pratical-7

April 17, 2024

---

1 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 documents by calculating Term Frequency and Inverse Document-Frequency.

[1]:
```
# # 1) Algorithm for Tokenization, POS Tagging, stop words removal, Stemming␣
 ↪and Lemmatization

# # Step 1: Download the required packages
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger')

# # Step 2: Initialize the text
# 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."

# # Step 3: Perform Tokenization
# # 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)

# # Step 4: Removing Punctuations and Stop Word
# # print stop words of English
# import re
# 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)

# # Step 5 : Perform 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)

# # Step 6: 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)))

# # Step 7: Apply POS Tagging to text
# import nltk
# 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]))
```

[12]:

```
import nltk
import re
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.stem import PorterStemmer
```

[3]:

```
# Step 1: Download the required packages
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

[3]:

```
True
```

[4]:

```
# Step 2: Initialize the text
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."
```

[5]:

```
# Step 3: Perform Tokenization
# Sentence Tokenization
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.']
```

[6]:

```
# Word Tokenization
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', '.']
```

[8]:

```
# Step 4: Removing Punctuations and Stop Word
# print stop words of English
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)
```

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

[9]:

```
# Step 5 : Perform Stemming
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
    rootWord=ps.stem(w)
print(rootWord)
```

```
wait
```

[10]:

```
# Step 6: Perform Lemmatization
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
```

[11]:

```python
# Step 7: Apply POS Tagging to text
data="The pink sweater fit her perfectly"
words=word_tokenize(data)
for word in words:
    print(nltk.pos_tag([word]))
```

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

[13]:

```python
# # 2) Algorithm for Create representation of document by calculating TFIDF

# # Step 1: Import the necessary libraries.
# import pandas as pd
# import math
# from sklearn.feature_extraction.text import TfidfVectorizer

# # Step 2: Initialize the Documents.
# documentA = 'Jupiter is the largest Planet'
# documentB = 'Mars is the fourth planet from the Sun'

# # Step 3: Create BagofWords (BoW) for Document A and B.
# bagOfWordsA = documentA.split(' ')
# bagOfWordsB = documentB.split(' ')

# # Step 4: Create Collection of Unique words from Document A and B.
# uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))

# # Step 5: Create a dictionary of words and their occurrence for each document␣
 ↪in the corpus
# numOfWordsA = dict.fromkeys(uniqueWords, 0)
# for word in bagOfWordsA:
#     numOfWordsA[word] += 1
#     numOfWordsB = dict.fromkeys(uniqueWords, 0)
# for word in bagOfWordsB:
#     numOfWordsB[word] += 1

# # Step 6: 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)

# # Step 7: Compute the term Inverse Document Frequency.
# 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

# # Step 8: 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
```

[14]:

```
# Step 1: Import the necessary libraries.
import pandas as pd
import math
from sklearn.feature_extraction.text import TfidfVectorizer
```

[15]:

```
# Step 2: Initialize the Documents.
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

[16]:

```
# Step 3: Create BagofWords (BoW) for Document A and B.
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

[17]:

```
# Step 4: Create Collection of Unique words from Document A and B.
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

[18]:

```
# Step 5: Create a dictionary of words and their occurrence for each document in the corpus
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
    numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

[19]:

```
# Step 6: 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)
```

[20]:

```
# Step 7: Compute the term Inverse Document Frequency.
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
```

[20]:

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

[21]:

```
# Step 8: 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
```

[22]:

```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

[22]:

```
    Jupiter  the    Planet       Sun   is    planet      from      Mars  \
0  0.138629  0.0  0.138629  0.000000  0.0  0.000000  0.000000  0.000000
1  0.000000  0.0  0.000000  0.086643  0.0  0.086643  0.086643  0.086643

    largest    fourth
0  0.138629  0.000000
1  0.000000  0.086643
```

[ ]:

```

```