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		
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:

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

```python
nltk.download('shakespeare')
```

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

```python
shakespeare.fileids()
```

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

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

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

```python
len(dream)
```

```
21538
```

```python
#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.

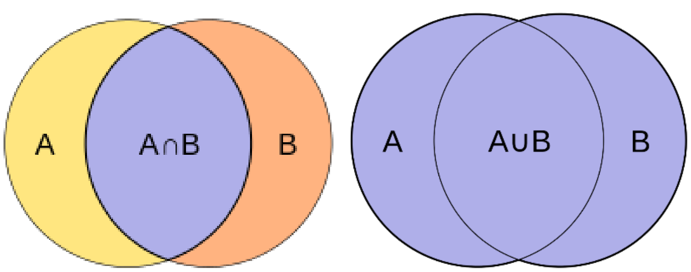
Background Study:

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.

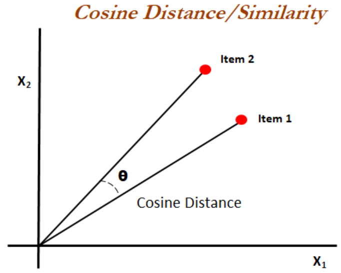
Jaccard similarity

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
```

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

```python
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
```

```python
# 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']
```

```python
#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'}
```

```python
# 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
```

```python
# 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:

```
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
```

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)
```

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 thei
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
```