CSE508_Winter2024_A1_2021337
Report

Q1.

1.1 File name : Part1_2.ipynb
Used **nltk** for removing stopwords and making word_Tokenizer
Issue 1: With word_tokenize : Words like ‘aren’t’ get separated as ‘aren’& ’t’ two different words. So, during removing stopwords, ‘t’ is not considered as stopword.
Sol : Used ‘TweetTokenizer’ to resolve it.
Rest operations are simply done by common python libraries.

1.2 5 files output before and after preprocessing

1.

loving these vintage springs on my vintage strat. they have a good tension and great stability. if you are floating your bridge and want the most out of your springs than these are the way to go.

['loving', 'these', 'vintage', 'springs', 'on', 'my', 'vintage', 'strat', '.', 'they', 'have', 'a', 'good', 'tension', 'and', 'great', 'stability', '.', 'if', 'you', 'are', 'floating', 'your', 'bridge', 'and', 'want', 'the', 'most', 'out', 'of', 'your', 'springs', 'than', 'these', 'are', 'the', 'way', 'to', 'go', '.']

['loving', 'vintage', 'springs', 'vintage', 'strat', '.', 'good', 'tension', 'great', 'stability', '.', 'floating', 'bridge', 'want', 'springs', 'way', 'go', '.']

['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']

['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']

2.

i purchased these for a cigar box guitar, they work great for that purpose. i have no complaints.
['i', 'purchased', 'these', 'for', 'a', 'cigar', 'box', 'guitar', ',', 'they', 'work', 'great', 'for', 'that', 'purpose', '.', 'i', 'have', 'no', 'complaints', '.']
['purchased', 'cigar', 'box', 'guitar', ',', 'work', 'great', 'purpose', '.', 'complaints', '.']
['purchased', 'cigar', 'box', 'guitar', 'work', 'great', 'purpose', 'complaints']
['purchased', 'cigar', 'box', 'guitar', 'work', 'great', 'purpose', 'complaints']

3.

odyssey 8-space rack is great. i put casters on mine to move it around when i need to.

['odyssey', '8', '-', 'space', 'rack', 'is', 'great', '.', 'i', 'put', 'casters', 'on', 'mine', 'to', 'move', 'it', 'around', 'when', 'i', 'need', 'to', '.']
['odyssey', '8', '-', 'space', 'rack', 'great', '.', 'put', 'casters', 'mine', 'move', 'around', 'need', '.']
['odyssey', '8', 'space', 'rack', 'great', 'put', 'casters', 'mine', 'move', 'around', 'need']
['odyssey', '8', 'space', 'rack', 'great', 'put', 'casters', 'mine', 'move', 'around', 'need']

4.

love it. great product. does exactly what i want it to do. hold up my guitar!! sturdy construction. no complaints. i love it!
['love', 'it', '.', 'great', 'product', '.', 'does', 'exactly', 'what', 'i', 'want', 'it', 'to', 'do', '.', 'hold', 'up', 'my', 'guitar', '!', '!', 'sturdy', 'construction', '.', 'no', 'complaints', '.', 'i', 'love', 'it', '!']
['love', '.', 'great', 'product', '.', 'exactly', 'want', '.', 'hold', 'guitar', '!', '!', 'sturdy', 'construction', '.', 'complaints', '.', 'love', '!']
['love', 'great', 'product', 'exactly', 'want', 'hold', 'guitar', 'sturdy', 'construction', 'complaints', 'love']
['love', 'great', 'product', 'exactly', 'want', 'hold', 'guitar', 'sturdy', 'construction', 'complaints', 'love']

5.

works, and my guitar looks like a new one!
['works', ',', 'and', 'my', 'guitar', 'looks', 'like', 'a', 'new', 'one', '!']
['works', ',', 'guitar', 'looks', 'like', 'new', 'one', '!']
['works', 'guitar', 'looks', 'like', 'new', 'one']
['works', 'guitar', 'looks', 'like', 'new', 'one']

Q2. Unigram Inverted Index and Boolean Queries

2.1

All the processed files are stored in “Processed_files”.
So, used that directory, and one be one read all the files.If found a new word, then add this as a new element to the dictionary, and the files it is present in is added as values. These values are stored in set.
If already existing token is found then just add the file ,if that file is not present.
Ex: A preprocessed data is saved as :
loving vintage springs vintage strat good tension great stability floating bridge want springs way go
So, we used **.split()** to split words and named them ‘tokens’.Then did the same process of visiting each file

2.2

Used pickle module:
Use dump and load method .
Dump method : #pickle data to a serialize format

Load method :#convert back serialise data to original datatype
On printing loaded_index : {'loving': {'file391.txt', 'file723.txt', 'file1.txt', 'file254.txt'}, 'vintage': {'file725.txt', 'file847.txt', 'file1.txt', 'file936.txt', 'file197.txt', 'file494.txt', 'file51.txt', 'file638.txt', 'file737.txt', 'file827.txt', 'file895.txt', 'file907.txt', 'file278.txt', 'file150.txt', 'file422.txt', 'file439.txt', 'file674.txt', 'file597.txt'}.....

2.3/2.4
For boolean queries:
We are using sets to perform these queries.
The set are nothing but, {files.txt} for each word present in loaded_index

```
def perform(op,set1,set2):
  if(op=='AND'):
    return set1&set2
  if(op=='OR'):
    return set1|set2
  elif (op=='AND NOT'):
    return set1-set2
  elif (op=='OR NOT'):
    return set1.symmetric_difference(set2)
  elif op =='NOT':
    all_doc=set1.union(set2)
    return all_doc-set2
  else:
    return set()
```

2.5
Input format : As I was unable to take input on google collab platform, I created a Part2.py. Taken input as query and operations. Created separate ‘N’ files for each query and stored in ‘Part2_user_input’
Then did preprocessing and stored in ‘Part2_processes_input’

2.6
Output format : File: Part1_2.ipynb
Used ‘Part2_processed_input’ folder to get processed files.

Query 1: car OR bag AND NOT canister
Number of documents retrieved for query 1: 6
Names of the documents retrieved for query 1: file542.txt, file174.txt, file746.txt, file264.txt, file166.txt, file886.txt
Query 2: coffee AND brewing OR NOT techniques OR cookbook
Number of documents retrieved for query 2: 0
Names of the documents retrieved for query 2: No documents found

Q3. Positional Index and Phrase Queries
File name:Part3.ipynb

3.1
Kept the storing format same, just introduced an array to store index of word in each file.Rest of the technique is similar to creating ‘Inverted index’.
Ex:
{'loving': {'file1.txt': [0], 'file254.txt': [28], 'file391.txt': [2], 'file723.txt': [6]}, 'vintage': {'file1.txt': [1, 3], 'file150.txt': [10], 'file197.txt': [7, 42], 'file278.txt': [4], 'file422.txt': [8], 'file439.txt': [3, 32], 'file494.txt': [10], 'file51.txt': [27], 'file597.txt': [28]....

3.2
Used again dump and load methods
3.3
Input is again taken in Part3.py
And data processed and saved in Part3_processed_input.

3.4
Output:
No. of documents retrieved for Query: car bag canister : 6
file166.txt
file542.txt
file264.txt
file174.txt
file746.txt
file886.txt

No. of documents retrieved for Query: coffee brewing techniques cookbook
: 2
file157.txt
file886.txt
Rest steps are same