Assignment 7 : Text Analytics

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

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

Part 1:

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

In [3]:

```
#Installation of punkt from nltk

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

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

Out[3]:

```
True
```

Tokenization

In [4]:

```
from nltk import word_tokenize, sent_tokenize

sent = "Sachin is considered to be one of the greatest cricket players. Virat is the ca
ptain of the  Indian cricket team"
print(word_tokenize(sent))
print(sent_tokenize(sent))
```

```
['Sachin', 'is', 'considered', 'to', 'be', 'one', 'of', 'the', 'greatest',
'cricket', 'players', '.', 'Virat', 'is', 'the', 'captain', 'of', 'the',
'Indian', 'cricket', 'team']
['Sachin is considered to be one of the greatest cricket players.', 'Virat
is the captain of the  Indian cricket team']
```

Stop Words Removal

In [5]:

```
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
stop_words = stopwords.words('english')
print(stop_words)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselv
es', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'hersel
f', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'the
mselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'hav
e', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'th
e', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'on
ce', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'no
t', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "would
n't"]
```

In [6]:

```
token = word_tokenize(sent)
cleaned_token = []
for word in token:
  if word not in stop_words:
    cleaned_token.append(word)

print("This is the unclean version : ",token)
print("This is the cleaned version : ",cleaned_token)
```

```
This is the unclean version :  ['Sachin', 'is', 'considered', 'to', 'be',
'one', 'of', 'the', 'greatest', 'cricket', 'players', '.', 'Virat', 'is',
'the', 'captain', 'of', 'the', 'Indian', 'cricket', 'team']
This is the cleaned version :  ['Sachin', 'considered', 'one', 'greatest',
'cricket', 'players', '.', 'Virat', 'captain', 'Indian', 'cricket', 'tea
m']
```

In [7]:

```
words = [cleaned_token.lower() for cleaned_token in cleaned_token if cleaned_token.isal
pha()]
```

In [8]:

```
print(words)
```

```
['sachin', 'considered', 'one', 'greatest', 'cricket', 'players', 'virat',
'captain', 'indian', 'cricket', 'team']
```

Stemming

Stemming just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling.

In [9]:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
port_stemmer_output = [stemmer.stem(words) for words in words]
print(port_stemmer_output)
```

```
['sachin', 'consid', 'one', 'greatest', 'cricket', 'player', 'virat', 'cap
tain', 'indian', 'cricket', 'team']
```

Lemmatization

Lemmaization considers the context and converts the word to its meaningul base form , which is called Lemma.

In [10]:

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
lemmatizer_output = [lemmatizer.lemmatize(words) for words in words]
print(lemmatizer_output)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
['sachin', 'considered', 'one', 'greatest', 'cricket', 'player', 'virat',
'captain', 'indian', 'cricket', 'team']
```

POS Tagging

In [11]:

```
from nltk import pos_tag
import nltk
nltk.download('averaged_perceptron_tagger')
token = word_tokenize(sent)
cleaned_token = []
for word in token:
  if word not in stop_words:
    cleaned_token.append(word)
tagged = pos_tag(cleaned_token)
print(tagged)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('Sachin', 'NNP'), ('considered', 'VBD'), ('one', 'CD'), ('greatest', 'JJ
S'), ('cricket', 'NN'), ('players', 'NNS'), ('.', '.'), ('Virat', 'NNP'),
('captain', 'NN'), ('Indian', 'JJ'), ('cricket', 'NN'), ('team', 'NN')]
```

Part 2:

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

In [12]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

In [13]:

```
docs = [  "Sachin is considered to be one of the greatest cricket players",
          "Federer is considered one of the greatest tennis players",
          "Nadal is considered one of the greatest tennis players",
          "Virat is the captain of the Indian cricket team"]
```

In [14]:

```
vectorizer = TfidfVectorizer(analyzer = "word", norm = None , use_idf = True , smooth_i
df=True)
Mat = vectorizer.fit(docs)
print(Mat.vocabulary_)
```

```
{'sachin': 12, 'is': 7, 'considered': 2, 'to': 16, 'be': 0, 'one': 10, 'o
f': 9, 'the': 15, 'greatest': 5, 'cricket': 3, 'players': 11, 'federer':
4, 'tennis': 14, 'nadal': 8, 'virat': 17, 'captain': 1, 'indian': 6, 'tea
m': 13}
```

In [15]:

```
tfidfMat = vectorizer.fit_transform(docs)
```

In [16]:

```
print(tfidfMat)
```

```
  (0, 11)	1.2231435513142097
  (0, 3)	1.5108256237659907
  (0, 5)	1.2231435513142097
  (0, 15)	1.0
  (0, 9)	1.0
  (0, 10)	1.2231435513142097
  (0, 0)	1.916290731874155
  (0, 16)	1.916290731874155
  (0, 2)	1.2231435513142097
  (0, 7)	1.0
  (0, 12)	1.916290731874155
  (1, 14)	1.5108256237659907
  (1, 4)	1.916290731874155
  (1, 11)	1.2231435513142097
  (1, 5)	1.2231435513142097
  (1, 15)	1.0
  (1, 9)	1.0
  (1, 10)	1.2231435513142097
  (1, 2)	1.2231435513142097
  (1, 7)	1.0
  (2, 8)	1.916290731874155
  (2, 14)	1.5108256237659907
  (2, 11)	1.2231435513142097
  (2, 5)	1.2231435513142097
  (2, 15)	1.0
  (2, 9)	1.0
  (2, 10)	1.2231435513142097
  (2, 2)	1.2231435513142097
  (2, 7)	1.0
  (3, 13)	1.916290731874155
  (3, 6)	1.916290731874155
  (3, 1)	1.916290731874155
  (3, 17)	1.916290731874155
  (3, 3)	1.5108256237659907
  (3, 15)	2.0
  (3, 9)	1.0
  (3, 7)	1.0
```

In [17]:

```
features_names = vectorizer.get_feature_names_out()
print(features_names)
```

```
['be' 'captain' 'considered' 'cricket' 'federer' 'greatest' 'indian' 'is'
 'nadal' 'of' 'one' 'players' 'sachin' 'team' 'tennis' 'the' 'to' 'virat']
```

In [18]:

```
dense = tfidfMat.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist , columns = features_names)
```

In [19]:

```
df
```

Out[19]:

	be	captain	considered	cricket	federer	greatest	indian	is	nadal	c
0	1.916291	0.000000	1.223144	1.510826	0.000000	1.223144	0.000000	1.0	0.000000	1.
1	0.000000	0.000000	1.223144	0.000000	1.916291	1.223144	0.000000	1.0	0.000000	1.
2	0.000000	0.000000	1.223144	0.000000	0.000000	1.223144	0.000000	1.0	1.916291	1.
3	0.000000	1.916291	0.000000	1.510826	0.000000	0.000000	1.916291	1.0	0.000000	1.

In [20]:

```
features_names = sorted(vectorizer.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Fu
tureWarning: Function get_feature_names is deprecated; get_feature_names i
s deprecated in 1.0 and will be removed in 1.2. Please use get_feature_nam
es_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [21]:

```
docList = ['Doc 1','Doc 2','Doc 3','Doc 4']
skDocsIfIdfdf = pd.DataFrame(tfidfMat.todense(),index = sorted(docList),  columns=featu
res_names)
print(skDocsIfIdfdf)
```

```
             be   captain  considered   cricket   federer  greatest     ind
ian  \
Doc 1  1.916291  0.000000    1.223144  1.510826  0.000000  1.223144  0.000
000
Doc 2  0.000000  0.000000    1.223144  0.000000  1.916291  1.223144  0.000
000
Doc 3  0.000000  0.000000    1.223144  0.000000  0.000000  1.223144  0.000
000
Doc 4  0.000000  1.916291    0.000000  1.510826  0.000000  0.000000  1.916
291

        is     nadal   of       one   players    sachin      team    tenni
s  \
Doc 1  1.0  0.000000  1.0  1.223144  1.223144  1.916291  0.000000  0.00000
0
Doc 2  1.0  0.000000  1.0  1.223144  1.223144  0.000000  0.000000  1.51082
6
Doc 3  1.0  1.916291  1.0  1.223144  1.223144  0.000000  0.000000  1.51082
6
Doc 4  1.0  0.000000  1.0  0.000000  0.000000  0.000000  1.916291  0.00000
0

       the        to     virat
Doc 1  1.0  1.916291  0.000000
Doc 2  1.0  0.000000  0.000000
Doc 3  1.0  0.000000  0.000000
Doc 4  2.0  0.000000  1.916291
```

In [22]:

```
#Compute Cosine Similarity
csim = cosine_similarity(tfidfMat,tfidfMat)
```

In [23]:

```
csimDf = pd.DataFrame(csim,index=sorted(docList),columns=sorted(docList))
```

In [24]:

```
print(csimDf)
```

```
          Doc 1     Doc 2     Doc 3     Doc 4
Doc 1  1.000000  0.492416  0.492416  0.277687
Doc 2  0.492416  1.000000  0.754190  0.215926
Doc 3  0.492416  0.754190  1.000000  0.215926
Doc 4  0.277687  0.215926  0.215926  1.000000
```