNK serve/VB to/TO protect/VB)
(CHUNK wanted/VBD to/TO wait/VB)
(CHUNK allowed/VBN to/TO place/VB)
(CHUNK expected/VBN to/TO become/VB)
(CHUNK expected/VBN to/TO approve/VB)
(CHUNK expected/VBN to/TO make/VB)
(CHUNK intends/VBZ to/TO make/VB)
(CHUNK seek/VB to/TO set/VB)
(CHUNK like/VB to/TO see/VB)
(CHUNK designed/VBN to/TO provide/VB)
(CHUNK get/VB to/TO hear/VB)
(CHUNK expects/VBZ to/TO tell/VB)
(CHUNK expected/VBN to/TO give/VB)
(CHUNK prefer/VB to/TO pay/VB)
(CHUNK required/VBN to/TO obtain/VB)
(CHUNK permitted/VBN to/TO teach/VB)
(CHUNK designed/VBN to/TO reduce/VB)
(CHUNK Asked/VBN to/TO elaborate/VB)
(CHUNK got/VBN to/TO go/VB)
(CHUNK raised/VBN to/TO pay/VB)
```

2. Apply chinking on a piece of text.

CODE AND OUTPUT:

Q2. Chinking

```
grammar = r"""
  NP:
    {<.*>+}          # Chunk everything
    }<VBD|IN>+{      # Chink sequences of VBD and IN
  """
data = [("My", "PRP"), ("name", "NN"), ("is", "VBZ"),
       ("Mahima", "NNP"), ("Munjal", "NNP")]
my_chinked_pattern = nltk.RegexpParser(grammar)
print(my_chinked_pattern.parse(data))
```

```
(S (NP My/PRP name/NN is/VBZ Mahima/NNP Munjal/NNP))
```

3. Implementation of Named Entity recognition.

CODE AND OUTPUT:

Q3. Name Entity Recoginition

```
[5]  nltk.download('treebank')
     nltk.download('maxent_ne_chunker')
     nltk.download('words')
     nltk.download('punkt')
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

(a) Using ne_chunk( ) function

```
[6]  #Using ne_chunk() function

     from nltk.corpus import treebank_chunk
     from nltk.chunk import ne_chunk
     print(ne_chunk(treebank_chunk.tagged_sents()[0]))
```

```
(S
  (PERSON Pierre/NNP)
  (ORGANIZATION Vinken/NNP)
  ,/,
  61/CD
  years/NNS
  old/JJ
  ,/,
  will/MD
  join/VB
  the/DT
  board/NN
  as/IN
  a/DT
  nonexecutive/JJ
  director/NN
  Nov./NNP
  29/CD
  ./.)
```

(b) Using Spacy

```
[7]  #Using Spacy

     import spacy
     from spacy import displacy
     from collections import Counter
     import en_core_web_sm
     nlp = en_core_web_sm.load()
```

```
[8]  doc = nlp('Mahima Munjal wishes to invest in share of Google company worth a market value of $1 trillion.')
     print([(X.text, X.label_) for X in doc.ents])
```

```
     [('Mahima Munjal', 'PERSON'), ('Google', 'ORG'), ('$1 trillion', 'MONEY')]
```

```
[9]  doc = nlp('Mahima Munjal loves McDonald. She eats a burger every Friday at 4 pm.')
     print([(X.text, X.label_) for X in doc.ents])
```

```
     [('Mahima Munjal', 'PERSON'), ('McDonald', 'ORG'), ('every Friday', 'DATE'), ('4 pm', 'TIME')]
```

VALUE ADDED
EXPERIMENT NO. 2

Student Name and Roll Number: MAHIMA MUNJAL (17CSU098)
Semester /Section: VIII-B
Link to Code: https://github.com/munjalmahima/TWIA_LAB_EXPERIMENTS/blob/master/LAB_PRACTICAL_VA2_17CSU098.ipynb
Date: 22 January 2021
Faculty Signature:
Grade:

Objective:

1. **Apply word and sentence tokenization on a piece of text.**
2. **Implement Punkt Tokenization**
3. **Explore Gutenberg Corpus.**

Background Study:

Tokenization

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

Punkt Sentence Tokenizer

This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintexts in the target language before it can be used.

Gutenberg Corpus

The Project Gutenberg English corpus is a corpus made up of all English e-books available in the Gutenberg database in October 2014. Project Gutenberg Corpus, an open science approach to a curated version of the complete PG data containing more than 50,000 books and more than 3×109 word-tokens.

Outcome: Students will be able to understand the concept of tokenization and why it is an important aspect of NLTK and why it is needed. They will also be able to learn Punkt Tokenization. They will also be able to explore Gutenberg Corpus.

Problem Statement:

1. **Apply word and sentence tokenization on a piece of text.**

CODE AND OUTPUT:

```
[1] import nltk
```

```
[2] nltk.download('punkt')
```

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

WORD SENTENCE TOKENIZATION

```
[4] from nltk.tokenize import word_tokenize
    from nltk.tokenize import sent_tokenize
```

```
[5] text = "Mahima Munjal is a good girl. She studies in The Northcap University"
```

```
[6] word_tokenize(text)
```

```
['Mahima',
 'Munjal',
 'is',
 'a',
 'good',
 'girl',
 '.',
 'She',
 'studies',
 'in',
 'The',
 'Northcap',
 'University']
```

```
sent_tokenize(text)
```

```
['Mahima Munjal is a good girl.', 'She studies in The Northcap University']
```

2. Implement Punkt Tokenization

CODE AND OUTPUT:

PUNKT TOKENIZER

```
[21] import nltk.data
     text = '''Punkt knows that the periods in Ms. Mahima Munjal and Ashok K. Munjal
     do not mark sentence boundaries. And Sometimes sentences can start with
     non-capitalized words. i am a good girl.'''
     text
```

```
'Punkt knows that the periods in Ms. Mahima Munjal and Ashok K. Munjal \ndo not mark sentence boundaries. And Sometimes sentences can start with \nnon-capitalized words. i am a good girl.'
```

```
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
print('\n-----\n'.join(sent_detector.tokenize(text.strip())))
```

```
Punkt knows that the periods in Ms. Mahima Munjal and Ashok K. Munjal
do not mark sentence boundaries.
-----
And Sometimes sentences can start with
non-capitalized words.
-----
i am a good girl.
```

3. Explore Gutenberg Corpus.

CODE AND OUTPUT:

EXPLORING GUTENBERG CORPUS

```
[40] nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
True
```

```
from nltk.corpus import gutenberg
gutenberg.fileids()
```

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

[44]
```
#Raw text of gutenberg file austen-sense.txt

raw=gutenberg.raw('austen-sense.txt')
raw
```

```
'[Sense and Sensibility by Jane Austen 1811]\n\nCHAPTER 1\n\n\nThe family of Dashwood had long been settled in Sussex.\nTheir estate was large, a
centre of their property, where, for many generations,\nthey had lived in so respectable a manner as to engage\nthe general good opinion of their
his estate was a single man, who lived\nto a very advanced age, and who for many years of his life,\nhad a constant companion and housekeeper in
years before his own,\nproduced a great alteration in his home; for to supply\nher loss, he invited and received into his house the family\nof hi
r\nof the Norland estate, and the person to whom he intended\nto bequeath it.  In the society of his nephew and niece,\nand their children, the o
His attachment to them all increased.\nT...'
```

[43]
```
#No. of letters in austen-sense.txt
len(raw)
```

```
673022
```

[45]
```
#Sentences of gutenberg file austen-sense.txt

sentences=gutenberg.sents('austen-sense.txt')
sentences
```

```
[['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen', '1811', ']'], ['CHAPTER', '1'], ...]
```

[46]
```
#No. of sentences in austen-sense.txt

len(sentences)
```

```
4999
```

[47]
```
#Words of gutenberg file austen-sense.txt

words=gutenberg.words('austen-sense.txt')
words
```

```
['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', ...]
```

```
#No. of words in austen-sense.txt
len(words)
```

```
141576
```