assignment7

May 7, 2024

[9]:
```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
from nltk import sent_tokenize
from nltk import word_tokenize
from nltk.corpus import stopwords
```

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

[11]:
```
# Sentence Tokenization
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."
tokenized_text = nltk.sent_tokenize(text)
print("\n Sentence Tokenization: \n", tokenized_text)
```

```
 Sentence 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.']
```

[13]:
```
# Word Tokenization
tokenized_word = nltk.word_tokenize(text)
print('\nWord Tokeniztion: \n', tokenized_word)
```

```
Word Tokeniztion:
 ['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', '.']
```

[15]:
```
# POS Tagging
tagged_text = nltk.pos_tag(tokenized_word)
print("\nPOS Tagging: \n", tagged_text)
```

```
POS Tagging:
 [('Tokenization', 'NN'), ('is', 'VBZ'), ('the', 'DT'), ('first', 'JJ'),
('step', 'NN'), ('in', 'IN'), ('text', 'JJ'), ('analytics', 'NNS'), ('.', '.'),
('The', 'DT'), ('process', 'NN'), ('of', 'IN'), ('breaking', 'VBG'), ('down',
'RP'), ('a', 'DT'), ('text', 'NN'), ('paragraph', 'NN'), ('into', 'IN'),
('smaller', 'JJR'), ('chunks', 'NNS'), ('such', 'JJ'), ('as', 'IN'), ('words',
'NNS'), ('or', 'CC'), ('sentences', 'NNS'), ('is', 'VBZ'), ('called', 'VBN'),
('Tokenization', 'NN'), ('.', '.')]
```

[19]:
```
# Stop words removal
stop_words = set(nltk.corpus.stopwords.words("english"))
text = "How to remove stop words with NLTK library in Python?"
text = re.sub('[^a-zA-Z]', ' ', text)
tokens = nltk.word_tokenize(text.lower())
filtered_text = [w for w in tokens if w not in stop_words]
print ("Tokenized Sentence:", tokens)
print ("Filtered Sentence:", filtered_text)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 4
      2 stop_words = set(nltk.corpus.stopwords.words("english"))
      3 text = "How to remove stop words with NLTK library in Python?"
----> 4 text = re.sub('[^a-zA-Z]', ' ', text)
      5 tokens = nltk.word_tokenize(text.lower())
      6 filtered_text = [w for w in tokens if w not in stop_words]

NameError: name 're' is not defined
```

[21]:
```
import nltk
import re
```

[24]:
```
# Stop words removal
stop_words = set(nltk.corpus.stopwords.words("english"))
text = "How to remove stop words with NLTK library in Python?"
text = re.sub('[^a-zA-Z]', ' ', text)
tokens = nltk.word_tokenize(text.lower())
filtered_text = [w for w in tokens if w not in stop_words]
print("Tokenized Sentence:", tokens)
print("Filtered Sentence:", filtered_text)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filtered Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

[26]:
```
# Stemming
from nltk.stem import PorterStemmer
e_words = ["wait", "waiting", "waited", "waits"]
ps = PorterStemmer()
for w in e_words:
    rootWord = ps.stem(w)
    print('Stemming for ', w, ': ', rootWord)
```

```
Stemming for  wait :  wait
Stemming for  waiting :  wait
Stemming for  waited :  wait
Stemming for  waits :  wait
```

[28]:
```
# 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
```

[30]:
```
# Algorithm for Create representation of document by calculating TFIDF
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[30], line 2
      1 # Algorithm for Create representation of document by calculating TFIDF
----> 2 from sklearn.feature_extraction.text import TfidfVectorizer
```

```
ModuleNotFoundError: No module named 'sklearn'
```

[32]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in
c:\users\khara\appdata\local\programs\python\python312\lib\site-packages (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in
c:\users\khara\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.26.3)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\khara\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.13.0)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\khara\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\khara\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

[34]:
```
# Step 1: Import the necessary libraries.
documentA = 'Jupiter is the largest planet'
documentB = 'Mars is the fourth planet from the Sun'
```

[35]:
```
# Step 2: Initialize the Documents.
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

[37]:
```
# Step 3: Create BagofWords (BoW) for Document A and B. word tokenization
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

[39]:
```
# Step 4: Create Collection of Unique words from Document A and B.
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1 #How many times each word is repeated
numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

[41]:
```
# Step 5: Compute the term frequency for each of our documents.
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)
```

[43]:
```
# Step 6: Compute the term Inverse Document Frequency.
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():
        if val > 0: idfDict[word] = math.log(N / float(val))
        else: idfDict[word] = 0
    return idfDict

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

[47]:
```
# Step 7: 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)

print('----------------Term Frequency----------------------')
df = pd.DataFrame([tfA, tfB])
print(df)

print('----------------Inverse Document Frequency----------------------')
print(idfs)

print('------------------- TF-IDF-------------------------------------')
df = pd.DataFrame([tfidfA, tfidfB])
print(df)
```

```
----------------Term Frequency----------------------
```

```
--------------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)
Cell In[47], line 12
      9 tfidfB = computeTFIDF(tfB, idfs)
     11 print('----------------Term Frequency----------------------')
---> 12 df = pd.DataFrame([tfA, tfB])
     13 print(df)
     15 print('----------------Inverse Document Frequency----------------------')

NameError: name 'pd' is not defined
```

[49]:
```python
import pandas as pd

# Step 7: 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

# Example term frequencies (TF)
tfA = {"cat": 3, "dog": 2, "bird": 0}
tfB = {"cat": 1, "dog": 1, "bird": 2}

# Example inverse document frequencies (IDF)
idfs = {"cat": 1.5, "dog": 2.0, "bird": 1.2}

# Compute TF-IDF
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)

print('----------------Term Frequency----------------------')
df = pd.DataFrame([tfA, tfB])
print(df)

print('----------------Inverse Document Frequency----------------------')
print(idfs)

print('------------------- TF-IDF-----------------------------------')
df = pd.DataFrame([tfidfA, tfidfB])
print(df)
```

```
----------------Term Frequency----------------------
   cat  dog  bird
0    3    2     0
1    1    1     2
----------------Inverse Document Frequency----------------------
{'cat': 1.5, 'dog': 2.0, 'bird': 1.2}
```

```
------------------- TF-IDF--------------------------------------
   cat  dog  bird
0  4.5  4.0   0.0
1  1.5  2.0   2.4
```

[ ]: