assignment7

April 17, 2025

[11]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages
(3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages
(from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages
(from nltk) (4.67.1)
```

[22]:
```
import nltk
nltk.download('punkt')   #punkt is used for tokenization, dividing a text into␣
 ↪words or sentences.
nltk.download('averaged_perception_tagger')   #function is used for␣
 ↪Part-Of-Speech (POS) tagging, which helps to identify whether a word is a␣
 ↪noun, verb, adjective, etc.
nltk.download("wordnet")    #wordnet is a lexical database used for word␣
 ↪meanings and relationships
nltk.download('stopwords')   #stopwords are common words like "is", "and",␣
 ↪"the" that are usually ignored in text processing.
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Error loading averaged_perception_tagger: Package
[nltk_data]     'averaged_perception_tagger' not found in index
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

[22]:
```
True
```

[41]:
```
from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.tag import pos_tag
from nltk.corpus import wordnet
```

Perform tokenization on document

[42]:
```
# Function to tokenize a document
def tokenize_text(document):
    return word_tokenize(document)    # breaks down the input document (a␣
 ↪string of text) into individual words or tokens.

# Example Usage:
document = "NLP is a field of AI or subset of AI. NLTK is a powerful tool for␣
 ↪text analysis. NLP plays crucial role in various tasks such as building a␣
 ↪chatgpt etc."
tokens = tokenize_text(document)
print("Tokens:", tokens)
```

```
Tokens: ['NLP', 'is', 'a', 'field', 'of', 'AI', 'or', 'subset', 'of', 'AI', '.',
'NLTK', 'is', 'a', 'powerful', 'tool', 'for', 'text', 'analysis', '.', 'NLP',
'plays', 'crucial', 'role', 'in', 'various', 'tasks', 'such', 'as', 'building',
'a', 'chatgpt', 'etc', '.']
```

[48]:
```
import nltk

# Correct download for the averaged perceptron tagger
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

[48]:
```
True
```

[49]:
```
# Apply POS tagging
tagged_tokens = nltk.pos_tag(tokens)  # POS tagging identifies the grammatical␣
 ↪category of each word in a sentence (e.g., noun, verb, adjective)

# Print the tagged tokens
print(tagged_tokens)   # returns list of tuples where each tuple contains a␣
 ↪token and its corresponding POS tag
```

```
[('NLP', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('field', 'NN'), ('of', 'IN'),
('AI', 'NNP'), ('or', 'CC'), ('subset', 'NN'), ('of', 'IN'), ('AI', 'NNP'),
('.', '.'), ('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('powerful', 'JJ'),
('tool', 'NN'), ('for', 'IN'), ('text', 'JJ'), ('analysis', 'NN'), ('.', '.'),
```

```
('NLP', 'NNP'), ('plays', 'VBZ'), ('crucial', 'JJ'), ('role', 'NN'), ('in',
'IN'), ('various', 'JJ'), ('tasks', 'NNS'), ('such', 'JJ'), ('as', 'IN'),
('building', 'VBG'), ('a', 'DT'), ('chatgpt', 'NN'), ('etc', 'NN'), ('.', '.')]
```

[36]:
```
from nltk.corpus import stopwords

# Function to remove stopwords from the token list
def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))      #retrieves a list of␣
 ↪common stopwords for the English language. Then set() operation convertes␣
 ↪list into a set for faster lookups.
    return [word for word in tokens if word.lower() not in stop_words]   #list␣
 ↪comprehension, a concise way to create a new list by iterating over the␣
 ↪tokens and filtering out words that are in the stop_words set.

# Example Usage:
filtered_tokens = remove_stopwords(tokens)   # removes stopwords (common words␣
 ↪like "the", "is", "in", etc.) from a list of tokens.
print("Filtered Tokens (after stopwords removal):", filtered_tokens)
print( ' \n\nStop words are : ', stopwords.words('english'))
```

```
Filtered Tokens (after stopwords removal): ['NLP', 'field', 'AI', 'subset',
'AI', '.', 'NLTK', 'powerful', 'tool', 'text', 'analysis', '.', 'NLP', 'plays',
'crucial', 'role', 'various', 'tasks', 'building', 'chatgpt', 'etc', '.']
```

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

```
"you're", 'yours', 'yourself', 'yourselves', "you've"]
```

[37]:
```
from nltk.stem import PorterStemmer
#Stemming  reduces a word to its root or base form, often by stripping off␣
 ↪prefixes or suffixes.

# Function to apply stemming to a list of tokens
def stemming(tokens):
    stemmer = PorterStemmer()    #PorterStemmer is commonly used algorithm for␣
 ↪stemming in NLP.Creating object of PorterStemmer
    return [stemmer.stem(word) for word in tokens]

# Example Usage:
stemmed_tokens = stemming(filtered_tokens)
print("Stemmed Tokens:", stemmed_tokens)
```

```
Stemmed Tokens: ['nlp', 'field', 'ai', 'subset', 'ai', '.', 'nltk', 'power',
'tool', 'text', 'analysi', '.', 'nlp', 'play', 'crucial', 'role', 'variou',
'task', 'build', 'chatgpt', 'etc', '.']
```

[38]:
```
from nltk.stem import WordNetLemmatizer
#lemmatization reduces a word to its dictionary form, considering its meaning␣
 ↪and part of speech (POS). It generally produces more meaningful results than␣
 ↪stemming.

# Function to apply lemmatization to a list of tokens
def lemmatization(tokens):
    lemmatizer = WordNetLemmatizer()     #tool that performs lemmatization
    return [lemmatizer.lemmatize(word) for word in tokens]   #For each word, it␣
 ↪applies the lemmatize method of the WordNetLemmatizer object to reduce the␣
 ↪word to its base or dictionary form

# Example Usage:
lemmatized_tokens = lemmatization(filtered_tokens)
print("Lemmatized Tokens:", lemmatized_tokens)
```

```
Lemmatized Tokens: ['NLP', 'field', 'AI', 'subset', 'AI', '.', 'NLTK',
'powerful', 'tool', 'text', 'analysis', '.', 'NLP', 'play', 'crucial', 'role',
'various', 'task', 'building', 'chatgpt', 'etc', '.']
```

[39]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Function to compute TF-IDF for a list of documents
def compute_tfidf(documents):       #takes a list of documents as input.
```

```
    vectorizer = TfidfVectorizer()      #utility that transforms text data␣
 ↪(documents) into numerical vectors using the TF-IDF (Term Frequency-Inverse␣
 ↪Document Frequency) technique.
    tfidf_matrix = vectorizer.fit_transform(documents)   #fit() method learns␣
 ↪the vocabulary (the unique words) and calculates the IDF (Inverse Document␣
 ↪Frequency) from the documents.The transform() method then applies this␣
 ↪learned information to transform the list of documents into a TF-IDF matrix␣
 ↪(a sparse matrix).

    # Convert the TF-IDF matrix to a DataFrame for easier visualization
    return pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.
 ↪get_feature_names_out())   #toarray() method converts the sparse matrix␣
 ↪(returned by fit_transform) into a regular dense numpy array, which is␣
 ↪easier to work with.

# Example Usage:
documents = [
    "NLTK is a powerful tool for text analysis.",
    "Text analysis involves techniques like tokenization, stemming, and␣
 ↪lemmatization.",
    "Document preprocessing is essential in text analytics."
]

tfidf_df = compute_tfidf(documents)   #DataFrame that contains the TF-IDF␣
 ↪scores for each word in the documents.
print("\nTF-IDF Representation:")
print(tfidf_df)
```

```
TF-IDF Representation:
   analysis  analytics       and  document  essential       for        in  \
0  0.324124   0.000000  0.000000  0.000000   0.000000  0.426184  0.000000
1  0.270118   0.000000  0.355173  0.000000   0.000000  0.000000  0.000000
2  0.000000   0.410747  0.000000  0.410747   0.410747  0.000000  0.410747

   involves        is  lemmatization      like      nltk  powerful  \
0  0.000000  0.324124       0.000000  0.000000  0.426184  0.426184
1  0.355173  0.000000       0.355173  0.355173  0.000000  0.000000
2  0.000000  0.312384       0.000000  0.000000  0.000000  0.000000

   preprocessing  stemming  techniques      text  tokenization      tool
0       0.000000  0.000000    0.000000  0.251711      0.000000  0.426184
1       0.000000  0.355173    0.355173  0.209771      0.355173  0.000000
2       0.410747  0.000000    0.000000  0.242594      0.000000  0.000000
```

[ ]: