
# TEXT AND WEB INTELLIGENCE ANALYTICS

## LAB MANUAL

| | |
|---|---|
| **SUBMITTED BY** | **Shruti Jain** |
| **ROLL NUMBER** | **17CSU186** |
| **CLASS** | **CSE-VIII-D** |
| **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 | 30-01-2021 | | |
| 2 | Stopwords | 06/02/2021 | | |
| 3 | Exploring Names Corpus | 13/02/2021 | | |
| 4 | Similarity metrices | 27/02/2021 | | |
| 5 | Lesk Algorithm for Word Sense Disambiguation | 06/03/2021 | | |
| 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 | Implementation of Fatman Evolutionary Network on jupyter notebook | 09/05/2021 | | |

**EXPERIMENT NO. 1**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 30 January 2021** |
| **Link to Code:** **https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/stopwords%20and%20synset%20implementation.ipynb** |
| **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:**

In [16]:
```
#Import a file and find the frequency of each word in a file.
import nltk
import os
import nltk.corpus
from nltk.probability import FreqDist
from nltk.corpus import brown
text=(brown.words(categories='news'))

```

In [22]:
```
#cleaning data of punctuations
import string
l=string.punctuation.split()
no_punct_dream=[words for words in text if words not in string.punctuation]
no_punct_dream
#Finding frequency
fdist=nltk.FreqDist(w.lower() for w in no_punct_dream)
fdist
```

Out[22]:
```
FreqDist({'the': 6386, 'of': 2861, 'and': 2186, 'to': 2144, 'a': 2130, 'in': 2020, 'for': 969, 'that': 829, 'is': 733, '``': 732, ...})
```

In [8]:
```
#Find the five most frequently occurring words from the document
fdist.most_common(5)
```

Out[8]:
```
[('the', 5580), (',', 5188), ('.', 4030), ('of', 2849), ('and', 2146)]
```

**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: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 6 February 2021** |
| **Link to Code:** **https://github.com/shruti17csu186/TWIA-LAB-IMENTS/blob/main/stopwords%20and%20synset%20implementation.ipynb** |
| **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:

```
In [10]:
#Print all the Arabic Stopwords.
from nltk.corpus import stopwords
stopwords=stopwords.words("Arabic")
print(stopwords)
```

```
['إذ', 'إذا', 'إذما', 'إذن', 'أف', 'أقل', 'أكثر', 'ألا', 'إلا', 'التي', 'الذي', 'الذين', 'اللاتي', 'اللائي', 'اللتان', 'اللتيا', 'اللتين', 'اللذان', 'اللذين', 'اللو
اتي', 'إلى', 'إليك', 'إليكم', 'إليكما', 'إليكن', 'أم', 'أما', 'أما', 'إما', 'أن', 'إن', 'إنا', 'أنا', 'أنت', 'أنتم', 'أنتما', 'أنتن', 'إنما', 'إنه', 'أنى', 'أد
ى', 'آه', 'آها', 'أو', 'أولاء', 'أولئك', 'أوه', 'آي', 'أي', 'أيها', 'إي', 'أين', 'أينما', 'إيه', 'بخ', 'بس', 'بعد', 'بعض', 'بك', 'بكم', 'بكم', 'بك
م', 'بكما', 'بكن', 'بل', 'بلى', 'بما', 'بماذا', 'بمن', 'بنا', 'به', 'بها', 'بهم', 'بهما', 'بهن', 'بي', 'بين', 'بيد', 'تلك', 'تلكم', 'تلكما', 'ته', 'تـ
ي', 'تين', 'تينك', 'ثم', 'ثمة', 'حاشا', 'حبذا', 'حتى', 'حيث', 'حيثما', 'حين', 'خلا', 'دون', 'ذا', 'ذات', 'ذاك', 'ذان', 'ذانك', 'ذلك', 'ذلكم', 'ذل
كما', 'ذلكن', 'ذه', 'ذو', 'ذوا', 'ذواتا', 'ذواتي', 'ذي', 'ذين', 'ذينك', 'ريث', 'سوف', 'سوى', 'شتان', 'عدا', 'عسى', 'عل', 'على', 'عليك', 'عليه
به', 'عما', 'عن', 'عند', 'غير', 'فإذا', 'فإن', 'فلا', 'فمن', 'في', 'فيم', 'فيما', 'فيه', 'فيها', 'قد', 'كأن', 'كأنما', 'كأي', 'كأين', 'كذا', 'كذلك',
'كل', 'كلا', 'كلاهما', 'كلتا', 'كلما', 'كليكما', 'كليهما', 'كم', 'كم', 'كما', 'كي', 'كيت', 'كيف', 'كيفما', 'لا', 'لاسيما', 'لدى', 'لست', 'لستم', 'لستم
ا', 'لستن', 'لسن', 'لسنا', 'لعل', 'لك', 'لكم', 'لكما', 'لكن', 'لكنما', 'لكي', 'لكيلا', 'لم', 'لما', 'لن', 'لنا', 'له', 'لها', 'لهم', 'لهما', 'لهن', 'ل
و', 'لولا', 'لوما', 'لي', 'لئن', 'ليت', 'ليس', 'ليسا', 'ليست', 'ليستا', 'ليسوا', 'ما', 'ماذا', 'متى', 'مذ', 'مع', 'مما', 'ممن', 'من', 'منه', 'منه
ا', 'مذ', 'مه', 'مهما', 'نحن', 'نحو', 'نعم', 'ها', 'هاتان', 'هاته', 'هاتي', 'هاتين', 'هاك', 'هاهنا', 'هذا', 'هذان', 'هذه', 'هذي', 'هذين', 'هكذا',
'هل', 'هلا', 'هم', 'هما', 'هن', 'هنا', 'هناك', 'هنالك', 'هو', 'هؤلاء', 'هي', 'هيا', 'هيت', 'هيهات', 'والذي', 'والذين', 'وإذ', 'وإذا', 'وإن', 'ول
يا', 'ولكن', 'ولو', 'وما', 'ومن', 'وهو']
```

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

### CODE AND OUTPUT:

```
In [13]:
from nltk.corpus import stopwords
stopwords=stopwords.words("english")
l=["not", "nor", "very", "most","shouldn't","couldn't"]
for i in l :
    if i in stopwords:
        stopwords.remove(i)
stopwords
```

```
"she's",
'her',
'hers',
'herself',
'it',
"it's",
'its',
'itself',
'they',
'them',
'their',
'theirs',
'themselves',
'what',
'which',
'who',
'whom',
'this',
'that',
```

**EXPERIMENT NO. 3**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 13 February 2021** |
| **Faculty Signature:** |
| **Link to Code:https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/stopwords%20and%20synset%20implementation.ipynb** |
| **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:
[('Averil', 'female'),
 ('Ashely', 'female'),
 ('Abbot', 'male'),
 ('Connor', 'male'),
 ('Kara', 'female'),
 ('Nissa', 'female'),
 ('Belia', 'female'),
 ('Bren', 'female'),
 ('Alison', 'female'),
 ('Hermia', 'female'),
 ('Vera', 'female'),
 ('Ralina', 'female'),
 ('Clari', '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: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 27 February 2021** |
| **Link to Code:** **https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/similarity%20matrices%20and%20tfidf%20implementation.ipynb** |
| **Faculty Signature:** |
| **Grade:** |

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

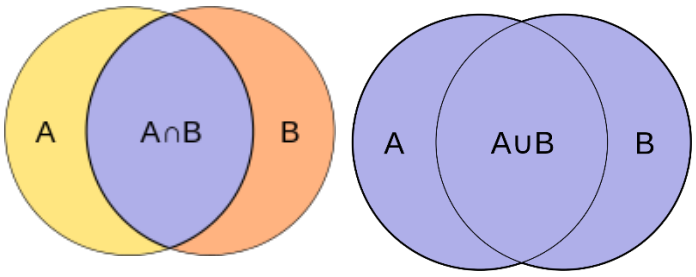
**Background Study:**

**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**

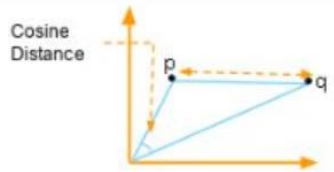
The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of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

$$d = \frac{\sum_{i=0}^{n-1} q_i - p_x}{\sum_{i=0}^{n-1}(q_i)^2 \times \sum_{i=0}^{n-1}(p_i)^2}$$




**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:

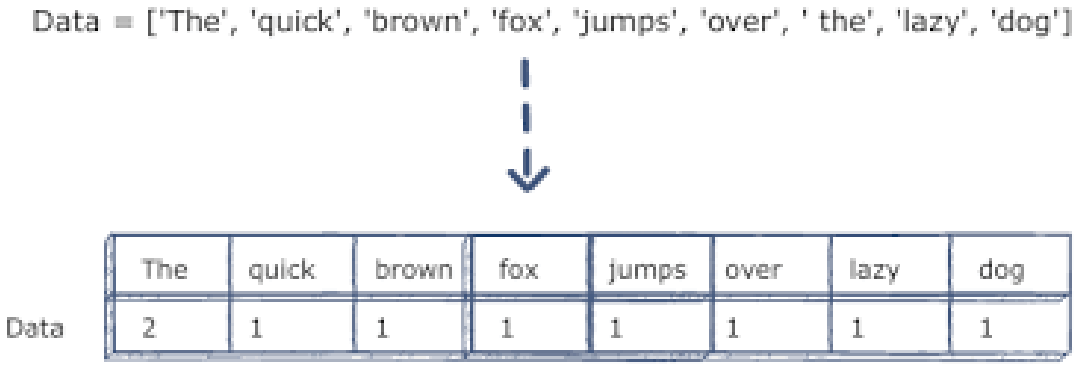
- The term frequency of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in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




**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 -

In [1]:
```
import nltk
from nltk.metrics import *
doc_1 = "Data is the new oil of the digital economy"
doc_2 = "Data is a new oil"
words_doc1=doc_1.lower().split()
words_doc1
words_doc2=doc_2.lower().split()
words_doc2
set_doc1=set(words_doc1)
set_doc2=set(words_doc2)
print(set_doc1)
intersection = set_doc1.intersection(set_doc2)
union = set_doc1.union(set_doc2)
jaccard=float(len(intersection)) / len(union)
jaccard
```

```
{'of', 'data', 'is', 'oil', 'digital', 'the', 'new', 'economy'}
```

Out[1]: 0.4444444444444444

## LEVENSHTEIN DISTANCE

In [8]:
```
#LEVENSHTEIN DISTANCE

import enchant
data1= "shruti jain is a good girl."
data2 = "shruti jain"
enchant.utils.levenshtein(data1,data2)
```

Out[8]: 16

## COSINE SIMILARITY

In [9]:
```
#COSINE SIMILARITY
import nltk
nltk.download ( ' punkt ' )
nltk.download('stopwords')

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


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

```

```
[nltk_data] Error loading  punkt : Package ' punkt ' not found in
[nltk_data]     index
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Shruti\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Enter first string: shruti is a good girl
Enter second string: shruti studies at ncu
```

In [11]:
```
# tokenization
I_list = word_tokenize(I)
II_list = word_tokenize(II)
print('First List', I_list)
print('Second Lzst', II_list)
```

```
First List ['shruti', 'is', 'a', 'good', 'girl']
Second Lzst ['shruti', 'studies', 'at', 'ncu']
```

In [14]:
```
#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  {'shruti', 'good', 'girl'}
second Set {'shruti', 'ncu', 'studies'}
```

In [24]:
```
# 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 SIMILARITY MATRIX

In [35]:
```
#Cosine similarity matrix
I = input("Enter first string: ").lower()
II = input("Enter second string: ").lower()
```

```
Enter first string: shruti jain is the student of text and web analytics
Enter second string: shruti jain is the student of northcap university
```

In [36]:
```
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
```

Out[36]:

| | analytics | and | is | jain | northcap | of | shruti | student | text | the | university | web |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| II | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

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

### CODE AND OUTPUT:

In [37]:
```
#2.Calculate the TF-IDF vectorizer on 2 documents.
#TF-IDF
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["Do not limit your challenges,challenge your limits",
        "your challenges",
        "their limits"]
vect = TfidfVectorizer()
X = vect.fit_transform(text)
print(vect.get_feature_names())
vect.idf_
```

```
['challenge', 'challenges', 'do', 'limit', 'limits', 'not', 'their', 'your']
```

Out[37]:
```
array([1.69314718, 1.28768207, 1.69314718, 1.69314718, 1.28768207,
       1.69314718, 1.69314718, 1.28768207])
```

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

### CODE AND OUTPUT:

In [42]:
```
#3.Apply the max-df, min-df param in the TF-IDF function.
data= [I,II]
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
count_vec = CountVectorizer(stop_words='english',analyzer="word",ngram_range=(1,1),max_df=0.50,min_df=1,max_features=Non
count_train= count_vec.fit(data)
bag_of_words=count_vec.transform(data)
print("features:",count_vec.get_feature_names())

```

```
features: ['analytics', 'northcap', 'text', 'university', 'web']
```

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

### AND OUTPUT:

In [50]:
```
#4.Compute Cosine Similarity using both TF-IDF and Count vectorizer
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
[nltk_data]     C:\Users\Shruti\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[50]:
```
True
```

In [51]:
```
train_dataset=["shruti is a good girl","shruti studies at northcap university"]#document
test_dataset=["a good girl named shruti studies at nothcap university"]#query
stopWords=stopwords.words("English")
```

In [56]:

```
vectorizer = CountVectorizer(stop_words = stopWords)
print('Vectorizer : ',vectorizer)
transformer = TfidfTransformer()
print('\n\nTF-IDF Transformer : ',transformer)
#Using Count Vectorizer
trainVectorizerArray = vectorizer.fit_transform(train_dataset).toarray()
testVectorizerArray =vectorizer.transform(test_dataset).toarray()
print ('Fit Vectorizer to train set \n', trainVectorizerArray)
print ('\n\nTransform Vectorizer to test set\n', testVectorizerArray)

```

```
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)
Fit Vectorizer to train set
```


```
TF-IDF Transformer :  TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=False, use_idf=True)
Fit Vectorizer to train set
```


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


Transform Vectorizer to test set
 [[1 1 0 1 1 1]]
```

In [57]:

```
 #Using
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
 [[0.6316672  0.6316672  0.         0.44943642 0.         0.        ]
 [0.         0.         0.53404633 0.37997836 0.53404633 0.53404633]]


Fit transformer to test set
 [[0.4472136 0.4472136 0.        0.4472136 0.4472136 0.4472136]]
```

**EXPERIMENT NO. 5**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 6 March 2021** |
| **Faculty Signature:** |
| **Link to Code:** **https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/Lesk%20Implementation.ipynb** |
| **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

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

**Problem Statement:** Implement Lesk algorithm for Word Sense Disambiguation

**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
```

## 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: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 1 April 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/LDA%20implementation%20and%20comparision.ipynb |
| **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.

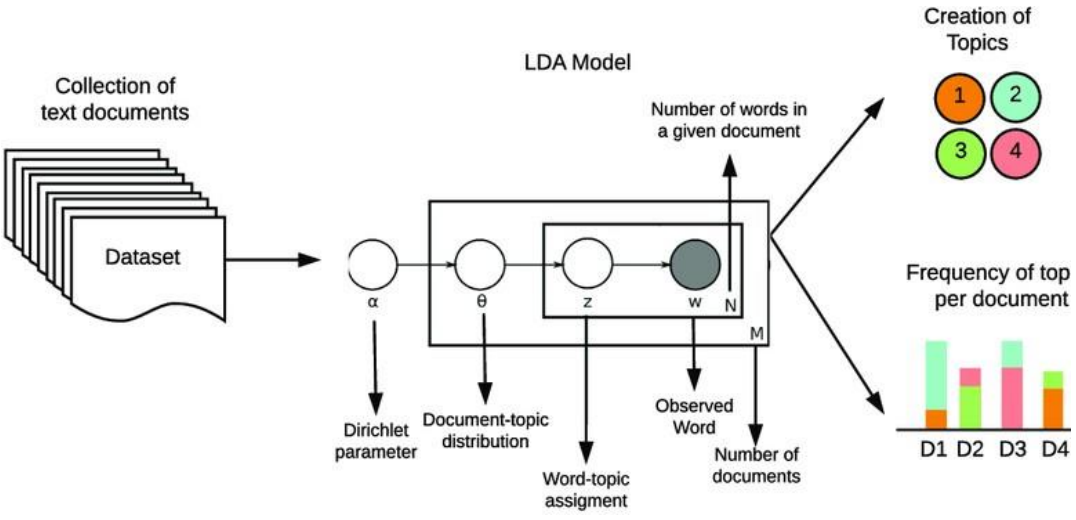
**Dataset:** https://www.kaggle.com/therohk/million-headlines

**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:

### Dataset

In [1]:
```
import pandas as pd

data = pd.read_csv('abcnews-date-text.csv', error_bad_lines=False);
data_text = data[['headline_text']]
data_text['index'] = data_text.index
documents = data_text
```

### Pre-Proccessing ¶

In [13]:
```
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import *
import numpy as np
np.random.seed(2018)
```

In [14]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Shruti\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[14]: True

In [15]:
```
# not stemming as it will not provide valid results
#Lemmatizing
#removing stopwords and words with len<3
def lemmatize_stemming(text):
    return 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
```

In [16]:
```
processed_docs = documents['headline_text'].map(preprocess)
processed_docs[:10]
```

Out[16]:
```
0              [decide, community, broadcast, licence]
1                          [witness, aware, defamation]
2          [call, infrastructure, protection, summit]
3                           [staff, aust, strike, rise]
4            [strike, affect, australian, travellers]
5              [ambitious, olsson, win, triple, jump]
6              [antic, delight, record, break, barca]
7    [aussie, qualifier, stosur, waste, memphis, ma...
8            [aust, address, security, council, iraq]
9                          [australia, lock, timetable]
Name: headline_text, dtype: object
```

In [22]:
```
dictionary = gensim.corpora.Dictionary(processed_docs)
```

In [49]:
```
dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
```

In [35]:
```
from gensim import corpora, models

tfidf = models.TfidfModel(bow_corpus)
corpus_tfidf = tfidf[bow_corpus]
from pprint import pprint
for doc in corpus_tfidf:
    pprint(doc)
    break
```

```
[(0, 0.5918674193999763),
 (1, 0.3937180767686992),
 (2, 0.5009876624450964),
 (3, 0.49365007440105513)]
```

In [36]:
```
# training lda model
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=10, id2word=dictionary, passes=2, workers=2)
```

In [40]:
```
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.057*"australia" + 0.041*"trump" + 0.024*"australian" + 0.022*"china" + 0.019*"world" + 0.019*"sydney" + 0.017*"ope
n" + 0.017*"coronavirus" + 0.015*"border" + 0.012*"win"
Topic: 1
Words: 0.023*"market" + 0.019*"year" + 0.016*"record" + 0.012*"care" + 0.012*"price" + 0.012*"years" + 0.012*"australian" +
0.011*"business" + 0.011*"country" + 0.010*"age"
Topic: 2
Words: 0.065*"coronavirus" + 0.032*"covid" + 0.029*"government" + 0.015*"rise" + 0.015*"restrictions" + 0.014*"water" + 0.01
2*"royal" + 0.012*"scott" + 0.011*"tasmanian" + 0.010*"commission"
Topic: 3
Words: 0.027*"kill" + 0.022*"die" + 0.019*"coast" + 0.018*"shoot" + 0.017*"miss" + 0.016*"crash" + 0.015*"attack" + 0.015*"g
old" + 0.015*"dead" + 0.014*"island"
Topic: 4
Words: 0.040*"police" + 0.026*"charge" + 0.026*"case" + 0.025*"court" + 0.020*"death" + 0.020*"murder" + 0.017*"face" + 0.01
3*"jail" + 0.013*"people" + 0.012*"arrest"
Topic: 5
Words: 0.027*"test" + 0.020*"tasmania" + 0.015*"morrison" + 0.014*"drum" + 0.012*"premier" + 0.011*"save" + 0.010*"race" +
0.009*"show" + 0.009*"park" + 0.009*"alan"
Topic: 6
Words: 0.050*"say" + 0.024*"victoria" + 0.022*"health" + 0.019*"change" + 0.019*"adelaide" + 0.014*"indigenous" + 0.011*"rur
al" + 0.011*"service" + 0.010*"national" + 0.010*"climate"
Topic: 7
Words: 0.030*"queensland" + 0.028*"election" + 0.022*"live" + 0.013*"federal" + 0.013*"work" + 0.012*"school" + 0.011*"counc
il" + 0.010*"farm" + 0.009*"labor" + 0.009*"fund"
```

In [42]:
```
for idx, topic in lda_model_tfidf.print_topics(-1):
    print('Topic: {} Word: {}'.format(idx, topic))
```

```
Topic: 0 Word: 0.007*"coronavirus" + 0.007*"border" + 0.007*"kill" + 0.006*"september" + 0.005*"footage" + 0.005*"china" +
0.005*"brexit" + 0.005*"protesters" + 0.005*"biden" + 0.005*"attack"
Topic: 1 Word: 0.013*"crash" + 0.011*"queensland" + 0.010*"bushfire" + 0.010*"coast" + 0.007*"weather" + 0.007*"die" + 0.007
*"michael" + 0.007*"police" + 0.006*"beach" + 0.006*"death"
Topic: 2 Word: 0.006*"plan" + 0.005*"coal" + 0.005*"water" + 0.005*"action" + 0.005*"government" + 0.005*"legal" + 0.005*"co
uncil" + 0.004*"spring" + 0.004*"closure" + 0.004*"coronavirus"
Topic: 3 Word: 0.021*"coronavirus" + 0.016*"covid" + 0.010*"health" + 0.009*"government" + 0.007*"royal" + 0.007*"commissio
n" + 0.007*"federal" + 0.006*"care" + 0.006*"update" + 0.006*"sport"
Topic: 4 Word: 0.012*"scott" + 0.011*"morrison" + 0.010*"restrictions" + 0.010*"coronavirus" + 0.009*"christmas" + 0.008*"tu
rnbull" + 0.007*"island" + 0.007*"stories" + 0.006*"morning" + 0.005*"malcolm"
Topic: 5 Word: 0.025*"news" + 0.016*"rural" + 0.010*"climate" + 0.010*"business" + 0.009*"thursday" + 0.009*"john" + 0.009
*"victorian" + 0.008*"david" + 0.008*"national" + 0.006*"farm"
Topic: 6 Word: 0.013*"market" + 0.009*"australian" + 0.009*"friday" + 0.008*"price" + 0.008*"share" + 0.007*"coronavirus" +
0.007*"rise" + 0.006*"grandstand" + 0.006*"finance" + 0.005*"festival"
Topic: 7 Word: 0.030*"trump" + 0.018*"donald" + 0.017*"interview" + 0.015*"country" + 0.010*"speak" + 0.010*"hour" + 0.007
*"australias" + 0.007*"cricket" + 0.007*"peter" + 0.007*"alan"
Topic: 8 Word: 0.014*"drum" + 0.011*"australia" + 0.010*"world" + 0.009*"tuesday" + 0.009*"league" + 0.008*"wednesday" + 0.0
08*"street" + 0.008*"wall" + 0.007*"final" + 0.006*"brief"
Topic: 9 Word: 0.018*"charge" + 0.016*"murder" + 0.016*"police" + 0.012*"court" + 0.011*"woman" + 0.010*"jail" + 0.010*"arre
st" + 0.009*"sentence" + 0.009*"assault" + 0.009*"guilty"
```

### Testing both the models

In [51]:
```
# Bag Of Words
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', lda_model.log_perplexity(bow_corpus))  # a measure of how good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=processed_docs, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -9.118709657723862

Coherence Score:  0.24535051024041796
```

In [54]:
```
# TFIDF
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', lda_model_tfidf.log_perplexity(bow_corpus))  # a measure of how good the model is. lower the bet

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model_tfidf, texts=processed_docs, dictionary=dictionary, coherence='c_v'
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -8.997436855159867

Coherence Score:  0.30983791499722557
```

In [ ]:
```
#since tdidf has more coherence score therefore it is more effective than bow
```

## Conclusion:

**Since Tf-Idf approach has more coherence score than Bag of Words approach therefore it more effective in text classification**

**EXPERIMENT NO. 7**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 8 April 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/knn%20na%C3%AFve%20Bayes(Gaussian%20and%20multinomial).ipynb |
| **Grade:** |

**Objectives:**

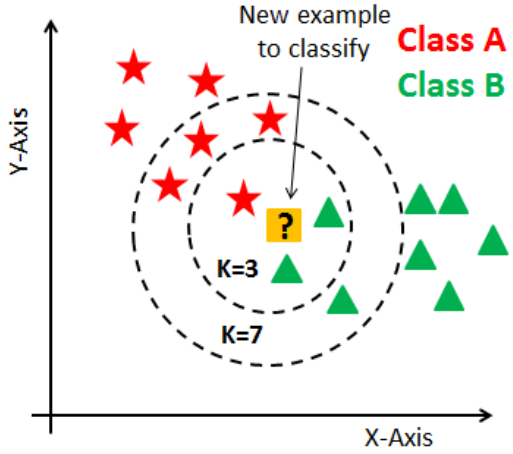
1. Perform KNN, Naïve Bayes and Multinomial Naïve Bayes on 20 Newsgroup Dataset.

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

**Background Study:**

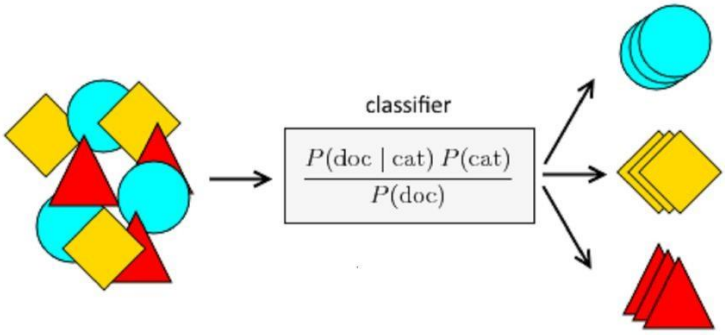
**K- Nearest Neighbors (KNN)**

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.



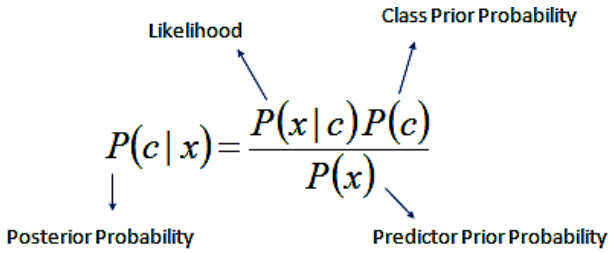

**Naïve Bayes**

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 dataset and compare their accuracy scores.

**CODE AND OUTPUT:**

## Importing libraries

In [1]:

```
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
import pandas as pd
from sklearn import metrics
import seaborn as sns; sns.set()
```

## importing Data

In [2]:
```
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=42)

test_data = fetch_20newsgroups(subset='test',
                               categories=categories, shuffle=True, random_state=42)
```

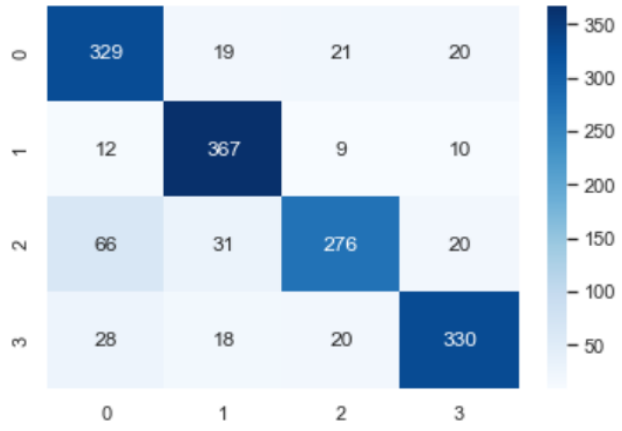
## KNN Classifier

In [3]:
```
knn = KNeighborsClassifier(n_neighbors=3)
#pipelinnig the fuctions
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', knn)
])
pipeline.fit(train_data.data, train_data.target)
predicted = pipeline.predict(test_data.data)
```

In [4]:
```
# accuray of the above
cm=metrics.confusion_matrix(test_data.target,predicted)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(test_data.target,predicted))
print(metrics.accuracy_score(test_data.target,predicted))
```

```
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       389
           1       0.84      0.92      0.88       398
           2       0.85      0.70      0.77       393
           3       0.87      0.83      0.85       396

    accuracy                           0.83      1576
   macro avg       0.83      0.83      0.82      1576
weighted avg       0.83      0.83      0.82      1576
```

```
0.8261421319796954
```



## MultiNomial Naive Bayes Classifier

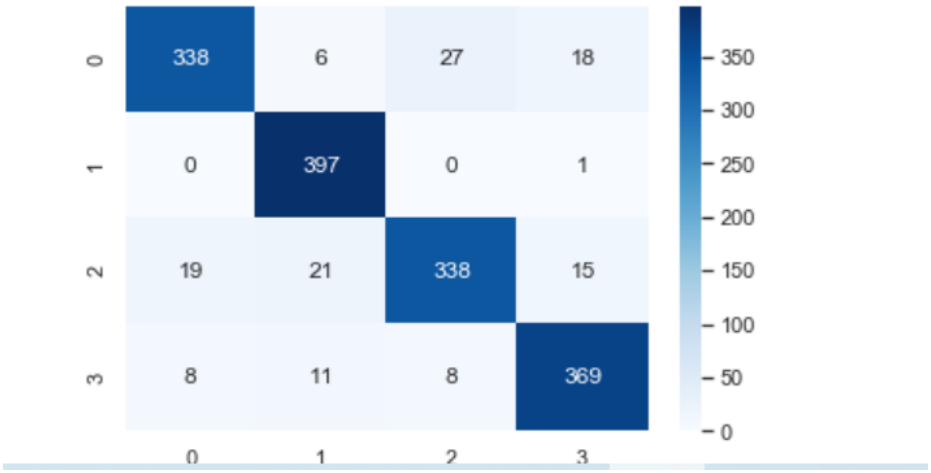
In [5]:

```
mnb = MultinomialNB()
#pipelinnig the fuctions
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', mnb)
])
pipeline.fit(train_data.data, train_data.target)
predicted = pipeline.predict(test_data.data)
```

In [6]:

```
# accuracy of the above
cm=metrics.confusion_matrix(test_data.target,predicted)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(test_data.target,predicted))
print(metrics.accuracy_score(test_data.target,predicted))
```

```
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       389
           1       0.91      1.00      0.95       398
           2       0.91      0.86      0.88       393
           3       0.92      0.93      0.92       396

    accuracy                           0.91      1576
   macro avg       0.92      0.91      0.91      1576
weighted avg       0.92      0.91      0.91      1576

0.9149746192893401
```

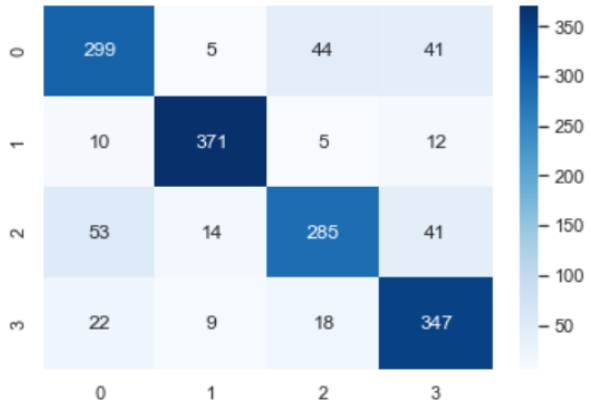


### Gaussian Naive bayes Classifier

In [7]:
```
gnb = GaussianNB()

#countvectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(train_data.data)

#tfidf vectorizer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf=X_train_tfidf.todense()

#fitting data into guassian NB classifier
clf = gnb.fit(X_train_tfidf, train_data.target)

#transforming the test data
X_new_counts = count_vect.transform(test_data.data)

X_new_tfidf = tfidf_transformer.transform(X_new_counts)
X_new_tfidf=X_new_tfidf.todense()

 # making prediction
predicted = gnb.predict(X_new_tfidf)

```

In [8]:
```
#accuracy of the above
cm=metrics.confusion_matrix(test_data.target,predicted)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(test_data.target,predicted))
print(metrics.accuracy_score(test_data.target,predicted))
```

```
              precision    recall  f1-score   support

           0       0.78      0.77      0.77       389
           1       0.93      0.93      0.93       398
           2       0.81      0.73      0.77       393
           3       0.79      0.88      0.83       396

    accuracy                           0.83      1576
   macro avg       0.83      0.83      0.82      1576
weighted avg       0.83      0.83      0.83      1576

0.8261421319796954
```




In [ ]:
```
# as we can see above the best result is given by Multinomial Naive bayes classifier
```

**Conclusion:**

**After seeing the accuracy of the above algos we see that multinomial naïve bayes has the highest accuracy of 92%.**

**EXPERIMENT NO. 8**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 2 May 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/k-means%20hierarchical%20and%20k-medoids%20clustering%20.ipynb |
| **Grade:** |

**Objectives:**

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

**Background Study:**

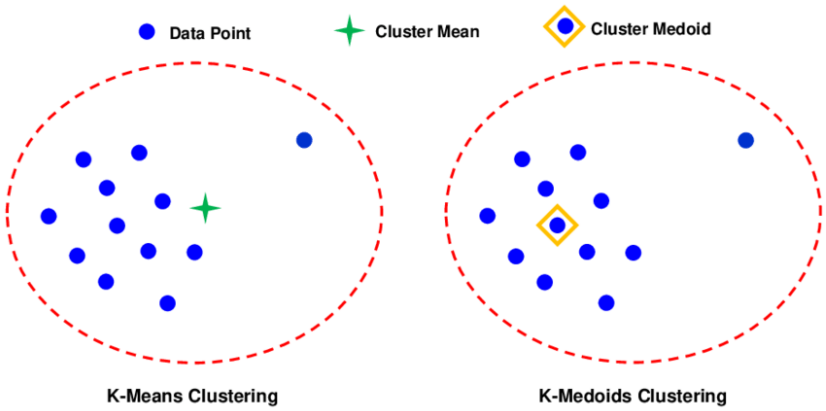
**K-Means**

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.

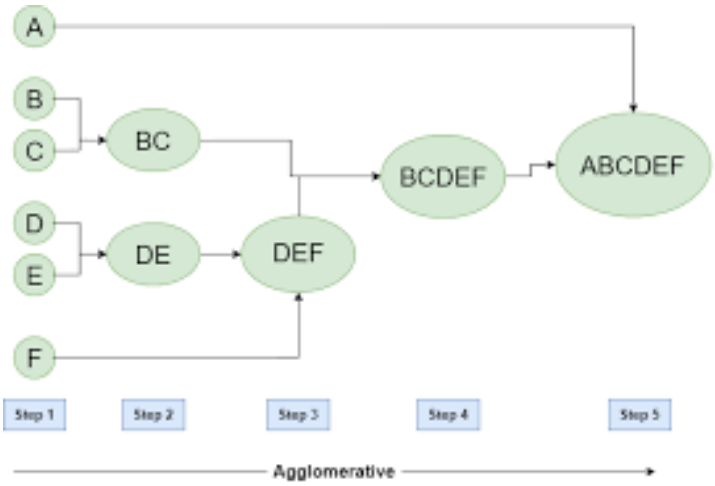
**K-Medoids**

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

In [2]:

```
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_extra.cluster import KMedoids
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.decomposition import TruncatedSVD
```

#### importing Data

In [3]:

```
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=42)

test_data = fetch_20newsgroups(subset='test',
                               categories=categories, shuffle=True, random_state=42)
```

## K-means Clustering

In [4]:

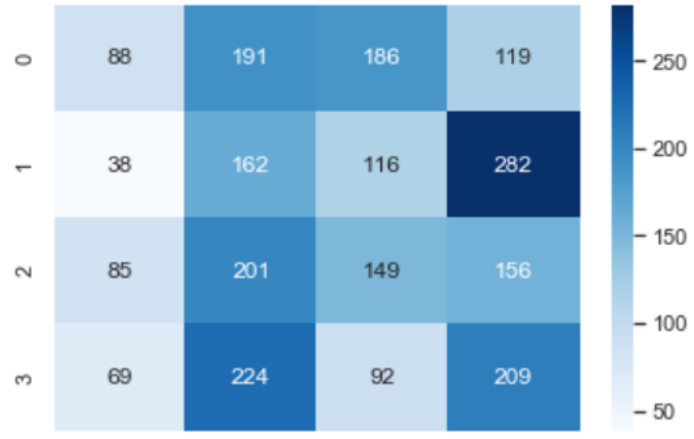
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_transformer = TfidfVectorizer(max_df=0.5, min_df=0.2, stop_words='english')
X_train_tfidf = tfidf_transformer.fit_transform(train_data.data)
km = KMeans(n_clusters=4, init='k-means++', max_iter=100, n_init=1)
km.fit(X_train_tfidf )
predicted=km.predict(X_train_tfidf)
print("Homogeneity: %0.3f"% metrics.homogeneity_score(train_data.target, predicted))
print("Completeness: %0.3f" % metrics.completeness_score(train_data.target, predicted))
print("V-measure: %0.3f" % metrics.v_measure_score(train_data.target, predicted))
print("Silhouette Coefficien: %0.3f"% metrics.silhouette_score(X_train_tfidf, predicted , sample_size=1000))

```

```
Homogeneity: 0.023
Completeness: 0.024
V-measure: 0.023
Silhouette Coefficien: 0.158
```

In [5]:

```
cm=metrics.confusion_matrix(train_data.target,predicted)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(train_data.target,predicted))
print(metrics.accuracy_score(train_data.target,predicted))
```

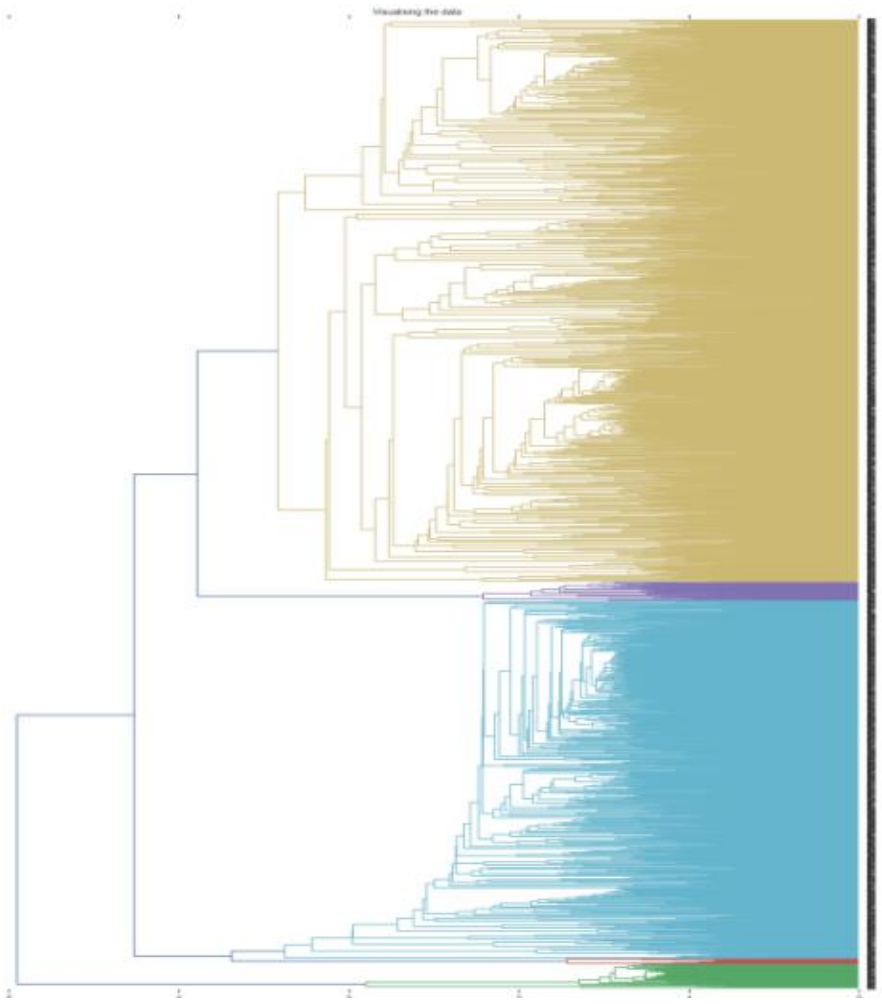
```
              precision    recall  f1-score   support

           0       0.31      0.15      0.20       584
           1       0.21      0.27      0.24       598
           2       0.27      0.25      0.26       591
           3       0.27      0.35      0.31       594

    accuracy                           0.26      2367
   macro avg       0.27      0.26      0.25      2367
weighted avg       0.27      0.26      0.25      2367

0.25686523024926067
```



## Hierachical Clustering

In [*]:

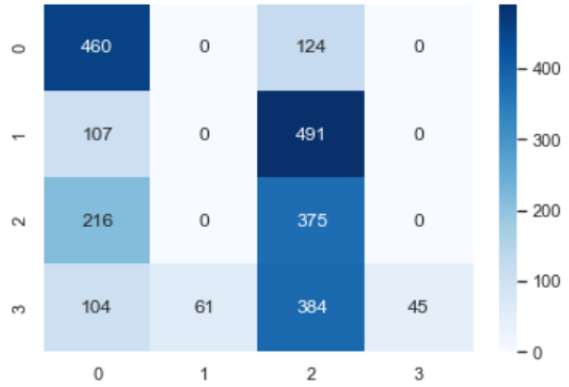
```
hc= AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
#countvectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(train_data.data)

#tfidf vectorizer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf=X_train_tfidf.todense()

#plotting of dendrograph
fig, ax = plt.subplots(figsize=(15, 20)) # set size
plt.title('Visualising the data')
ax= shc.dendrogram((shc.linkage(X_train_tfidf, method ='ward')),orientation="left")


plt.tick_params(\
    axis= 'x',
    which='both',
    bottom='off',
    top='off',
    labelbottom='off')

plt.tight_layout()

#fitting data into Heirarchical classifier
hc = hc.fit_predict(X_train_tfidf)


```



In [11]:

```
cm=metrics.confusion_matrix(train_data.target,hc)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(train_data.target,hc))
print(metrics.accuracy_score(train_data.target,hc))

```

```
              precision    recall  f1-score   support

           0       0.52      0.79      0.63       584
           1       0.00      0.00      0.00       598
           2       0.27      0.63      0.38       591
           3       1.00      0.08      0.14       594

    accuracy                           0.37      2367
   macro avg       0.45      0.37      0.29      2367
weighted avg       0.45      0.37      0.28      2367

0.3717786227291931
```




### K-medoids Clustering

In [6]:

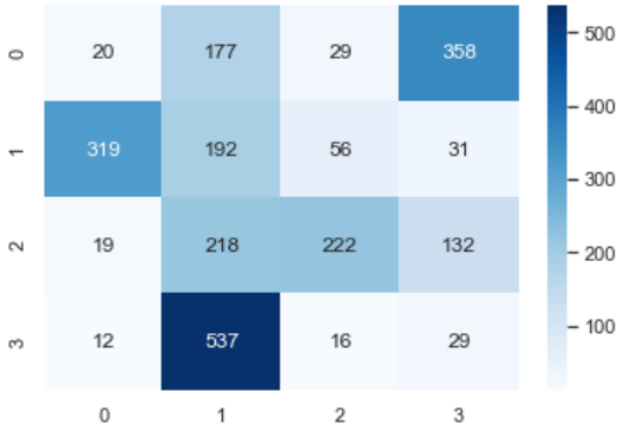
```
kMedoids = KMedoids(n_clusters = 4, random_state = 0)
#pipelinnig the fuctions
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', kMedoids)
])
pipeline.fit(train_data.data)
predicted = pipeline.predict(train_data.data)

```

In [7]:

```
cm=metrics.confusion_matrix(train_data.target,predicted)
ax = sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
print(metrics.classification_report(train_data.target,predicted))
print(metrics.accuracy_score(train_data.target,predicted))

```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.05 | 0.03 | 0.04 | 584 |
| 1 | 0.17 | 0.32 | 0.22 | 598 |
| 2 | 0.69 | 0.38 | 0.49 | 591 |
| 3 | 0.05 | 0.05 | 0.05 | 594 |
| accuracy | | | 0.20 | 2367 |
| macro avg | 0.24 | 0.19 | 0.20 | 2367 |
| weighted avg | 0.24 | 0.20 | 0.20 | 2367 |

```
0.19560625264047318
```




In [9]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_transformer = TfidfVectorizer(max_df=0.5, min_df=0.2, stop_words='english')
X_train_tfidf = tfidf_transformer.fit_transform(train_data.data)
kMedoids = KMedoids(n_clusters = 3, random_state = 0)
kMedoids.fit(X_train_tfidf)
predicted=km.predict(X_train_tfidf)
print("Homogeneity: %0.3f"% metrics.homogeneity_score(train_data.target, predicted))
print("Completeness: %0.3f" % metrics.completeness_score(train_data.target, predicted))
print("V-measure: %0.3f" % metrics.v_measure_score(train_data.target,predicted))
print("Silhouette Coefficien: %0.3f"% metrics.silhouette_score(X_train_tfidf, predicted , sample_size=1000))

```

```
Homogeneity: 0.023
Completeness: 0.024
V-measure: 0.023
Silhouette Coefficien: 0.151
```

**EXPERIMENT NO. 9**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 9 May 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/homophily.docx |
| **Grade:** |

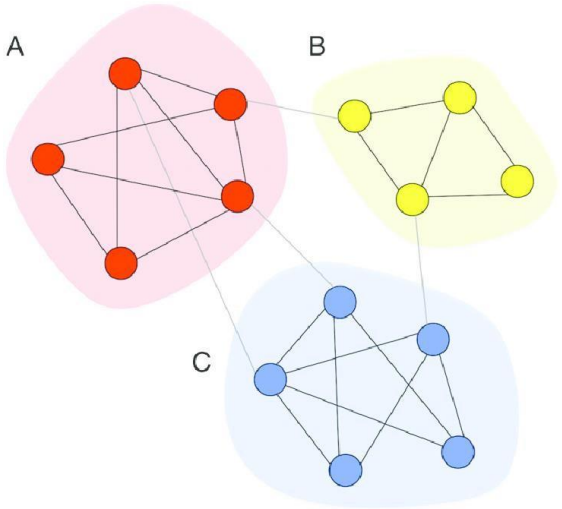
**Objectives:**

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 xand yshare characteristic ais 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 redand blueis 1/3 and 2/3 respectively, so the chance homophily is (1/3)^2+(2/3)^2 = 5/9.

**INSTRUCTIONS**

- Create a function that takes a dictionary charswith 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 charsdefined 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:**

script.py

```
from collections import Counter


def frequency(chars):
    # Enter code here!
    chars_counts_dict = Counter(chars.values())
    print(chars_counts_dict)
    return chars_counts_dict

def chance_homophily(chars):
    # Enter code here!
    chars_counts_dict=frequency(chars)
    chars_counts = np.array(list(chars_counts_dict.values()))
    print(chars_counts)
    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)
```

Light Mode | Run Code | Submit Answer | IPython Shell

**OUTPUT:**

IPython Shell

```
def chance_homophily(chars):
    # Enter code here!
    chars_counts_dict=frequency(chars)
    chars_counts = np.array(list(chars_counts_dict.values()))
    print(chars_counts)
    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)

Counter({'blue': 2, 'red': 1})
[2 1]
0.555555555556
```

**VALUE ADDED**

**EXPERIMENT NO. 1**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 6 March 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/chunking%20and%20chinkin.ipynb |
| **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

**Problem Statement:**

**CODE AND OUTPUT:**

**Chunking**

In [2]:
```
#Chunking
# Loading Libraries
from nltk.chunk.regexp import ChunkString, ChunkRule
from nltk.tree import Tree

# ChunkString() starts with the flat tree
tree = Tree('S', [('the', 'DT'), ('book', 'NN'),
              ('has', 'VBZ'), ('many', 'JJ'), ('chapters', 'NNS')])
tree.draw()
chunk_string = ChunkString(tree)
print ("Chunk String : ", chunk_string)
```

```
Chunk String :  <DT> <NN> <VBZ> <JJ> <NNS>
```

In [3]:
```
grammar = r"""
  NP: {<DT|JJ|NN.*>+}          # Chunk sequences of DT, JJ, NN
  PP: {<IN><NP>}               # Chunk prepositions followed by NP
  VP: {<VB.*>} # Chunk verbs and their arguments

  """
```

In [5]:
```
import nltk
chunker = nltk.RegexpParser(grammar)
sent=chunker.parse(tree)
sent.draw()
```

In [6]:
```
chunker1 = nltk.RegexpParser(grammar)
sent = [('the', 'DT'), ('sushi', 'NN'), ('roll', 'NN'), ('was', 'VBD'),
        ('filled', 'VBN'), ('with', 'IN'), ('the', 'DT'), ('fish', 'NN')]
result2=chunker.parse(sent)
result2.draw()
```

In [10]:
```
#Name Entity Recognition using NLTK
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')

my_sent = "WASHINGTON -- In the wake of a string of abuses by New York police officers in the 1990s, Loretta E. Lynch, t

parse_tree = nltk.ne_chunk(nltk.tag.pos_tag(nltk.word_tokenize(my_sent)), binary=True)

parse_tree.draw()
print(parse_tree)

```

In [11]:

```
from nltk.tree import Tree
nltk.download('maxent_ne_chunker')
txt="WASHINGTON -- In the wake of a string of abuses by New York police officers in the 1990s, Loretta E. Lynch, the top

pos_tag = nltk.pos_tag(txt.split())
parse_tree = nltk.ne_chunk(pos_tag )
# print(chunk)
parse_tree.draw()
NE=[]
for chunk in parse_tree:
    if hasattr(chunk, 'label'):
        NE=(chunk.label(), ' '.join(c[0] for c in chunk))
        print(NE)

```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\Shruti\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
```

```
('GPE', 'WASHINGTON')
('GPE', 'New York')
('PERSON', 'Loretta E.')
```

**VALUE ADDED**

**EXPERIMENT NO. 2**

| |
|---|
| **Student Name and Roll Number: SHRUTI JAIN (17CSU186)** |
| **Semester /Section: VIII-D** |
| **Date: 9 May 2021** |
| **Faculty Signature:** |
| **Link to Code:** https://github.com/shruti17csu186/TWIA-LAB-EXPERIMENTS/blob/main/FATMAN%20EVOLUTIONARY%20NETWORK%20IMPLEMENTATION.ipynb |
| **Grade:** |

**Objective:**

To implement Fatman Evolutionary Model in Social Networks on python jupyter notebook.

**Background Study:**

In the Fatman evolutionary model, there are 3 main concepts to make an evolutionary model i.e Homophily, Social influence, and closure.

1. **Homophily-** People who are similar to each other, they tend to make friends with each other.
2. **Social Influence-** People change their behavior and properties because of social influence.
3. **Closure-** There are 3 main kinds of closures i.e triadic closure, membership closure, and foci closure.

- **Triadic closure-** In triadic closure, if A is a friend of B and B is a friend of C then C will eventually become friends of A.
- **Membership closure-** In membership closure, if A and B are in the same club then there is a tendency that they will become friends.
- **Foci closure-** It is the probability of becoming friends when they have the same foci.

So, According to Fatman Hypothesis, beware of fat friends, if your friend is fat then the probability of you gaining the weight becomes high.

**Outcome:**

Students will be able to understand the concept of Fatman evolutionary model and able to implement it on python.

**Problem Statement:**

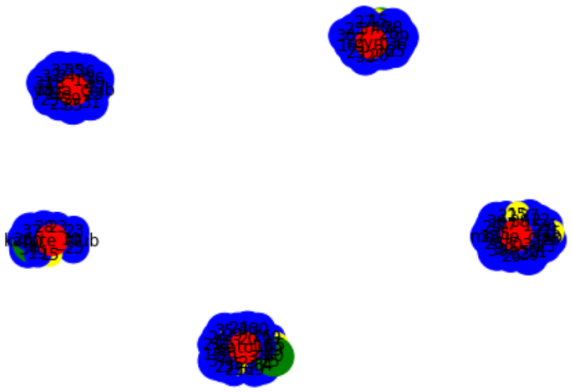
Assume there is a city with several people, and we have everyone's BMI. Then we will see as evolution continues people having similar BMI become friends to each other which is Homophily. Then we will create some foci like Gym etc. and see how eventually people with similar foci become a friend and then we will implement social foci

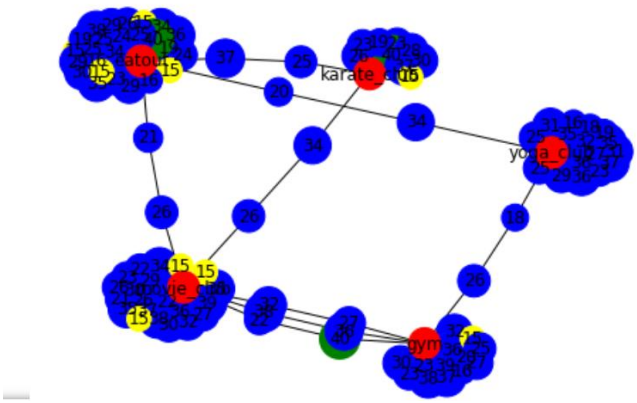
**CODE AND OUTPUT:**

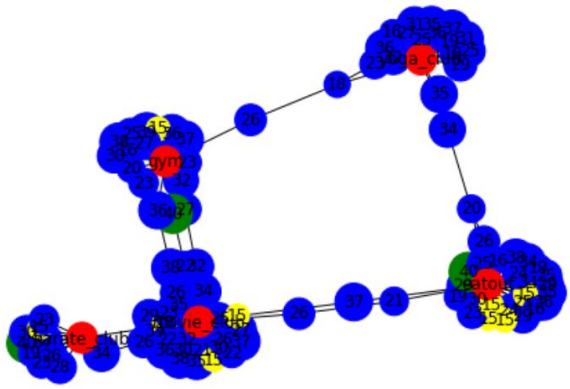
In [2]:

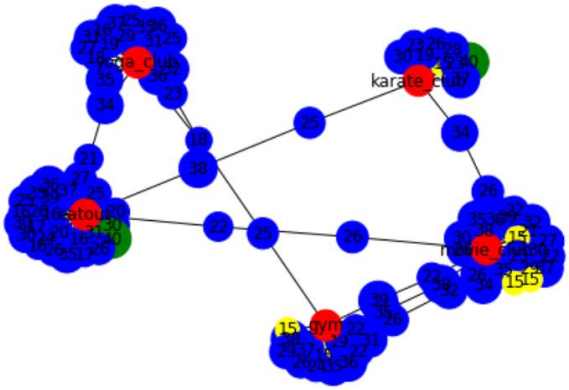
```
import networkx as nx
import matplotlib.pyplot as plt
import random
import math

# 1- Create a graph with lets say 100 nodes
# 2- Add egdes and labels.
def create():
    G = nx.Graph()
    G.add_nodes_from(range(1, 101))
    return G

def visualize(G):
    labeldict = get_labels(G)
    nodesize = get_size(G)
    color = get_colors(G)
    nx.draw(G, labels=labeldict, node_size=nodesize, node_color=color)
    plt.show()


def assign_bmi(G):

    for each in G.nodes():
        G.nodes[each]['name'] = random.randint(15, 40)
        G.nodes[each]['type'] = 'person'


def get_labels(G):
    dict1 = {}

    for each in G.nodes():
        dict1[each] = G.nodes[each]['name']

    return dict1

def get_size(G):
    array1 = []
    for each in G.nodes():
        if (G.nodes[each]['type'] == 'person'):
            array1.append(G.nodes[each]['name'] * 20)
        else:
            array1.append(500)
    return array1

# 3. Add foci nodes
def add_foci_nodes(G):
    n = G.number_of_nodes()
    i = n + 1
    foci_nodes = ['gym', 'eatout', 'movie_club',
                  'karate_club', 'yoga_club']

    for j in range(5):
        G.add_node(i)
        G.nodes[i]['name'] = foci_nodes[j]
        G.nodes[i]['type'] = 'foci'
        i += 1

```

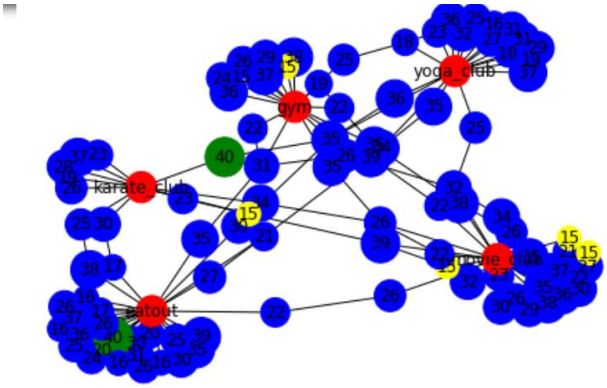
```
def get_colors(G):
    c = []

    for i in G.nodes():
        if (G.nodes[i]['type'] == 'person'):
            if (G.nodes[i]['name'] == 15):
                c.append('yellow')
            elif (G.nodes[i]['name'] == 40):
                c.append('green')
            else:
                c.append('blue')
        else:
            c.append('red')
    return c

def get_person_nodes(G):
    p = []

    for i in G.nodes():
        if (G.nodes[i]['type'] == 'person'):
            p.append(i)
    return p

def get_foci_nodes(G):
    f = []

    for i in G.nodes():
        if (G.nodes[i]['type'] == 'foci'):
            f.append(i)
    return f

def add_foci_edges(G):
    foci_nodes = get_foci_nodes(G)
    person_nodes = get_person_nodes(G)

    for i in person_nodes:
        r = random.choice(foci_nodes)
        G.add_edge(i, r)

def homophily(G):
    pnodes = get_person_nodes(G)

    for u in pnodes:
        for v in pnodes:
            if (u != v):
                diff = abs(G.nodes[u]['name'] - G.nodes[v]['name'])
                p = 1 / (diff + 1000)
                r = random.uniform(0, 1)
                if (r < p):
                    G.add_edge(u, v)


def common(u, v, G):
    nu = set(G.neighbors(u))
    nv = set(G.neighbors(v))
    return (len(nu & nv))

def closure(G):
    array1 = []
    for u in G.nodes():
        for v in G.nodes():
            if (u != v and G.nodes[u]['type'] == 'person' or G.nodes[v]['type'] == 'person'):

                # common function will return start node,
                # end node and Graph itself.
                k = common(u, v, G)
                p = 1 - math.pow(1 - 0.01, k)
                tmp = []
                tmp.append(u)
                tmp.append(v)
                tmp.append(p)
                array1.append(tmp)
    for i in array1:
        u = i[0]
        v = i[1]
        p = i[2]
        r = random.uniform(0, 1)
        if r < p:
            G.add_edge(u, v)
def social_influence(G):
    fnodes = get_foci_nodes(G)

    for i in fnodes:
        if (G.nodes[i]['name'] == 'eatout'):
            for j in G.neighbors(i):
                if (G.nodes[j]['name'] != 40):
                    G.nodes[j]['name'] = G.nodes[j]['name'] + 1

        if (G.nodes[i]['name'] == 'gym'):
            for j in G.neighbors(i):
                if (G.nodes[j]['name'] != 15):
                    G.nodes[j]['name'] = G.nodes[j]['name'] - 1


G = create()
assign_bmi(G)

# to add foci field in the nodes
add_foci_nodes(G)
add_foci_edges(G)
visualize(G)

for t in range(1, 3):
    homophily(G)
    visualize(G)
    closure(G)
    visualize(G)
    social_influence(G)
    visualize(G)
```

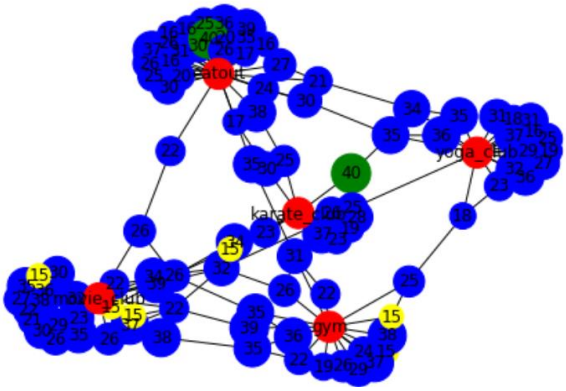


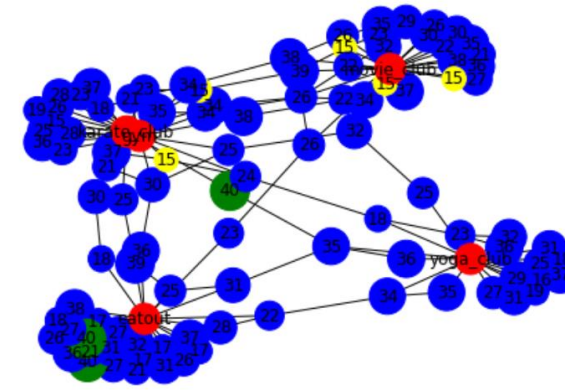

yoga_club
gym


yoga_club
gym


yoga_club
gym


yoga_club
gym
karate_club


yoga_club
gym
karate_club


yoga_club
gym