Name: Pranav Mehendale

Roll No.: TCOD34

Batch: T11

Assignment 7 - Group A

Text Analytics

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

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

```
import nltk

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

```
True
```

```
text= "Tokenization is the first step in text analytics. The process
of breaking down a text paragraph into smaller chunks such as words or
sentences is called Tokenization."
```

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 text paragraph into smaller chunks such as words or
sentences is called Tokenization.']
```

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', 'process', 'of', 'breaking', 'down', 'a',
'text', 'paragraph', 'into', 'smaller', 'chunks', 'such', 'as',
'words', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

Removing punctuation and stop word

```
from nltk.corpus import stopwords
import re

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

```
{'o', 'am', 'y', "needn't", 'needn', 'have', 'more', 'once', 'off',
'for', 'has', 'from', 't', 'whom', 'of', "won't", 'won', 'or', 'hadn',
"weren't", "mightn't", 'through', 'up', 'such', "you're", 'had',
'when', 'an', 'her', 'being', 'was', 'him', 'this', 'nor', 'than',
'what', 'were', "that'll", 'them', 'ours', 'very', 'by', 'hasn',
'our', 'few', 'ain', 'mustn', 'myself', 'about', 'same', 'the',
'before', 'only', 'ourselves', 'a', 's', "haven't", 'hers', "wasn't",
"it's", 'should', 'in', 'you', 'yours', "wouldn't", 'ma', 'down',
'itself', 'with', 'how', 're', 'will', 'isn', 'do', 'she', 'is',
'can', "aren't", 'wouldn', 'if', 'don', "you'll", "hadn't", 'they',
'now', 'shouldn', 'so', 'couldn', 'no', 'where', 'these', 'most',
"couldn't", 'under', 'are', 'aren', 'there', 'other', 'each', 'who',
'their', 'on', "don't", 'it', 'out', 'over', 'some', "you'd", 'until',
'that', 'against', 'then', 'any', 'did', 'll', "shouldn't", "hasn't",
'after', 'your', 'themselves', 'not', 'above', 'd', 'mightn', 'own',
'be', 'just', 'herself', 'we', 'both', 'into', 'again', 'as', 'its',
'but', 'been', 'haven', 'does', 'too', 'yourself', 'during', 'having',
'and', 'because', 'at', 'further', 'below', 'here', 'didn', 'doesn',
"she's", "didn't", "doesn't", 'those', 'shan', 'weren', "isn't",
'why', 'doing', 'theirs', "shan't", 'i', "should've", 'me', 'he',
'while', 'yourselves', 'which', 've', "you've", 'm', "mustn't",
'himself', 'between', 'my', 'his', 'all', 'wasn', 'to'}
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with',
'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library',
'python']
```

Perfoming 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
```

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 {} is
{}".format(w,wordnet_lemmatizer.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

```
import nltk
from nltk.tokenize import word_tokenize
data="The pink sweater fits her perfectly"
words=word_tokenize(data)
for word in words:
    print(nltk.pos_tag([word]))

[('The', 'DT')]
[('pink', 'NN')]
```

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

Algorithm for Creating representation of document by calculating TFIDF

Importing the necessary libraries

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

Initializing the Documents

```python
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

Creating BagofWords for Documents A and B.

```python
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Split unique words from documents A & B

```python
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

Create a dictionary of words and their occurrence for each document in the corpus

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

Compute the term frequency for each of our documents.

```python
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)
```

Computing the term Inverse Document Frequency.

```
import math
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])
idfs
```

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

Compute the term TF/IDF for all words.

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

```
    Jupiter   is    Planet    fourth      Mars       Sun    planet
largest  \
0  0.138629  0.0  0.138629  0.000000  0.000000  0.000000  0.000000
0.138629
1  0.000000  0.0  0.000000  0.086643  0.086643  0.086643  0.086643
0.000000

       from  the
0  0.000000  0.0
1  0.086643  0.0
```