Text Analytics

May 2, 2022

[1]:
```
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\abc\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\abc\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\abc\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\abc\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[1]:
```
True
```

[2]:
```
text = "Tokenization is the first step in text analytics. The process of␣
 ↪breaking down a text paragraph into smaller chunks such as worlds or␣
 ↪sentences is called tokenization."
```

[3]:
```
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 worlds or sentences is called
tokenization.']
```

[4]:
```
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', 'worlds', 'or', 'sentences', 'is', 'called',
'tokenization', '.']
```

[5]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words("english"))
print(stop_words)
```

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

[6]:
```
tokens = word_tokenize(text.lower())
filtered_text = []
stop_word = []
for w in tokens:
    if w not in stop_words:
        filtered_text.append(w)
    else:
        stop_word.append(w)
print("Tokenized Words:\n",tokens)
print("\nStop Words:\n",stop_word)
print("\nFiltered Text:\n",filtered_text)
```

```
Tokenized Words:
 ['tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.',
'the', 'process', 'of', 'breaking', 'down', 'a', 'text', 'paragraph', 'into',
'smaller', 'chunks', 'such', 'as', 'worlds', 'or', 'sentences', 'is', 'called',
'tokenization', '.']
```

```
Stop Words:
 ['is', 'the', 'in', 'the', 'of', 'down', 'a', 'into', 'such', 'as', 'or', 'is']

Filtered Text:
 ['tokenization', 'first', 'step', 'text', 'analytics', '.', 'process',
'breaking', 'text', 'paragraph', 'smaller', 'chunks', 'worlds', 'sentences',
'called', 'tokenization', '.']
```

[7]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
e_words = ['waits','waiting','wait','waited']
for w in e_words:
    r_word = ps.stem(w)
print(r_word)
```

```
wait
```

[8]:
```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print(f"Lemma    for    {w}    is    {wordnet_lemmatizer.lemmatize(w)}")
```

```
Lemma    for    studies    is    study
Lemma    for    studying    is    studying
Lemma    for    cries    is    cry
Lemma    for    cry    is    cry
```

[9]:
```
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]))
```

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

1 calculating TFIDF

[10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

[11]:
```
documentA = 'Jupiter is the best notebook for data science'
documentB = 'Google Colab is also best for data science'
```

[12]:
```
bowA = documentA.split(' ')
bowB = documentB.split(' ')
```

[13]:
```
uniquewords = set(bowA).union(set(bowB))
```

[14]:
```
nowA = dict.fromkeys(uniquewords, 0)
for word in bowA:
    nowA[word] += 1
    nowB = dict.fromkeys(uniquewords, 0)
for word in bowB:
    nowB[word] += 1
```

[15]:
```
def computeTF(word_dict , bag_of_word):
    tfdict = {}
    bag_of_word_count = len(bag_of_word)
    for word, count in word_dict.items():
        tfdict[word] = count / float(bag_of_word_count)
    return tfdict
tfA = computeTF(nowA,bowA)
tfB = computeTF(nowB,bowB)
```

[16]:
```
print(tfA)
```

```
{'Google': 0.0, 'notebook': 0.125, 'Jupiter': 0.125, 'the': 0.125, 'data':
0.125, 'also': 0.0, 'is': 0.125, 'Colab': 0.0, 'for': 0.125, 'science': 0.125,
'best': 0.125}
```

[17]:
```
print(tfB)
```

```
{'Google': 0.125, 'notebook': 0.0, 'Jupiter': 0.0, 'the': 0.0, 'data': 0.125,
'also': 0.125, 'is': 0.125, 'Colab': 0.125, 'for': 0.125, 'science': 0.125,
'best': 0.125}
```

[18]:
```
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([nowA,nowB])
print(idfs)
```

```
{'Google': 0.6931471805599453, 'notebook': 0.6931471805599453, 'Jupiter':
0.6931471805599453, 'the': 0.6931471805599453, 'data': 0.0, 'also':
0.6931471805599453, 'is': 0.0, 'Colab': 0.6931471805599453, 'for': 0.0,
'science': 0.0, 'best': 0.0}
```

[19]:
```
import pandas as pd
def computeTFIDF(tfbagofword, idfs):
    tfidf = {}
    for word, val in tfbagofword.items():
        tfidf[word] = val * idfs[word]
    return tfidf
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame(tfidfA, tfidfB)
df
```

[19]:

	Google	notebook	Jupiter	the	data	also	is	Colab	for	\
Google	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
notebook	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
Jupiter	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
the	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
data	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
also	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
is	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
Colab	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
for	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
science	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	
best	0.0	0.086643	0.086643	0.086643	0.0	0.0	0.0	0.0	0.0	

	science	best
Google	0.0	0.0
notebook	0.0	0.0
Jupiter	0.0	0.0
the	0.0	0.0
data	0.0	0.0
also	0.0	0.0
is	0.0	0.0
Colab	0.0	0.0
for	0.0	0.0
science	0.0	0.0
best	0.0	0.0