lklcqqydz

March 31, 2025

[1]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (3.8.1)Note: you may need to restart the
kernel to use updated packages.

Requirement already satisfied: click in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (from nltk) (4.65.0)
Requirement already satisfied: colorama in c:\users\vishwajeet
kulkarni\anaconda3\lib\site-packages (from click->nltk) (0.4.6)
```

[2]:
```
import nltk
```

[3]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to C:\Users\Vishwajeet
[nltk_data]     Kulkarni\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to C:\Users\Vishwajeet
[nltk_data]     Kulkarni\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to C:\Users\Vishwajeet
[nltk_data]     Kulkarni\AppData\Roaming\nltk_data…
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Vishwajeet
[nltk_data]     Kulkarni\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

[3]:
```
True
```

[4]:
```
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]:
```
#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', '.']
```

[7]:
```
# Import necessary libraries
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
# Clean the text (remove non-alphabetic characters)
text = re.sub('[^a-zA-Z]', ' ', text)
# Tokenize the cleaned text
tokens = word_tokenize(text.lower())
# Remove stop words
stop_words = set(stopwords.words("english"))
filtered_text = [w for w in tokens if w not in stop_words]
# Print the results
print("Tokenized Sentence:", tokens)
print("Filtered Sentence:", filtered_text)
```

```
Tokenized Sentence: ['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']
Filtered Sentence: ['tokenization', 'first', 'step', 'text', 'analytics',
'process', 'breaking', 'text', 'paragraph', 'smaller', 'chunks', 'words',
'sentences', 'called', 'tokenization']
```

[11]:
```
# Import the necessary library
from nltk.stem import PorterStemmer
# List of words
```

```
e_words = ["wait", "waiting", "waited", "waits"]
# Initialize the stemmer
ps = PorterStemmer()
# Perform stemming and print the results
for w in e_words:
    rootWord = ps.stem(w)
    print(f"{w} → {rootWord}")
```

```
wait → wait
waiting → wait
waited → wait
waits → wait
```

[13]:
```
# Import necessary libraries
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
# Tokenize the text
tokenization = word_tokenize(text)
# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()
# Perform lemmatization and print the result
for w in tokenization:
    print(f"Lemma for {w}: {lemmatizer.lemmatize(w)}")
```

```
Lemma for Tokenization: Tokenization
Lemma for is: is
Lemma for the: the
Lemma for first: first
Lemma for step: step
Lemma for in: in
Lemma for text: text
Lemma for analytics: analytics
Lemma for The: The
Lemma for process: process
Lemma for of: of
Lemma for breaking: breaking
Lemma for down: down
Lemma for a: a
Lemma for text: text
Lemma for paragraph: paragraph
Lemma for into: into
Lemma for smaller: smaller
Lemma for chunks: chunk
Lemma for such: such
Lemma for as: a
Lemma for words: word
Lemma for or: or
Lemma for sentences: sentence
```

```
Lemma for is: is
Lemma for called: called
Lemma for Tokenization: Tokenization
```

[14]:
```python
# Import necessary libraries
import nltk
from nltk.tokenize import word_tokenize
# Download necessary NLTK resources
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
# Define the text
data = "The pink sweater fit her perfectly"
# Tokenize the text
words = word_tokenize(data)
# Perform POS tagging
for word in words:
    print(nltk.pos_tag([word]))
```

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

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

[20]:
```python
# Import necessary libraries
import pandas as pd  # For data manipulation
from sklearn.feature_extraction.text import TfidfVectorizer  # For TF-IDF␣
 ↪vectorization
```

[21]:
```python
# Define two documents as strings
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

[22]:
```python
# Split the documents into individual words (tokenization)
bagOfWordsA = documentA.split(' ')  # Splits documentA into a list of words
bagOfWordsB = documentB.split(' ')  # Splits documentB into a list of words
```

[23]:
```
# Create a set of unique words from both documents
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

[24]:
```
# Initialize dictionaries with unique words as keys and 0 as initial count
numOfWordsA = dict.fromkeys(uniqueWords, 0)  # For documentA
numOfWordsB = dict.fromkeys(uniqueWords, 0)  # For documentB
# Count occurrences of each word in documentA
for word in bagOfWordsA:
    numOfWordsA[word] += 1
# Count occurrences of each word in documentB
for word in bagOfWordsB:
    numOfWordsB[word] += 1
# Print the word frequency dictionaries
print("Word Frequency in Document A:", numOfWordsA)
print("Word Frequency in Document B:", numOfWordsB)
```

```
Word Frequency in Document A: {'fourth': 0, 'largest': 1, 'Jupiter': 1, 'is': 1,
'planet': 0, 'from': 0, 'Planet': 1, 'Mars': 0, 'Sun': 0, 'the': 1}
Word Frequency in Document B: {'fourth': 1, 'largest': 0, 'Jupiter': 0, 'is': 1,
'planet': 1, 'from': 1, 'Planet': 0, 'Mars': 1, 'Sun': 1, 'the': 2}
```

[25]:
```
# Define a function to compute Term Frequency (TF)
def computeTF(wordDict, bagOfWords):
    tfDict = {}  # Dictionary to store term frequencies
    bagOfWordsCount = len(bagOfWords)  # Total number of words in the document
    # Calculate TF for each word
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)

    return tfDict
# Compute TF for both documents
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
# Print the results
print("TF for Document A:", tfA)
print("TF for Document B:", tfB)
```

```
TF for Document A: {'fourth': 0.0, 'largest': 0.2, 'Jupiter': 0.2, 'is': 0.2,
'planet': 0.0, 'from': 0.0, 'Planet': 0.2, 'Mars': 0.0, 'Sun': 0.0, 'the': 0.2}
TF for Document B: {'fourth': 0.125, 'largest': 0.0, 'Jupiter': 0.0, 'is':
0.125, 'planet': 0.125, 'from': 0.125, 'Planet': 0.0, 'Mars': 0.125, 'Sun':
0.125, 'the': 0.25}
```

[26]:
```
# Import the necessary library
import math
# Define the IDF computation function
def computeIDF(documents):
```

```
    N = len(documents)  # Total number of documents
    # Initialize the IDF dictionary with all unique words and 0 occurrences
    idfDict = dict.fromkeys(documents[0].keys(), 0)
# Count the number of documents containing each word
    for document in documents:
        for word, val in document.items():
            if val > 0:  # Word is present in the document
                idfDict[word] += 1
# Calculate IDF using the formula
    for word, val in idfDict.items():
        idfDict[word] = math.log(N / (float(val) + 1))

    return idfDict
# Compute IDF for both documents
idf = computeIDF([numOfWordsA, numOfWordsB])
# Print the IDF values
print("IDF values:", idf)
```

```
IDF values: {'fourth': 0.0, 'largest': 0.0, 'Jupiter': 0.0, 'is':
-0.40546510810816444, 'planet': 0.0, 'from': 0.0, 'Planet': 0.0, 'Mars': 0.0,
'Sun': 0.0, 'the': -0.40546510810816444}
```

[29]:
```
# Define function to compute TF-IDF values
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}  # Dictionary to store TF-IDF values
    # Calculate TF-IDF for each word
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
# Compute TF-IDF for both documents
tfidfA = computeTFIDF(tfA, idf)  # TF-IDF for Document A
tfidfB = computeTFIDF(tfB, idf)  # TF-IDF for Document B
# Create a DataFrame with the TF-IDF values
df = pd.DataFrame([tfidfA, tfidfB])
# Display the DataFrame
print("TF-IDF DataFrame:")
print(df)
```

```
TF-IDF DataFrame:
   fourth  largest  Jupiter        is  planet  from  Planet  Mars  Sun  \
0     0.0      0.0      0.0 -0.081093     0.0   0.0     0.0   0.0  0.0
1     0.0      0.0      0.0 -0.050683     0.0   0.0     0.0   0.0  0.0

        the
0 -0.081093
1 -0.101366
```