ASSIGNMENT 7

Text Analytics I

1.Extract sample document and apply following document preprocessing methods: Tokenisation, POS Tagging, stop words removal, Stemming and Lemmatization

2.Create representation of document by calculating Term Frequency and Inverse Document Frequency

> Downloading required packages

In [1]:
```
import nltk
```

In [2]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

Out[2]:
```
True
```

> Initializing the Text

In [3]:
```
text = "Tokenization is the first step in text analytics. The process of b
text
```

Out[3]:
```
'Tokenization is the first step in text analytics. The process of breaki
ng down a text paragraph into smaller chunks such word or sentence is ca
lled Tokenization'
```

In [4]:
```
from nltk.tokenize import sent_tokenize
sent_token = sent_tokenize(text)
```

In [5]:
```
sent_token
```

Out[5]:
```
['Tokenization is the first step in text analytics.',
 'The process of breaking down a text paragraph into smaller chunks such
word or sentence is called Tokenization']
```

In [6]:
```
from nltk.tokenize import word_tokenize
word_token = word_tokenize(text)
```

In [7]:
```
word_token
```

Out[7]:
```
['Tokenization',
 'is',
 'the',
 'first',
 'step',
 'in',
 'text',
 'analytics',
 '.',
 'The',
 'process',
 'of',
 'breaking',
 'down',
 'a',
 'text',
 'paragraph',
 'into',
 'smaller',
 'chunks',
 'such',
 'word',
 'or',
 'sentence',
 'is',
 'called',
 'Tokenization']
```

> Removing Punctuation and Stop words

In [8]:
```
from nltk.corpus import stopwords
```

In [9]:
```
stopword = set(stopwords.words("english"))
```

In [10]:
```
stopword
```

Out[10]:
```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
```

In [11]:
```
text = "How to Remove Stopwords with NLTK Library in python?"
```

In [12]:
```
import re
text = re.sub('[^a-z A-Z]',' ',text)
```

In [13]:
```
text
```

Out[13]:
```
'How to Remove Stopwords with NLTK Library in python '
```

In [14]:
```
text1 = re.sub('[^a-z A-Z ?]',' ',text)
```

In [15]:
```
text1
```

Out[15]:
```
'How to Remove Stopwords with NLTK Library in python '
```

In [16]:
```
tokens = word_tokenize(text.lower())
```

In [17]:
```
tokens
```

Out[17]:
```
['how', 'to', 'remove', 'stopwords', 'with', 'nltk', 'library', 'in', 'p
ython']
```

In [18]:
```
tokens = word_tokenize(text.upper())
```

In [19]:
```
tokens
```

Out[19]:
```
['HOW', 'TO', 'REMOVE', 'STOPWORDS', 'WITH', 'NLTK', 'LIBRARY', 'IN', 'P
YTHON']
```

> Filtering Text

In [20]:
```
filtered_text = ["Welcome to DSBDA Lab"]
for word in tokens:
    if word not in stopword:
        filtered_text.append(word)
```

In [21]:
```
filtered_text
```

Out[21]:
```
['Welcome to DSBDA Lab',
 'HOW',
 'TO',
 'REMOVE',
 'STOPWORDS',
 'WITH',
 'NLTK',
 'LIBRARY',
 'IN',
 'PYTHON']
```

In [22]:
```
from nltk.stem import PorterStemmer
```

In [23]:
```
e_words = ["wait","waiting","waited","waits"]
e_words
ps = PorterStemmer()
for w in e_words:
    rootword = ps.stem(w)
    print(rootword)
```

```
wait
wait
wait
wait
```

In [24]:
```
e1_words = ["studies","studying","cries","crying"]
ps = PorterStemmer()
for w in e1_words:
    rootword = ps.stem(w)
    print(rootword)
```

```
studi
studi
cri
cri
```

> Perform Lemmatization

In [25]:
```
from nltk.stem import WordNetLemmatizer
```

In [26]:
```
word_net = WordNetLemmatizer()
```

In [27]:
```
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
```

In [28]:
```
tokenization
```

Out[28]:
```
['studies', 'studying', 'cries', 'cry']
```

In [29]:
```
for w in tokenization:
    print("Lemma for {} is {}".format(w, word_net.lemmatize(w)))
```

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

> Apply POS tagging to text

In [30]:
```
import nltk
from nltk.tokenize import word_tokenize
```

In [31]:
```
data = "The pink sweater fits her perfectly"
words = word_tokenize(data)
```

In [32]:
```
words
```

Out[32]:
```
['The', 'pink', 'sweater', 'fits', 'her', 'perfectly']
```

In [33]:
```
for word in words:
    print(nltk.pos_tag([word]))
```

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

PART 2

i) Import necessary libraries

In [34]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

ii) Initializing the documents

In [35]:
```
documentA = "jupiter is the largest planet"
documentB = "mars is the fourth planet from the sun"
```

In [36]:
```
documentA
```

Out[36]: 'jupiter is the largest planet'

In [37]:
```
documentB
```

Out[37]: 'mars is the fourth planet from the sun'

iii) Create BagofWords(BOW) for A and B

In [38]:
```
bagofwordsA = documentA.split(' ')
```

In [39]:
```
bagofwordsA
```

Out[39]: ['jupiter', 'is', 'the', 'largest', 'planet']

In [40]:
```
print(bagofwordsA)
```

```
['jupiter', 'is', 'the', 'largest', 'planet']
```

In [41]:
```
bagofwordsB = documentB.split(" ")
```

In [42]:
```
bagofwordsB
```

Out[42]: ['mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'sun']

iv) Create collection of unique words from documents

In [43]:
```
unique = set(bagofwordsA).union(bagofwordsB)
```

In [44]:
```
unique
```

Out[44]:
```
{'fourth', 'from', 'is', 'jupiter', 'largest', 'mars', 'planet', 'sun',
'the'}
```

v) Create dictionary of words and their occurance for each document in the corpus

In [45]:
```
numofwordsA = dict.fromkeys(unique,0)
for words in bagofwordsA:
    numofwordsA[words]+=1
```

In [46]:
```
numofwordsA
```

Out[46]:
```
{'planet': 1,
 'largest': 1,
 'sun': 0,
 'is': 1,
 'jupiter': 1,
 'from': 0,
 'the': 1,
 'fourth': 0,
 'mars': 0}
```

In [47]:
```
numofwordsB = dict.fromkeys(unique,0)
for words in bagofwordsB:
    numofwordsB[words]+=1
```

In [48]:
```
numofwordsB
```

Out[48]:
```
{'planet': 1,
 'largest': 0,
 'sun': 1,
 'is': 1,
 'jupiter': 0,
 'from': 1,
 'the': 2,
 'fourth': 1,
 'mars': 1}
```

vi) Complete the term frequency for each of our documents

In [49]:
```
def computeTF(wordDict,bagofword):
    TFDict = {}
    bagofwordsCount = len(bagofword)
    for word,count in wordDict.items():
        TFDict[word]=count/float(bagofwordsCount)
    return TFDict
```

In [50]:
```
TFA = computeTF(numofwordsA,bagofwordsA)
```

In [51]:
```
TFB = computeTF(numofwordsB,bagofwordsB)
```

In [52]:
```
TFA
```

Out[52]:
```
{'planet': 0.2,
 'largest': 0.2,
 'sun': 0.0,
 'is': 0.2,
 'jupiter': 0.2,
 'from': 0.0,
 'the': 0.2,
 'fourth': 0.0,
 'mars': 0.0}
```

In [53]:
```
TFB
```

Out[53]:
```
{'planet': 0.125,
 'largest': 0.0,
 'sun': 0.125,
 'is': 0.125,
 'jupiter': 0.0,
 'from': 0.125,
 'the': 0.25,
 'fourth': 0.125,
 'mars': 0.125}
```

vii) Compute the term inverse document frequency

In [54]:
```
def computeIDF(document):
    import math
    n = len(document)
    idDict = dict.fromkeys(document[0].keys(),0)
    for document in document:
        for word,val in document.items():
            if val>0:
                idDict[word]+=1
    for word,val in idDict.items():
        idDict[word]=math.log(n/float(val))
    return idDict
```

In [55]:
```
IDFS = computeIDF([numofwordsA,numofwordsB])
```

In [56]:
```
IDFS
```

Out[56]:
```
{'planet': 0.0,
 'largest': 0.6931471805599453,
 'sun': 0.6931471805599453,
 'is': 0.0,
 'jupiter': 0.6931471805599453,
 'from': 0.6931471805599453,
 'the': 0.0,
 'fourth': 0.6931471805599453,
 'mars': 0.6931471805599453}
```

viii) Compute TF/IDF for all words

In [57]:
```
def computeTFIDF(tfbagofwords,IDFS):
    tfid = {}
    for word,val in tfbagofwords.items():
        tfid[word] = val*IDFS[word]
    return tfid
```

In [58]:
```
tfidfA = computeTFIDF(TFA,IDFS)
```

In [59]:
```
tfidfA
```

Out[59]:
```
{'planet': 0.0,
 'largest': 0.13862943611198905,
 'sun': 0.0,
 'is': 0.0,
 'jupiter': 0.13862943611198905,
 'from': 0.0,
 'the': 0.0,
 'fourth': 0.0,
 'mars': 0.0}
```

In [60]:
```
tfidfB = computeTFIDF(TFB,IDFS)
```

In [61]:
```
tfidfB
```

Out[61]:
```
{'planet': 0.0,
 'largest': 0.0,
 'sun': 0.08664339756999316,
 'is': 0.0,
 'jupiter': 0.0,
 'from': 0.08664339756999316,
 'the': 0.0,
 'fourth': 0.08664339756999316,
 'mars': 0.08664339756999316}
```

In [62]:
```
df =pd.DataFrame([tfidfA,tfidfB])
```

In [63]:
```
df
```

Out[63]:

	planet	largest	sun	is	jupiter	from	the	fourth	mars
0	0.0	0.138629	0.000000	0.0	0.138629	0.000000	0.0	0.000000	0.000000
1	0.0	0.000000	0.086643	0.0	0.000000	0.086643	0.0	0.086643	0.086643