ass7-1-1

April 2, 2025

[45]:

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

[45]:

```
True
```

[41]:

```
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."
text
```

[41]:

```
'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.'
```

[5]:

```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)
#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.']
['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', '.']
```

[11]:

```
import re
# print stop words of English
from nltk.corpus import stopwords
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)
```

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

[15]:

```python
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
```

[19]:

```python
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
```

[53]:

```python
import nltk
nltk.download('averaged_perceptron_tagger_eng')
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]))
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger_eng.zip.
```

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

[3]:

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

[5]:

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

[11]:

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

[13]:

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

[15]:

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
numOfWordsB = dict.fromkeys(uniqueWords, 0)

for word in bagOfWordsA:
    numOfWordsA[word] += 1

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

[17]:

```
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)  # Total words in document
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)  # TF formula
    return tfDict

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

[19]:

```
def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)

    # Count number of documents containing each word
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    # Apply IDF formula
    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))

    return idfDict

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

[21]:

```
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)
```

[23]:

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

[23]:

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

    is      Mars
0  0.0  0.000000
1  0.0  0.086643
```

[ ]: