Inverted-Index

October 9, 2020

1 Importing Libraries

[1]:
```
import os
import re
import json
from nltk.stem.porter import PorterStemmer
import matplotlib.pyplot as plt
```

2 Define your path

[2]:
```
myPath = "./cacm/cacm.all"
```

3 Parser

Returns dictionary named "documents". indexes are the document ids. documents[index] = {'title', 'abstract', 'date', 'authors'} We use dictionary, because we can search and get document details with O(1) complexity.

[3]:
```
def parser(myPath = "./cacm/cacm.all"):
    documents = {}
    my_file = open(myPath,'r',encoding='windows-1252')
    while True:
        line = my_file.readline()
        if not line:
            break
        elif line.startswith('.I'):
            mode = 'i'
            index = line.split(' ')[-1:][0]
            index = index.replace('\n', '')
        elif line.startswith('.T'):
            mode = 't'
        elif line.startswith('.W'):
            mode = 'w'
        elif line.startswith('.B'):
            mode = 'b'
        elif line.startswith('.A'):
            mode = 'a'
        elif line.startswith('.'):
            mode = 'z'
        else:
            if mode=='t':
                documents[index] = {'title': line.replace('\n', ''), 'abstract':
↪ '', 'date': '', 'authors': ''}
            elif mode=='w':
                documents[index]['abstract'] += line.replace('\n', '')
            elif mode=='b':
                documents[index]['date'] = line.replace('\n', '')
            elif mode=='a':
                documents[index]['authors'] += line.replace('\n', '')
            elif mode=='z':
                continue
    my_file.close()
    return documents
```

4 Preprocessing

4.1 Tokenizing

4.2 Removing stopwords

4.3 Stemming

[4]:
```
def preprocessing(documents):
    # Tokenizing using list comprehension and regular expressions
    tokenizing = [re.findall('\w+',documents[documentId]["terms"]) for␣
↪documentId in documents]
    token_length = 0
    for doc in tokenizing:
        token_length+=len(doc)
    print(f'Number of tokens before preprocessing: {token_length}')
    # Open stopwords file
    stop_words = open("./stopwords.txt",'r',encoding='windows-1252')
    stop_words = stop_words.read()
    stop_words = stop_words.split()
    stopWords_removed = []
    stems = []
    finished_dic = {}
    # Remove stopwords using a list comprehension
    for doc in tokenizing:
        doc = [d for d in doc if d not in stop_words]
        stopWords_removed.append(doc)

    stopWord_length = 0
    for doc in stopWords_removed:
        stopWord_length += len(doc)
    print(f'Number of tokens after removing stop words: {stopWord_length}')
    # Using Porter Stemmer algorithm
    porter = PorterStemmer()
    for doc in stopWords_removed:
        doc = [porter.stem(s) for s in doc]
        stems.append(doc)

    stem_length = 0
    for doc in stems:
        stem_length+=len(doc)
    print(f'Number of tokens after stemming: {stem_length}')

    for documentId in documents:
        finished_dic[documentId] = {'terms': ''}
        finished_dic[documentId]['terms'] = ' '.join(stems[int(documentId)-1])

    return finished_dic
```

5 Create inverted index

returns dictionary inverted_index. #### Inverted_index: word : {'doc_id' : documentId_list, 'token_id' : token_id, 'tf' : term_frequency}

[5]:
```
def create_index(preprocessed):
    inverted_index={}
    token_id = 1
    for documentId, text in preprocessed.items():
        for word in text['terms'].lower().split():
            # If the term is in dictionary
            if inverted_index.get(word,False):
                # Add term frequency
                inverted_index[word]['tf']+=1
                if documentId not in inverted_index[word]['doc_id']:
                    inverted_index[word]['doc_id'].append(documentId)
            else:
                # If the term is not in inverted_index dictionary
                inverted_index[word]={'doc_id':[documentId], 'token_id':
 ↪token_id, 'tf':1}
                token_id+=1

    return inverted_index
```

6 Parsing dataset

[6]:
```
documents = parser(myPath)
```

6.1 Merge title and abstract parts into new "terms" value

[7]:
```
for docId in documents:
    documents[docId]['terms'] = documents[docId]['title'] +␣
 ↪documents[docId]['abstract']
```

7 Preprocessing and printing information about tokens

[8]:
```
preprocessed = preprocessing(documents)
```

```
Number of tokens before preprocessing: 161485
Number of tokens after removing stop words: 113749
Number of tokens after stemming: 113749
```

8 Printing number of documents

[9]:
```
doc_count = len(preprocessed)
print(f'Number of preprocessed documents: {doc_count}')
```

```
Number of preprocessed documents: 3204
```

9 Create inverted-index

[10]:
```
inverted_index = create_index(preprocessed)
print(f'Number of words in inverted-index: {len(inverted_index)}')
```

```
Number of words in inverted-index: 13351
```

10 Main program

[ ]:
```
import time
durations = []
while True:
    term = input('please Enter the term: ')
    # Keep start time in mind
    start_time = time.time()
    # If user enters "ZZEND", we break the loop and end the program
    if term == 'ZZEND':
        break
    print(f'Document frequency: {len(inverted_index[term]["doc_id"])}')
    for t in inverted_index[term]['doc_id']:
        # print document id
        print(f'Document id: {t}')
        # print document title
        print(f'Title: {documents[t]["title"]}')
        # print term frequency in the document
        print(f'Term frequency: {len(re.
↪findall(term,preprocessed[t]["terms"]))}')
        # Find out the occurrences of term in the document
        occurrences = [i.start() for i in re.finditer(term,␣
↪preprocessed[t]['terms'])]
        print('occurrences: ', occurrences)
        # Creat document summary with 8 words in it's context
        doc = preprocessed[t]['terms']
        doc = doc.replace(term, term.upper())
        doc = doc.split(' ')
        start = doc.index(term.upper())
        summary = ''
        try:
            for i in range(start, start+8):
                summary += doc[i] + ' '
        except:
            pass
        print(f'Document summary: {summary}')
        # Print query execution time;
        # append duration to durations list
        durations.append(time.time()-start_time)
        print('Query execution time: ',time.time()-start_time)
sum = 0
# Find average execution time based on durations list
try:
    for duration in durations:
        sum += duration
    average_time = sum / len(durations)
    print('Average query execution time: ', average_time)
except:
    pass
```

11 How to run the program?

Unzip "IR_InvertedIndex.zip", open project folder, then run the program by following command:

python IR_InvertedIndex.py