Assignment_No-7

ASSIGNMENT NO: 07

Title: 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.

#Section-A

[5]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[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.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

[5]:
```
True
```

Sample Sentences

[1]:
```
sentence1 = "I will walk 500 miles and I would walk 500 more. Just to be the␣
 ↪man who walks a thousand miles to fall down at your door!"
sentence2 = "I played the play playfully as the players were playing in the␣
 ↪play with playfullness"
```

Tokenization

[3]:
```
from nltk import word_tokenize, sent_tokenize
```

[6]:
```
print('Tokenized words:', word_tokenize(sentence1))
print('\nTokenized sentences:', sent_tokenize(sentence1))
```

```
Tokenized words: ['I', 'will', 'walk', '500', 'miles', 'and', 'I', 'would',
'walk', '500', 'more', '.', 'Just', 'to', 'be', 'the', 'man', 'who', 'walks',
'a', 'thousand', 'miles', 'to', 'fall', 'down', 'at', 'your', 'door', '!']

Tokenized sentences: ['I will walk 500 miles and I would walk 500 more.', 'Just
to be the man who walks a thousand miles to fall down at your door!']
```

POS Tagging

[7]:
```
from nltk import pos_tag
token = word_tokenize(sentence1) + word_tokenize(sentence2)
tagged = pos_tag(token)
print("Tagging Parts of Speech:", tagged)
```

```
Tagging Parts of Speech: [('I', 'PRP'), ('will', 'MD'), ('walk', 'VB'), ('500',
'CD'), ('miles', 'NNS'), ('and', 'CC'), ('I', 'PRP'), ('would', 'MD'), ('walk',
'VB'), ('500', 'CD'), ('more', 'JJR'), ('.', '.'), ('Just', 'NNP'), ('to',
'TO'), ('be', 'VB'), ('the', 'DT'), ('man', 'NN'), ('who', 'WP'), ('walks',
'VBZ'), ('a', 'DT'), ('thousand', 'NN'), ('miles', 'NNS'), ('to', 'TO'),
('fall', 'VB'), ('down', 'RP'), ('at', 'IN'), ('your', 'PRP$'), ('door', 'NN'),
('!', '.'), ('I', 'PRP'), ('played', 'VBD'), ('the', 'DT'), ('play', 'NN'),
('playfully', 'RB'), ('as', 'IN'), ('the', 'DT'), ('players', 'NNS'), ('were',
'VBD'), ('playing', 'VBG'), ('in', 'IN'), ('the', 'DT'), ('play', 'NN'),
('with', 'IN'), ('playfullness', 'NN')]
```

Stop-Words Removal

[9]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
token = word_tokenize(sentence1)
cleaned_token = []
for word in token:
  if word not in stop_words:
    cleaned_token.append(word)
print('Unclean version:', token)
print('\nCleaned version:', cleaned_token)
```

```
Unclean version: ['I', 'will', 'walk', '500', 'miles', 'and', 'I', 'would',
'walk', '500', 'more', '.', 'Just', 'to', 'be', 'the', 'man', 'who', 'walks',
'a', 'thousand', 'miles', 'to', 'fall', 'down', 'at', 'your', 'door', '!']

Cleaned version: ['I', 'walk', '500', 'miles', 'I', 'would', 'walk', '500', '.',
'Just', 'man', 'walks', 'thousand', 'miles', 'fall', 'door', '!']
```

Stemming

[10]:
```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
token = word_tokenize(sentence2)
stemmed = [stemmer.stem(word) for word in token]
```

```
print(" ".join(stemmed))
```

```
i play the play play as the player were play in the play with playful
```

Lemmatization

[11]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
token = word_tokenize(sentence2)
lemmatized_output = [lemmatizer.lemmatize(word) for word in token]
print(" ".join(lemmatized_output))
```

```
I played the play playfully a the player were playing in the play with
playfullness
```

1 Section-B

[12]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

[13]:
```
documentA = 'The old oak tree stood tall in the middle of the forest, its␣
 ↪branches reaching out like ancient fingers'
documentB = 'The children laughed joyfully as they played in the park, their␣
 ↪voices echoing through the trees'
```

Creating Bag of Words

[14]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

[16]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

[19]:
```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
  numOfWordsA[word] += 1
  numOfWordsB = dict.fromkeys(uniqueWords, 0)

for word in bagOfWordsB:
  numOfWordsB[word] += 1
```

Computing Term Frequency (TF)

[20]:
```
def computeTF(wordDict, bagOfWords):
  tfDict = {}
  bagOfWordsCount = len(bagOfWords)
  for word, count in wordDict.items():
    tfDict[word] = count / float(bagOfWordsCount)
  return tfDict
```

[21]:
```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

[22]:
```
tfA
```

[22]:
```
{'its': 0.05263157894736842,
 'The': 0.05263157894736842,
 'tree': 0.05263157894736842,
 'old': 0.05263157894736842,
 'reaching': 0.05263157894736842,
 'as': 0.0,
 'voices': 0.0,
 'in': 0.05263157894736842,
 'branches': 0.05263157894736842,
 'out': 0.05263157894736842,
 'joyfully': 0.0,
 'tall': 0.05263157894736842,
 'children': 0.0,
 'middle': 0.05263157894736842,
 'ancient': 0.05263157894736842,
 'fingers': 0.05263157894736842,
 'forest,': 0.05263157894736842,
 'like': 0.05263157894736842,
 'trees': 0.0,
 'the': 0.10526315789473684,
 'their': 0.0,
 'played': 0.0,
 'they': 0.0,
 'of': 0.05263157894736842,
 'through': 0.0,
 'park,': 0.0,
 'laughed': 0.0,
 'stood': 0.05263157894736842,
 'echoing': 0.0,
 'oak': 0.05263157894736842}
```

[23]:
```
tfB
```

[23]:
```
{'its': 0.0,
 'The': 0.0625,
 'tree': 0.0,
 'old': 0.0,
 'reaching': 0.0,
 'as': 0.0625,
 'voices': 0.0625,
 'in': 0.0625,
 'branches': 0.0,
```

```
 'out': 0.0,
 'joyfully': 0.0625,
 'tall': 0.0,
 'children': 0.0625,
 'middle': 0.0,
 'ancient': 0.0,
 'fingers': 0.0,
 'forest,': 0.0,
 'like': 0.0,
 'trees': 0.0625,
 'the': 0.125,
 'their': 0.0625,
 'played': 0.0625,
 'they': 0.0625,
 'of': 0.0,
 'through': 0.0625,
 'park,': 0.0625,
 'laughed': 0.0625,
 'stood': 0.0,
 'echoing': 0.0625,
 'oak': 0.0}
```

Computing Inverse Document Frequency (IDF)

[24]:
```
def computeIDF(documents):
  import math
  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
```

[24]:
```
{'its': 0.6931471805599453,
 'The': 0.0,
 'tree': 0.6931471805599453,
 'old': 0.6931471805599453,
 'reaching': 0.6931471805599453,
 'as': 0.6931471805599453,
 'voices': 0.6931471805599453,
 'in': 0.0,
```

```
 'branches': 0.6931471805599453,
 'out': 0.6931471805599453,
 'joyfully': 0.6931471805599453,
 'tall': 0.6931471805599453,
 'children': 0.6931471805599453,
 'middle': 0.6931471805599453,
 'ancient': 0.6931471805599453,
 'fingers': 0.6931471805599453,
 'forest,': 0.6931471805599453,
 'like': 0.6931471805599453,
 'trees': 0.6931471805599453,
 'the': 0.0,
 'their': 0.6931471805599453,
 'played': 0.6931471805599453,
 'they': 0.6931471805599453,
 'of': 0.6931471805599453,
 'through': 0.6931471805599453,
 'park,': 0.6931471805599453,
 'laughed': 0.6931471805599453,
 'stood': 0.6931471805599453,
 'echoing': 0.6931471805599453,
 'oak': 0.6931471805599453}
```

Computing Term Frequency-Inverse Document Frequency (TF/IDF)

[25]:
```
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
```

[25]:

	its	The	tree	old	reaching	as	voices	in	\
0	0.036481	0.0	0.036481	0.036481	0.036481	0.000000	0.000000	0.0	
1	0.000000	0.0	0.000000	0.000000	0.000000	0.043322	0.043322	0.0	

	branches	out	…	their	played	they	of	through	\
0	0.036481	0.036481	…	0.000000	0.000000	0.000000	0.036481	0.000000	
1	0.000000	0.000000	…	0.043322	0.043322	0.043322	0.000000	0.043322	

	park,	laughed	stood	echoing	oak
0	0.000000	0.000000	0.036481	0.000000	0.036481
1	0.043322	0.043322	0.000000	0.043322	0.000000

[2 rows x 30 columns]