Assignment no 7

Aim:

1. Basic concepts of Text Analytics
2. Text Analysis Operations using natural language

toolkit 3. Text Analysis Model using TF-IDF. 4. Bag of Words (BoW)

In [ ]:
```
Name:Sneha Navgire
Rollno:13246
Batch:B3
```

In [ ]:
```
#Part 1: Text Preprocessing (Lemmatization, Tokenization, POS Tagging, Stopword Rem
```

In [5]:
```
#Step 1: Download Required Packages
import nltk
nltk.download('punkt')  # For tokenization
nltk.download('stopwords')  # For stopwords
nltk.download('wordnet')  # For lemmatization
nltk.download('averaged_perceptron_tagger')  # For POS tagging
```

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

Out[5]: True

In [6]:
```
#Step 2: Initialize Text
text = "Tokenization is the first step in text analytics.The process of breaking do
```

In [7]:
```
#Step 3: Perform Tokenization
#1.Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text = sent_tokenize(text)
print(tokenized_text)
```

```
['Tokenization is the first step in text analytics.The process of breaking down a te
xt paragraph into smaller chunks such as words or sentences is called Tokenizatio
n.']
```

In [8]:
```
#2.Word Tokenization
from nltk.tokenize import word_tokenize
tokenized_word = word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics.The', 'proce
ss', 'of', 'breaking', 'down', 'a', 'text', 'paragraph', 'into', 'smaller', 'chunk
s', 'such', 'as', 'words', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

In [9]:
```
#Step 4: Removing Punctuation and Stop Words
#Remove Stop Words
from nltk.corpus import stopwords
import re

stop_words = set(stopwords.words("english"))
text = "How to remove stop words with NLTK library in Python?"
text = re.sub('[^a-zA-Z]', ' ', text)  # Remove punctuation
tokens = word_tokenize(text.lower())
filtered_text = [w for w in tokens if w not in stop_words]

print("Tokenized Sentence:", tokens)
print("Filtered Sentence:", filtered_text)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'librar
y', 'in', 'python']
Filtered Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [10]:
```
#Step 5: Perform Stemming
from nltk.stem import PorterStemmer

e_words = ["wait", "waiting", "waited", "waits"]
ps = PorterStemmer()

for w in e_words:
    rootWord = ps.stem(w)
    print(rootWord)
```

```
wait
wait
wait
wait
```

In [11]:
```
#Step 6: Perform Lemmatization
from nltk.stem import WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)

for w in tokenization:
    print("Lemma for {}: {}".format(w, wordnet_lemmatizer.lemmatize(w)))
```

```
Lemma for studies: study
Lemma for studying: studying
Lemma for cries: cry
Lemma for cry: cry
```

In [12]:
```
#Step 7: Apply POS Tagging to Text
from nltk.tokenize import word_tokenize
data = "The pink sweater fit her perfectly"
words = word_tokenize(data)

for word in words:
    print(nltk.pos_tag([word]))
```

```
[('The', 'DT')]
[('pink', 'NN')]
[('sweater', 'NN')]
[('fit', 'NN')]
[('her', 'PRP$')]
[('perfectly', 'RB')]
```

In [ ]:
```
#Part 2: TF-IDF Representation of Documents
```

In [13]:
```
#Step 1: Import Required Libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

In [14]:
```
#Step 2: Initialize the Documents
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [16]:
```
#Step 3: Create Bag of Words (BoW) for Document A and B
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [17]:
```
bagOfWordsA
```

Out[17]:
```
['Jupiter', 'is', 'the', 'largest', 'Planet']
```

In [18]:
```
bagOfWordsB
```

Out[18]:
```
['Mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'Sun']
```

In [20]:
```
#Step 4: Create Collection of Unique Words from Document A and B
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
uniqueWords
```

Out[20]:
```
{'Jupiter',
 'Mars',
 'Planet',
 'Sun',
 'fourth',
 'from',
 'is',
 'largest',
 'planet',
 'the'}
```

In [21]:
```
#Step 5: Create a Dictionary of Words and Their Occurrence for Each Document
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1

numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

In [22]:
```
numOfWordsA
```

Out[22]:
```
{'Sun': 0,
 'Jupiter': 1,
 'the': 1,
 'fourth': 0,
 'Planet': 1,
 'from': 0,
 'largest': 1,
 'Mars': 0,
 'is': 1,
 'planet': 0}
```

In [23]:
```
numOfWordsB
```

Out[23]:
```
{'Sun': 1,
 'Jupiter': 0,
 'the': 2,
 'fourth': 1,
 'Planet': 0,
 'from': 1,
 'largest': 0,
 'Mars': 1,
 'is': 1,
 'planet': 1}
```

In [24]:
```
#Step 6: Compute Term Frequency (TF)
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict

tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [25]:
```
tfA
```

Out[25]:
```
{'Sun': 0.0,
 'Jupiter': 0.2,
 'the': 0.2,
 'fourth': 0.0,
 'Planet': 0.2,
 'from': 0.0,
 'largest': 0.2,
 'Mars': 0.0,
 'is': 0.2,
 'planet': 0.0}
```

In [26]:
```
tfB
```

Out[26]:
```
{'Sun': 0.125,
 'Jupiter': 0.0,
 'the': 0.25,
 'fourth': 0.125,
 'Planet': 0.0,
 'from': 0.125,
 'largest': 0.0,
 'Mars': 0.125,
 'is': 0.125,
 'planet': 0.125}
```

In [27]:
```
#Step 7: Compute Inverse Document Frequency (IDF)
def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1
    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict

idfs = computeIDF([numOfWordsA, numOfWordsB])
```

In [28]:
```
idfs
```

Out[28]:
```
{'Sun': 0.6931471805599453,
 'Jupiter': 0.6931471805599453,
 'the': 0.0,
 'fourth': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'from': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'Mars': 0.6931471805599453,
 'is': 0.0,
 'planet': 0.6931471805599453}
```

In [29]:
```
#Step 8: Compute TF-IDF
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
```

```
    return tfidf

tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)

# Create a DataFrame for visualization
df = pd.DataFrame([tfidfA, tfidfB])
print(df)
```

```
        Sun   Jupiter  the    fourth    Planet      from   largest      Mars  \
0  0.000000  0.138629  0.0  0.000000  0.138629  0.000000  0.138629  0.000000
1  0.086643  0.000000  0.0  0.086643  0.000000  0.086643  0.000000  0.086643

    is    planet
0  0.0  0.000000
1  0.0  0.086643
```