7071CEM Information Retrieval

Student name: CHRISTY ABRAHAM JACOB

Table of Contents

Task 1. Search Engine

In this task I will develop a search engine that crawls through the publications in School of computing, Electronics and Maths [Coventry university] and fetches the most relevant document that matches the search query given by the end user.

Below described are the steps and procedures taken to develop the search engine.

1. Steps to crawl and store the data fetched during web page crawl

Step 1: Reading "robots.txt" file of School of CEM to fetch the crawl delay and the links that are disallowed to crawl

```
11    def fetch_crawl_delay():
12        import urllib.robotparser
13        robo_praser = urllib.robotparser.RobotFileParser()
14        robo_praser.set_url("https://pureportal.coventry.ac.uk/robots.txt")
15
16        robo_praser.read()
17
18        crwl_delay = robo_praser.crawl_delay("*")
19        print('Crawl_Delay :',crwl_delay, '\n')
```

```
In [20]: fetch_crawl_delay()
Crawl_Delay : 1
```

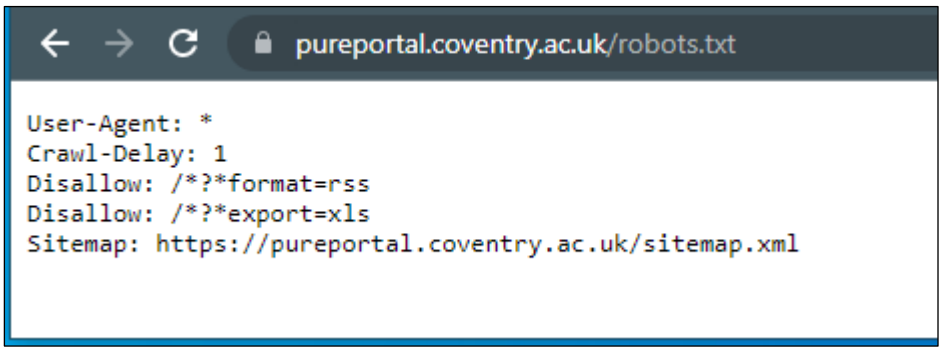
Above code parses the content in "robots.txt" file and assigns it to the variable named "crwl_delay" initialised in the main method.

At the start of the main method, system time is fetched in-order to calculate the interval between the requests made to each url in the code.

```
295    if __name__ == "__main__":
296
297        from datetime import datetime
298        import time
299
300        outer_start = datetime.now()
301        print("Started Timer at :",outer_start)
302        crwl_delay = 0
303
304        fetch_crawl_delay()
305        print("************************************************************")
```

```
253                if((datetime.now() - inner_start).seconds < crwl_delay):
254                    print("** Waiting for 1 second for the next request call")
255                    time.sleep(1)
```

Step 2: Defining a function to verify the links used to crawl the website



From `https://pureportal.coventry.ac.uk/robots.txt` we can notice that 2 links that contains /*?*format=rss and /*?*export=xls are disallowed. In-order to obey this rule, we need to create a function or develop a regular expression in python that verifies above url formats.

```
307         patterns = [r'/*[?]+export=xls',r'/*[?]+format=rss']
```

In the above regular expression, '*' verifies zero or more occurrences of the pattern towards the left, '+' verifies one or more occurrences of the pattern and, characters that needs to be verified are defined inside a pair of '[]' as shown in the above code snippet. (w3schools, n.d.)

This pattern along with the url is passed to the function defined to verify before the code accesses any url in the script.

```
30      def link_verify(patterns,url):
31          import re
32          for pattern in patterns:
33              try:
34                  re.compile(pattern)
35                  # print(f"'{pattern}' is a VALID regex pattern")
36                  if re.search(pattern, url):
37                      print(f"  '{url}' MATCHES the pattern",pattern)
38                      return True
39                  else:
40                      print(f"  '{url}' DOES NOT match the pattern",pattern)
41
42              except re.error:
43                  print(f"'{pattern}' is NOT a valid regex pattern")
44
45          return False
```

```
In [18]: link_verify(patterns,"www.example.com/?format=rss")
  'www.example.com/?format=rss' DOES NOT match the pattern /*[?]+export=xls
  'www.example.com/?format=rss' MATCHES the pattern /*[?]+format=rss
Out[18]: True
```

```
In [19]: link_verify(patterns,"www.example.com/?for=rss")
  'www.example.com/?for=rss' DOES NOT match the pattern /*[?]+export=xls
  'www.example.com/?for=rss' DOES NOT match the pattern /*[?]+format=rss
Out[19]: False
```

Step 3: Fetching data from the database or file that was already crawled for the same page

Script looks for a file named “pub_data.json” in the current working directory and uses it to compare and update with the new data crawled from the websites. Only changes or new keys(data) will be updated to the file during web crawling.

```
57      def read_file(fileName):
58          import json as js
59          try:
60              #trying to open a file in read mode
61              o = open(fileName,"r")
62
63              # returns JSON object as a dictionary
64              file = js.load(o)
65              o.close()
66              print("File found and read")
67          except FileNotFoundError:
68              print("File does not exist")
69              file = {}
70
71          return file
```

```
316         pub_data_retrv = {}
317         pub_data_retrv = read_file('pub_data.json')
```

```
In [23]: pub_data_retrv = {}
    ...: pub_data_retrv = read_file('pub_data.json')
File found and read
```

If file is not found, script starts recording all the data in the dictionary declared [pub_data_retrv] and writes the dictionary to a .json file at the end of the crawling.

Step 4: Fetching profile names and links of all the profile that belongs to school of CEM

In the main program I declare the variables that is required to store the profile data. In this script I have used a dictionary to store the profile and link as a key-value pair for future reference in the script and have declared an empty set to store profile links so that the script doesn’t store duplicate links while crawling.

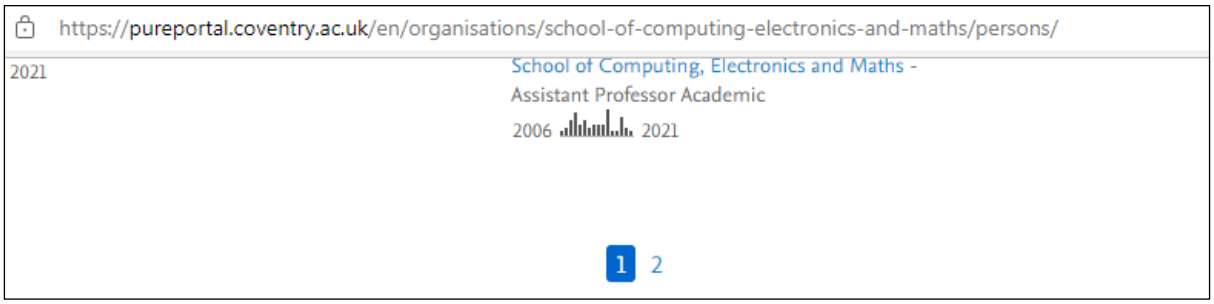
```
321     prof_data={}
322     prof_links = set()
323     prof_url = "https://pureportal.coventry.ac.uk/en/organisations/school-of-computing-electronics-and-maths/persons/"
```

To fetch the page content and parse it we can use a python library named “BeautifulSoup” and using the object created for beautifulSoup we can access different html tags to fetch the desired content.

obj.SELECT() method is used instead of obj.FINDALL() to fetch the content, because select() method uses css path to identify the content and is much efficient and accurate when compared to findAll() method.

```
104    def fetch_profiles():
105        # import bs4
106        from bs4 import BeautifulSoup
107        import requests
108        if(link_verify(patterns,prof_url) == False):
109            prof_page = requests.get(prof_url)
110            # print(page.text)
111
112            prof_soup = BeautifulSoup(prof_page.text, "html.parser")
113            # print(soup)
114
115            # Fetching the total page count
116            prof_page_count = int(prof_soup.select("nav.pages>ul>li:nth-last-child(2) a.step")[0].text)
117            print('Total pages to crawl :',prof_page_count)
```

```
In [27]:
    ...: prof_page_count = int(prof_soup.select("nav.pages>ul>li:nth-last-child(2)
a.step")[0].text)
    ...: print('Total pages to crawl :',prof_page_count)
Total pages to crawl : 2
```



In the above code snippet, css path is used to identify the total pages in profile url and use that value to navigate from one page to another during crawling. This approach avoids hardcoding the values and the code works efficiently even if the page count is updated on the website in future.

From the page count value obtained, we can iterate through the pages to fetch the profile links and name. Profile link is used as the key and name is stored as its value.

```
119            # Iterating through each page to fetch the Profile details
120            count=0
121            for i in range (0, prof_page_count): # page count
122
123                current_page_url = prof_url+'?page='+str(i)
124                print("************************************************")
125                print("Profile info from Page : ",i+1)
126                print(current_page_url)
127                print("************************************************")
128
129                if(link_verify(patterns,current_page_url)== False):
130                    page =  requests.get(current_page_url)
131                    soup = BeautifulSoup(page.text, "html.parser")
132                    profiles = soup.select("ul.grid-results h3.title a.link.person")
133                    for profile in profiles:
134                        # Fetching Profile link
135                        prof_links.add(profile.get('href'))
136                        prof_data.update({profile.get('href'):{}})
137
138                        # Fetching name of the profile
139                        print(profile.text)
140                        prof_data[profile.get('href')].update({'Name':profile.text})
141                count += 1
142            print('\n')
143
144        print("No. of pages crawled: ", count)
145        print("No. of profiles fetched :", len(prof_data))
```

```
In [30]: print("No. of pages crawled: ", count)
    ...: print("No. of profiles fetched :", len(prof_data))
No. of pages crawled:  2
No. of profiles fetched : 59
```

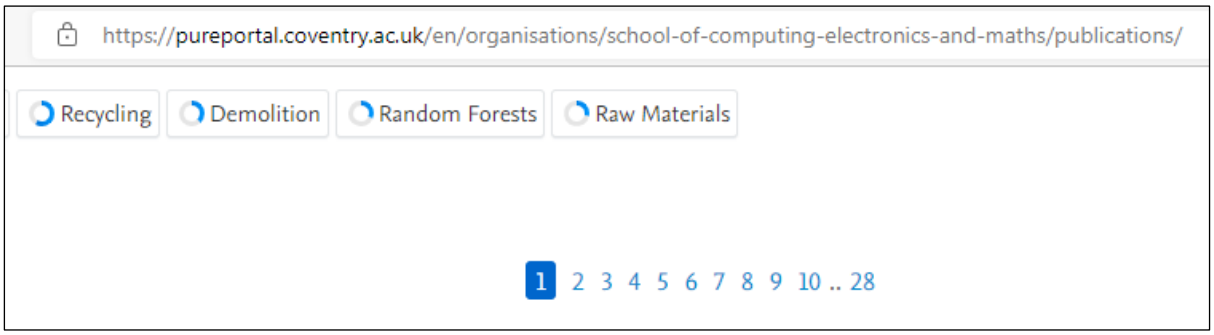
Step 5: Fetching publication details in which at least one co-author is a current faculty of school of CEM

In the next step, script fetches the total page count for publication to navigate to different publication pages without hardcoding the page count. First, I define the variables and url that's required to execute the publication crawl method [crawl_and_update] created in the script.

```
pub_skipped = set()
pub_without_abtrt = set()
valid_titles ={}
title =''
url = "https://pureportal.coventry.ac.uk/en/organisations/school-of-computing-electronics-and-maths/publications/"

crawl_and_update()
print("*****************************************************")
```

```
159  def crawl_and_update():
160      import re
161      from bs4 import BeautifulSoup
162      import requests
163      if(link_verify(patterns,url)== False):
164          page = requests.get(url)
165          # print(page.text)
166
167          soup = BeautifulSoup(page.text, "html.parser")
168          # print(soup)
169
170          # Fetching the total page count
171          page_count = int(soup.select("nav.pages>ul>li:nth-last-child(2) a.step")[0].text)
172          print('Total pages to crawl :',page_count)
```

```
In [34]: page_count = int(soup.select("nav.pages>ul>li:nth-last-child(2) a.step")
[0].text)
    ...: print('Total pages to crawl :',page_count)
Total pages to crawl : 28
```



Script fetches publication link first in-order to navigate to the publication page and fetch other details like authors, author profiles, publication title, publication year and abstract.

```
174         # Iterating through each page to fet the Publication details
175         count = 0
176         pub_id = 1
177         for i in range (0, page_count): # page count
178 
179             inner_start = datetime.now()
180 
181             current_page_url = url+'?page='+str(i)
182             print("************************************************")
183             print("Publication info from Page : ",i+1)
184             print(current_page_url)
185             print("************************************************")
186 
187             if(link_verify(patterns,current_page_url)== False):
188                 page =  requests.get(current_page_url)
189                 soup = BeautifulSoup(page.text, "html.parser")
190                 publications = soup.select("ul.list-results h3.title a.link")
```

Once all the publication links are fetched from the current page, script iterates through each publication link as shown in the below code snippet.

```
191     for publication in publications:
192         print(pub_id)
193 
194         # --Title
195         # print('--Title :',publication.text)
196         title = re.sub('\W+','_', publication.text)
197         update_key(title)
198         # print(title, publication.text)
199         update_values('Title', publication.text, title)
200 
201         # --Publication link
202         # print('--Publication link:',publication.get('href'))
203         update_values('Publication_link', publication.get('href'),title)
```

As the code to store and compare the values fetched is generic, its best practice to define a separate function that fetches and compares with the values that is fetched from the .json file or database.

update_key() method defined is used to check the existence of same keys/publication in the database. If the publication fails to match with any of the key fetched from the file, then a separate key is created for new value fetched.

```
79  def update_key(key):
80 
81      if(key in pub_data_retrv.keys()):
82          print("Key already exists")
83      else:
84          pub_data_retrv.update({key:{}})
85          print("New Key created")
```

```
In [44]: update_key(title)
Key already exists
```

Similarly, update_value() verifies the values for key passed as argument when the function is called.

```
87  def update_values(key, value, title):
88      if(key in pub_data_retrv[title].keys() and (pub_data_retrv[title][key]) != value):
89          pub_data_retrv[title].update({key:value})
90          print("Updated", 'value for the key: [',key,'] with title', title )
91
92      elif(key in pub_data_retrv[title].keys() and (pub_data_retrv[title][key]) == value):
93          print("Value is the same as the one fetched from file")
94          # a='' # do nothing!!!!!!!!!!!!!!!!!!!
95
96      elif(key not in pub_data_retrv[title].keys()):
97          pub_data_retrv[title].update({key:value})
98          print("New key created within the title key:",title)
```

```
In [46]: update_values('Title', publication.text, title)
Value is the same as the one fetched from file
```

Next, script verifies if the publication link obeys the “robots.txt” file conditions. If it obeys, other details like publication year and author names are fetched.

```
207     if(link_verify(patterns,publication.get('href'))== False):
208         pub_page = requests.get(publication.get('href'))
209         pub_soup = BeautifulSoup(pub_page.text, "html.parser")
210
211         # --Publication Year
212         Pub_year = pub_soup.select("tr.status td span.date")[0].text.split(' ')[-1]
213         # print("--Publication Year :", Pub_year)
214         update_values('Publication_Year', Pub_year, title)
215
216         # --Author Names
217         Author_names = pub_soup.select("p.relations.persons")[0].text
218         # print("--Authors:", Author_names)
219         update_values('Authors', Author_names.split(','), title)
```

As there can be multiple profile links for a single publication, count of the profile links in a particular publication is fetched first and then the links are compared with the links fetched from the profile page to identify if the publication has a co-author who is currently a faculty of school of CEM.

If the conditions are not satisfied, id/key of the publication is stored in “pub_skipped” variable for future references.

```
221     # --Profile Links
222     # print("--Profile Links:")
223     temp_pub_links = []
224     if(len(pub_soup.select("div.introduction.no-metrics p a.link.person")) == 0):
225         print("** None of the co-authors is a member of the school CEM")
226         pub_skipped.add(str(pub_id)+'--'+title)
227     else:
228         author_count = len(pub_soup.select("div.introduction.no-metrics p a.link.person"))
229         pub_valid = False
230         for author in pub_soup.select("div.introduction.no-metrics p a.link.person"):
231             # print(author.text, ':', author.get('href'))
232             temp_pub_links.append(author.get('href'))
233             if(pub_valid == False and (author.get('href') in prof_links)):
234                 pub_valid = True
235         if(pub_valid == False):
236             print("*** None of the authors have profile link from CEM ")
237             pub_skipped.add(str(pub_id)+'--'+title)
238         else:
239             valid_titles.update({pub_id:title})
240         update_values('Profile_Links', temp_pub_links, title)
```

Finally, the abstract is fetched and verified with the existing data in the database/file. If publication doesn't have an abstract, id/key is stored in "pub_without_abtrt" variable as we did for to verify the profile links.

```
242     # --Abstract
243     # print("--Abstract:")
244     if(len(pub_soup.select("div.textblock")) == 0):
245         print("** This publication does not have an Abstract", '\n')
246         pub_without_abtrt.add(str(pub_id)+'--'+title)
247     else:
248         # print(pub_soup.select("div.textblock")[0].text)
249         update_values('Abstract', pub_soup.select("div.textblock")[0].text, title)
```

At the end of each "for" loop, script verifies the if the crawl delay specified in the "robots.txt" file is obeyed before hitting/sending GET request to the next url, as shown in the below code snippet.

```
251             if((datetime.now() - inner_start).seconds < crwl_delay):
252                 print("** Waiting for 1 second to load next publication link")
253                 time.sleep(1)
254 
255             pub_id += 1
256             # print("\n")
257     count += 1
258 
259     if((datetime.now() - inner_start).seconds < crwl_delay):
260         print("** Waiting for 1 second to navigate to next page")
261         time.sleep(1)
```

Step 6: Updating/Storing the information fetched to database

Once the crawl_and_update function is executed, control is passed back to the main program. Next task of the script is to store the values fetched. For this course work, I have considered json file as my database, values are fetched and updated in the same filename as shown below.

```
273 def write_file(filename, data_dict):
274     import json as js
275     # create json object from dictionary
276     js_obj = js.dumps(data_dict)
277 
278     # open file for WRITING, "w"
279     file_upd = open(filename,"w")
280 
281     # write json object to file
282     file_upd.write(js_obj)
283 
284     # close file
285     # file_upd.close()
286 
287     print("File written to local disk succesfully")
288 
289     return file_upd
```

```
357 wrt_obj = write_file("pub_data.json",pub_data_retrv_write)
358 wrt_obj.close()
```

```
In [54]: wrt_obj = write_file("pub_data.json",pub_data_retrv_write)
    ...: wrt_obj.close()
File written to local disk succesfully
```

2. Steps to process the query and find the most relevant publication

These steps are saved in a separate script, in-order to make the crawling and GUI as two separate tasks which will be more user friendly and easier to schedule the crawling process which is discussed in the next section of this task.

Step 1: Reading the crawl data and inverted index data from the database/file

Main method of this section starts by reading the data from the crawling and inverted index files updated/created during previous runs. This script uses the same read and write methods from section 1 script, which helps us to generalise the code.

```
384    if __name__ == "__main__":
385
386        from datetime import datetime
387        from Task1SearchEngineCrawler import read_file, write_file
```

```
345        outer_start = datetime.now()
346        # print("Started Timer at :",outer_start)
347
348        pub_data_retrv = {}
349        pub_data_retrv = read_file('pub_data.json')
350
351        inverted_index = {}
352        inverted_index = read_file('index_data.json')
```

```
In [3]: pub_data_retrv = read_file('pub_data.json')
File found and read
```

```
In [4]: inverted_index = read_file('index_data.json')
File found and read
```

After reading the data from the pub_data.json file, **inverted index is updated**, so that any new values in the pub_data will be added/ created in the index_data.json file

```
354    # ================================================================================
355    #     # Updating inverted index
356    # ================================================================================
357        p_docs=[]
358
359        for key in pub_data_retrv.keys():
360            temp_string = ''
361            temp_string += key.replace('_',' ').lower()+' ' #adding title
362            if(verify_key('Authors',pub_data_retrv[key])):
363                temp_string += (''.join(pub_data_retrv[key]['Authors'])).lower()+' '
364            else:
365                temp_string += ''
366            if(verify_key('Abstract',pub_data_retrv[key])):
367                temp_string += (pub_data_retrv[key]['Abstract']).lower()
368            else:
369                temp_string += ''
370            p_docs.append(temp_string)
371
372        print("Docs created for creating inverted_index")
373        inverted_index = {key: set(value) for key, value in inverted_index.items()}
374        inverted_index_func(p_docs,inverted_index)
375        inverted_index = {key: list(value) for key, value in inverted_index.items()}
376        write_file('index_data.json', inverted_index)
377        inverted_index = {key: set(value) for key, value in inverted_index.items()}
378
379        #Opening GUI
380        tkinter_GUI(inverted_index)
```

Step 2: Creating the GUI function which is responsible for the query fetching and processing

Here I am using a library named “tkinter” to create a GUI for the search engine. All the components should be created within the mainloop() of the tkinter function. Therefore, the first line of code within this function will to create a tkinter object and the last line of code will be to call the mainloop() function of tkinter.

```
208    def tkinter_GUI(inverted_index):
209        import tkinter as tk
210        from PIL import Image, ImageTk
211        from tkinter.scrolledtext import ScrolledText
212        from tkinter import END
213        import re
214
215        root = tk.Tk()
216        root.title(' Seacrh Engine || Publications of School of CEM || - Christy Jacob')
```

Step 3: Creating a basic GUI for the search engine

Before adding any GUI element, first step is to create a canvas to place the GUI components.

```
218        canvas = tk.Canvas(root, width=1100, height=600)
219        canvas.grid(columnspan=3, rowspan=7)
```

Next, I am adding the Coventry logo to the GUI. It is always better to store the image in the same directory as that of the script file.

```
# Inserting logo
logo = Image.open('coventry-uni-logo.png')
logo = ImageTk.PhotoImage(logo)
logo_label = tk.Label(image=logo)
logo_label.image = logo
logo_label.grid(column=1, row=0)
```

To make the GUI more attractive and user friendly, we can also add text and text_boxes to enter the query and display the results.

```
search_inst = tk.Label(root, text="Enter your Search Query here")
search_inst.grid(columnspan=3, column=0, row=1)

text_box1 = tk.Text(root, height=1, width=80, padx=15, pady=15)
text_box1.grid(columnspan=3, column=0, row=2)
```

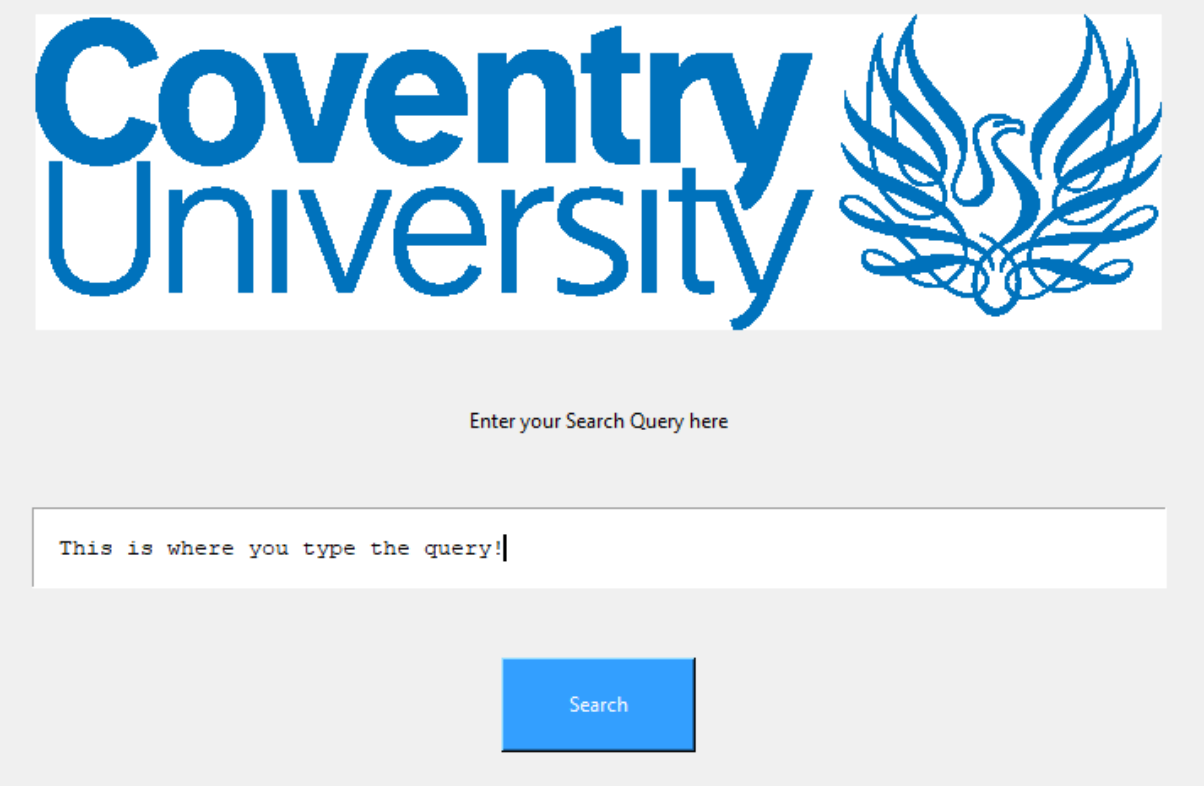
ScrolledText() method is used to create a text box instead of Text() function because, ScrolledText() automatically creates a scroll bar if the content exceeds the text_box line count.

```
skrl_text = ScrolledText(root)
skrl_text.config(width = 135, height = 18)
```

Search button is responsible for the execution the search_query() method which calls other processing methods to display the most relevant doc.

```
# Search button
search_text = tk.StringVar()
search_btn = tk.Button(root,
                       command=lambda:search_query(pub_data_retrv,
                                                   inverted_index),
                       textvariable = search_text, bg="#339FFF",
                       fg="white", height=3, width=15)
search_text.set("Search")
search_btn.grid(column=1, row=3)
```

Below displayed is a screenshot of the GUI of the search engine before starting the search.



Step 4: Processing the crawled data to construct inverted index

As the script have already collected the existing crawl data from the database, we need to select only the required data/columns for pre-processing. Here I am selecting tile, author names and abstract to create an inverted indexed matrix.

```
226     docs=[]
227
228     for index in posting_list_indexes:
229         temp_string = ''
230         temp_string += (list(pub_data_retrv.keys())[index]).replace('_',' ').lower()+' '
231         if(verify_key('Authors',list(pub_data_retrv.keys())[index])):
232             temp_string += (''.join(list(pub_data_retrv.keys())[index]['Authors'])).lower
233         else:
234             temp_string += ''
235         if(verify_key('Abstract',list(pub_data_retrv.keys())[index])):
236             temp_string += (list(pub_data_retrv.keys())[index]['Abstract']).lower()
237         else:
238             temp_string += ''
239         docs.append(temp_string)
240
241     print("Docs created")
```

```
In [15]: docs=[]
    ...:
    ...: for index in posting_list_indexes:
    ...:     temp_string = ''
    ...:     temp_string += (list(pub_data_retrv.keys())[index]).replace('_',' ').lower()+' ' #adding
title
    ...:     if(verify_key('Authors',list(pub_data_retrv.keys())[index])):
    ...:         temp_string += (''.join(list(pub_data_retrv.keys())[index]['Authors'])).lower()+' '
    ...:     else:
    ...:         temp_string += ''
    ...:     if(verify_key('Abstract',list(pub_data_retrv.keys())[index])):
    ...:         temp_string += (list(pub_data_retrv.keys())[index]['Abstract']).lower()
    ...:     else:
    ...:         temp_string += ''
    ...:     docs.append(temp_string)
    ...:
    ...: print("Docs created")
Docs created
```

As abstract can be null, we need to verify if the key exists in the database for that publication, in-order to avoid errors.

```
12  # function to check if key exists in the given dictionary
13  def verify_key(key, dict_name):
14      if key in dict_name:
15          return True
16      else:
17          return False
```

Values of column title, author names and abstract are combined to form a single document for each publication as shown in the above code snippet. These docs must be pre-processed before creating inverted index. In pre-processing, each doc is tokenized, stemmed, and stop words are removed. First, we must download the packages required for stop words, stemming, and tokenizing and create objects to access stemming and stop word methods.

```
19  def pre_processing(docs,filtered_docs):
20      import re
21
22      # Tokenizing, removing stop words and stemming:
23
24      import nltk
25      nltk.download("stopwords")
26      from nltk.corpus import stopwords
27      nltk.download("punkt")
28      from nltk.tokenize import word_tokenize
29      from nltk.stem import PorterStemmer
30      sw = stopwords.words('english')
31      ps = PorterStemmer()
```

Next, the function tokenizes and removes all the stop words present in the stop word object named “sw” [english].

pre_processing() method returns the filtered doc and stored in another variable which is passed as one of the arguments to calculate inverted index.

```
37      temp_filtered_docs = []
38      for doc in docs:
39          doc = re.sub('\W+',' ', doc) #Removing special characters
40          tokens = word_tokenize(doc)
41          tmp = ""
42          for w in tokens:
43              if w not in sw:
44                  tmp += (ps.stem(w) + " ").lower()
45          temp_filtered_docs.append(tmp)
46
47      filtered_docs += temp_filtered_docs #problem
48
49      # filtered_docs = temp_filtered_docs
50      return filtered_docs
```

```
277         crawl_filtered_docs = []
278         crawl_filtered_docs = pre_processing(docs,crawl_filtered_docs)
```

Step 5: Creating inverted index matrix

I have made used of the enumerate () method in the inverted_index_func() function to keep track of the iteration count and the value at the same time. Filtered doc is split into single words and compared it with the data collected from the database/file. If the term is present in the file, posting list is updated with the new doc's index, if the term is not present in the file, a new key is created for that term.

```
81      def inverted_index_func(crawl_filtered_docs,inverted_index):
82
83          for i, doc in enumerate(crawl_filtered_docs):
84              for term in doc.split():
85                  if term in inverted_index:
86                      inverted_index[term].add(i)
87                  else:
88                      inverted_index[term] = {i}
89
90          # print(inverted_index)
```

```
245     inverted_index_func(crawl_filtered_docs,inverted_index)
```

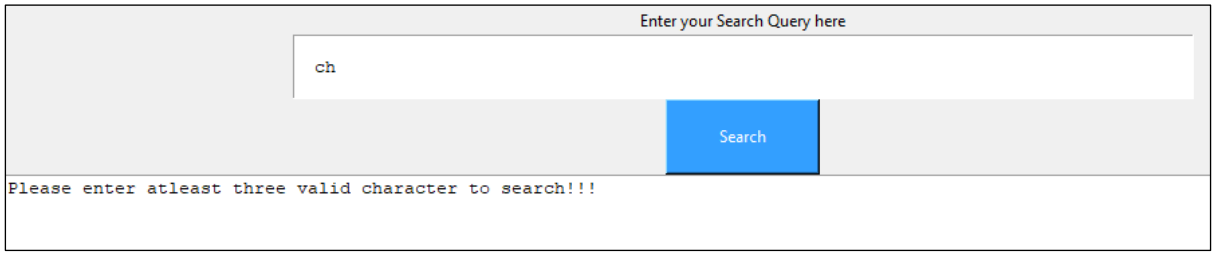
Updated inverted index is updated/written back to the database/file.

```
247     inverted_index = {key: list(value) for key, value in inverted_index.items()}
248     write_file('index_data.json', inverted_index)
```

Step 6: Fetching the query and pre-processing the query

Query is fetched from the text box and checked if it contains at least 3 valid characters other than special characters. If the query doesn't satisfy the condition a warning message is displayed, asking the user to enter valid character count.

```
252     query = text_box1.get("1.0","end-1c")
253
254     if(len(re.sub('\W+','', query)) < 3):
255         print("Please enter atleast three valid character to search!!!")
256         skrl_text.insert(END, "Please enter atleast three valid character to search!!!"+ "\n")
257         skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)
258         quit()
```

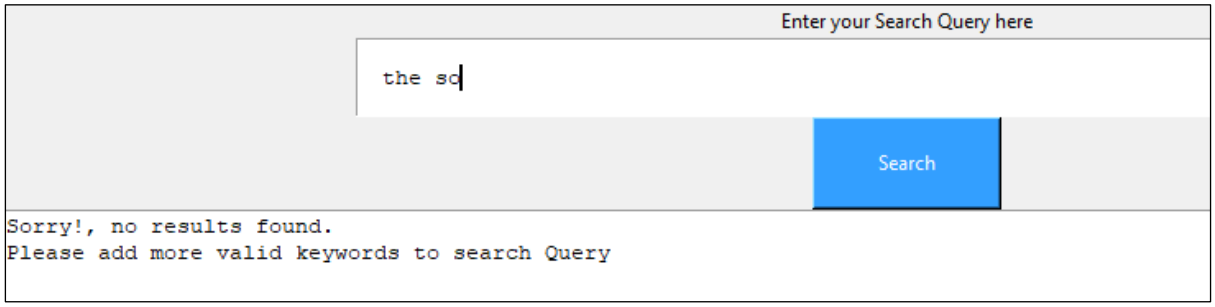


If the query is valid, it is pre-processed to find the posting list indexes.

```
262     # PRE-PROCESSING the query
263     query_filtered_docs = pre_processing_query([query])
264     # print(query_filtered_docs)
265
266     filtered_indexed_docs = []
267     posting_list_indexes = find_posting_list(query_filtered_docs,inverted_index )
```

Next, the script searches if the pre-processed query has a posting list in the inverted index dictionary created. If it doesn't find the posting list, another warning message is displayed, asking user to enter more keywords to get results.

```
269     if(len(posting_list_indexes) == 0):
270         skrl_text.insert(END, "Sorry!, no results found."+"\n")
271         skrl_text.insert(END, "Please add more valid keywords to search Query"+"\n")
272         skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)
```



If the query is valid, it adds the filtered query with the filtered data fetched with respect to the posting list.

```
274     else:
275         posting_list_indexes = list(posting_list_indexes)
276
277         for index in posting_list_indexes:
278             filtered_indexed_docs.append(crawl_filtered_docs[index])
279
280         filtered_indexed_docs += query_filtered_docs
281         print("Length of new filtered docs with Query :",len(filtered_indexed_docs))
```

Step 7: Performing Ranked retrieval using Vector Space Model

Filtered docs are converted to vectors using TfidfVectorizer library and the vector matrix is converted to list to calculate the rankings (cosine values) of each document.

```
In [16]: query = 'marwan fuad'
```

```
112     def vector_space(filtered_docs):
113         from sklearn.feature_extraction.text import TfidfVectorizer
114         vectorizer = TfidfVectorizer()
115         X = vectorizer.fit_transform(filtered_docs)
116         return X.todense()
```

```
289     # Creating vectors for each doc
290     vector_matrix = vector_space(filtered_indexed_docs)
291     print(vector_matrix.shape)
292     rows, columns = vector_matrix.shape
293     print("rows :", rows, " columns :", columns)
```

```
In [23]: print("rows :", rows, " columns :", columns)
rows : 10  columns : 307
```

```
136     def convert_matrix_list(matrix):
137         import numpy as np
138         np_matrix = matrix.flatten()
139         np_array = np.squeeze(np.asarray(np_matrix))
140         samp_list = np_array.tolist()
141         return samp_list
```

```
295     vector_lists_query = []
296     for matrix in vector_matrix:
297         vector_lists_query.append(convert_matrix_list(matrix))
```

```
In [25]: len(vector_lists_query)
Out[25]: 10
```

The rankings are then converted to a dataframe and sorted in descending order to display the search results, so that the most relevant documents (having greater cosine values) are listed first.

```
118     def inner(vec1, vec2):
119         prod = 0
120         for i in range(len(vec1)):
121             prod += vec1[i] * vec2[i]
122         return prod
123 
124     def length(vec):
125         import math
126         return math.sqrt(inner(vec, vec))
127 
128     def cosine(vec1, vec2):
129         len1 = length(vec1)
130         len2 = length(vec2)
131         prod = inner(vec1, vec2)
132         return prod / (len1 * len2)
```

```
144 def vector_space_model(vector_lists_query):
145 
146     # calculate the cosine values to rank the documents
147     rank_values = []
148     list_titles = []
149     for i in range (0,(len(vector_lists_query)-1)):
150         list_titles.append(list(pub_data_retrv.keys())[i])
151         rank_values.append(cosine(vector_lists_query[(len(vector_lists_query)-1)],
152                                   vector_lists_query[i]))
153 
154     print("rank_values length:", len(rank_values))
155     print("list_titles length:", len(list_titles))
156 
157     return rank_values,list_titles
```

```
299     rank_values, list_titles = vector_space_model(vector_lists_query)
```

```
In [26]: rank_values, list_titles =
vector_space_model(vector_lists_query)
rank_values length: 9
list_titles length: 9
```

Creating and sorting pandas dataframe

```
160     def sort_rank(rank_values, list_titles, posting_list_indexes):
161         import pandas as pd
162         rank_df = pd.DataFrame({
163             'Index' : posting_list_indexes,
164             'Title' : list_titles,
165             'Rank' : rank_values
166             })
167 
168         rank_df = rank_df.sort_values(by='Rank', ascending=False)
169         rank_df = rank_df.reset_index(drop=True)
170 
171         return rank_df
```

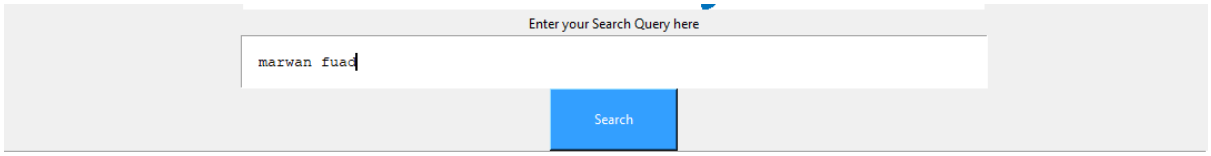
```
300     # Storing the ranks of the document
301     rank_df = sort_rank(rank_values, list_titles, posting_list_indexes)
```

Step 8: Displaying the results

Result is displayed in the scroll text field of tkinter

```
303     # display result
304     print("------------------------------------------------ \n")
305     print("Searh query :",query)
306     print("Search results \n")
307     print("Total execution time :", (datetime.now() - outer_start), '\n')
308     skrl_text.insert(END, "Search results:"+ "\n")
309     skrl_text.insert(END, "\n")
310     skrl_text.insert(END, "Total execution time :"+ str((datetime.now() - outer_start))+"\n")
311     skrl_text.insert(END, "\n")
```

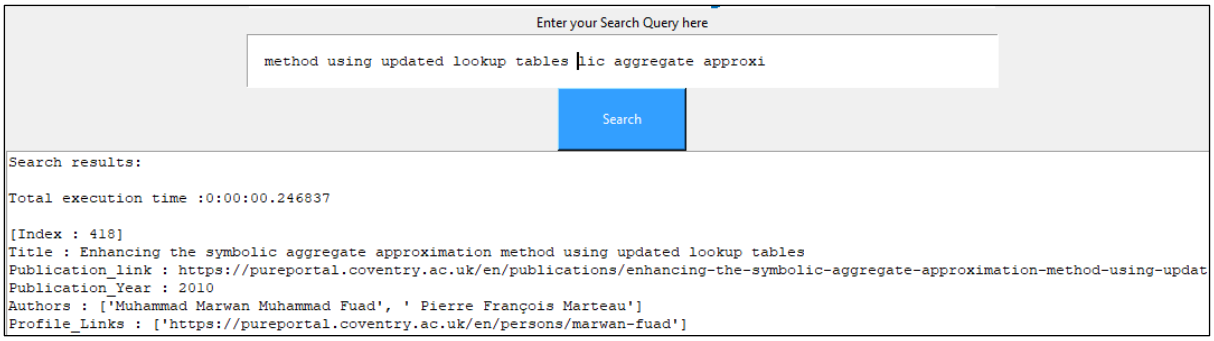
```
312     for i in list(rank_df['Index']):
313         print(list(pub_data_retrv.values())[i]);
314         skrl_text.insert(END, "[Index : "+str(i)+"]"+"\n")
315         skrl_text.insert(END,
316              'Title : '+str((list(pub_data_retrv.values())[i])['Title'])+ "\n")
317         skrl_text.insert(END,
318              'Publication_Link : '+str((list(pub_data_retrv.values())[i])['Publication_Link'])+ "\n")
319         skrl_text.insert(END,
320              'Publication_Year : '+str((list(pub_data_retrv.values())[i])['Publication_Year'])+ "\n")
321         skrl_text.insert(END,
322              'Authors : '+str((list(pub_data_retrv.values())[i])['Authors'])+ "\n")
323         skrl_text.insert(END,
324              'Profile_Links : '+str((list(pub_data_retrv.values())[i])['Profile_Links'])+ "\n")
325         try:
326             skrl_text.insert(END,
327              'Abstract : '+str((list(pub_data_retrv.values())[i])['Abstract'])+ "\n")
328         except KeyError:
329             skrl_text.insert(END, "No Abstract in the Publication Link"+"\n")
330         skrl_text.insert(END, "\n")
331         print("------------------------------------------------ \n")
332
333     skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)
```

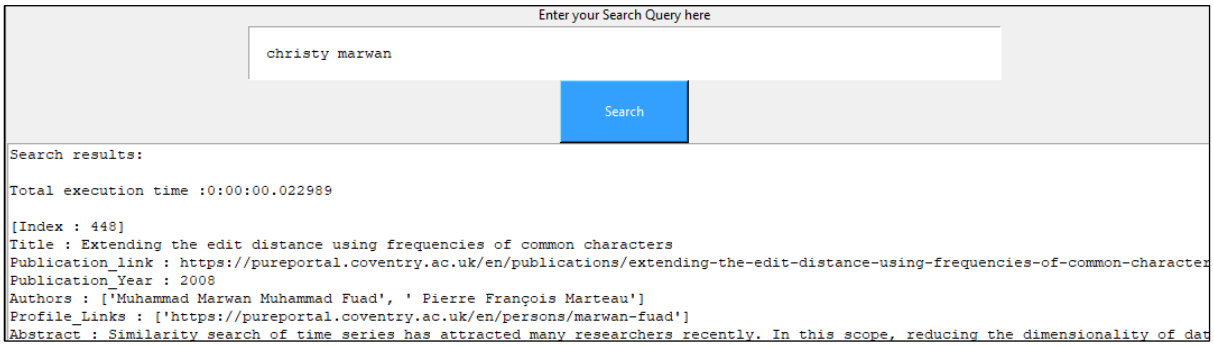


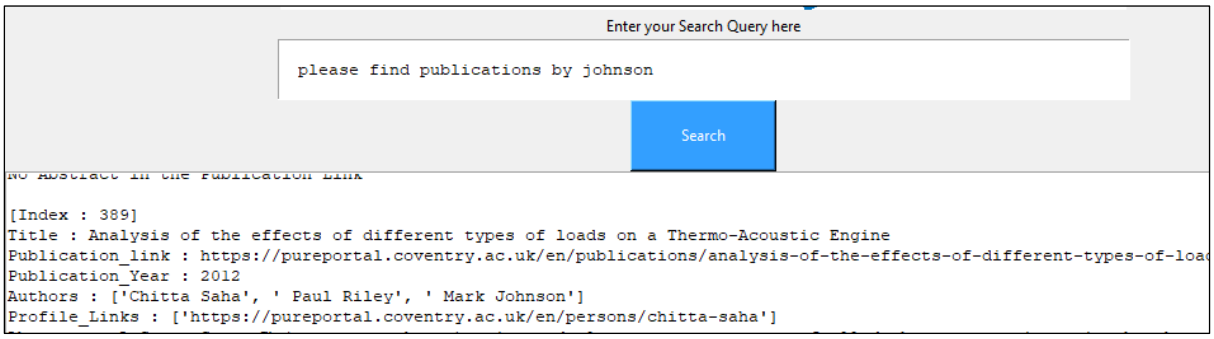

```
Search results:

Total execution time :0:00:00.023988

[Index : 448]
Title : Extending the edit distance using frequencies of common characters
Publication_link : https://pureportal.coventry.ac.uk/en/publications/extending-the-edit-distance-using-frequencies-of-common-character
Publication_Year : 2008
Authors : ['Muhammad Marwan Muhammad Fuad', ' Pierre François Marteau']
Profile_Links : ['https://pureportal.coventry.ac.uk/en/persons/marwan-fuad']
Abstract : Similarity search of time series has attracted many researchers recently. In this scope, reducing the dimensionality of data
 is required to scale up the similarity search. Symbolic representation is a promising technique of dimensionality reduction, since it
allows researchers to benefit from the richness of algorithms used for textual databases. To improve the effectiveness of similarity se
arch we propose in this paper an extension to the edit distance that we call the extended edit distance. This new distance is applied t
o symbolic sequential data objects, and we test it on time series data bases in classification task experiments. We also prove that our
 distance is a metric.
```

Step 9: Checking the efficiency of the search engine by checking the search time

Efficiency is very high a search takes only milliseconds to display the results.


Enter your Search Query here
method using updated lookup tables lic aggregate approxi
Search
Search results:
Total execution time :0:00:00.246837
[Index : 418]
Title : Enhancing the symbolic aggregate approximation method using updated lookup tables
Publication_link : https://pureportal.coventry.ac.uk/en/publications/enhancing-the-symbolic-aggregate-approximation-method-using-updat
Publication_Year : 2010
Authors : ['Muhammad Marwan Muhammad Fuad', ' Pierre François Marteau']
Profile_Links : ['https://pureportal.coventry.ac.uk/en/persons/marwan-fuad']


Enter your Search Query here
christy marwan
Search
Search results:
Total execution time :0:00:00.022989
[Index : 448]
Title : Extending the edit distance using frequencies of common characters
Publication_link : https://pureportal.coventry.ac.uk/en/publications/extending-the-edit-distance-using-frequencies-of-common-character
Publication_Year : 2008
Authors : ['Muhammad Marwan Muhammad Fuad', ' Pierre François Marteau']
Profile_Links : ['https://pureportal.coventry.ac.uk/en/persons/marwan-fuad']
Abstract : Similarity search of time series has attracted many researchers recently. In this scope, reducing the dimensionality of dat

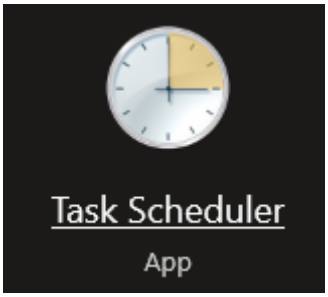

Enter your Search Query here
please find publications by johnson
Search
[Index : 389]
Title : Analysis of the effects of different types of loads on a Thermo-Acoustic Engine
Publication_link : https://pureportal.coventry.ac.uk/en/publications/analysis-of-the-effects-of-different-types-of-loa
Publication_Year : 2012
Authors : ['Chitta Saha', ' Paul Riley', ' Mark Johnson']
Profile_Links : ['https://pureportal.coventry.ac.uk/en/persons/chitta-saha']


Enter your Search Query here
'水下可见光通信空间分集系统'
Search
Search results:
Total execution time :0:00:00.012993
[Index : 102]
Title : 对数正态分布衰落下的光OFDM 水下可见光通信空间分集系统
Publication_link : https://pureportal.coventry.ac.uk/en/publications/optical-ofdm-spatial-diversity-system-in-lognormal-fading-uvlc
Publication_Year : 2020
Authors : ['Hongyan Jiang', ' Hongbing Qiu', ' He Ning', ' Wu Yue', ' Zahir Ahmad', ' Sujan Rajbhandari']

3. Scheduling the crawling script to run every week to fetch the data and update the database/file in local repository

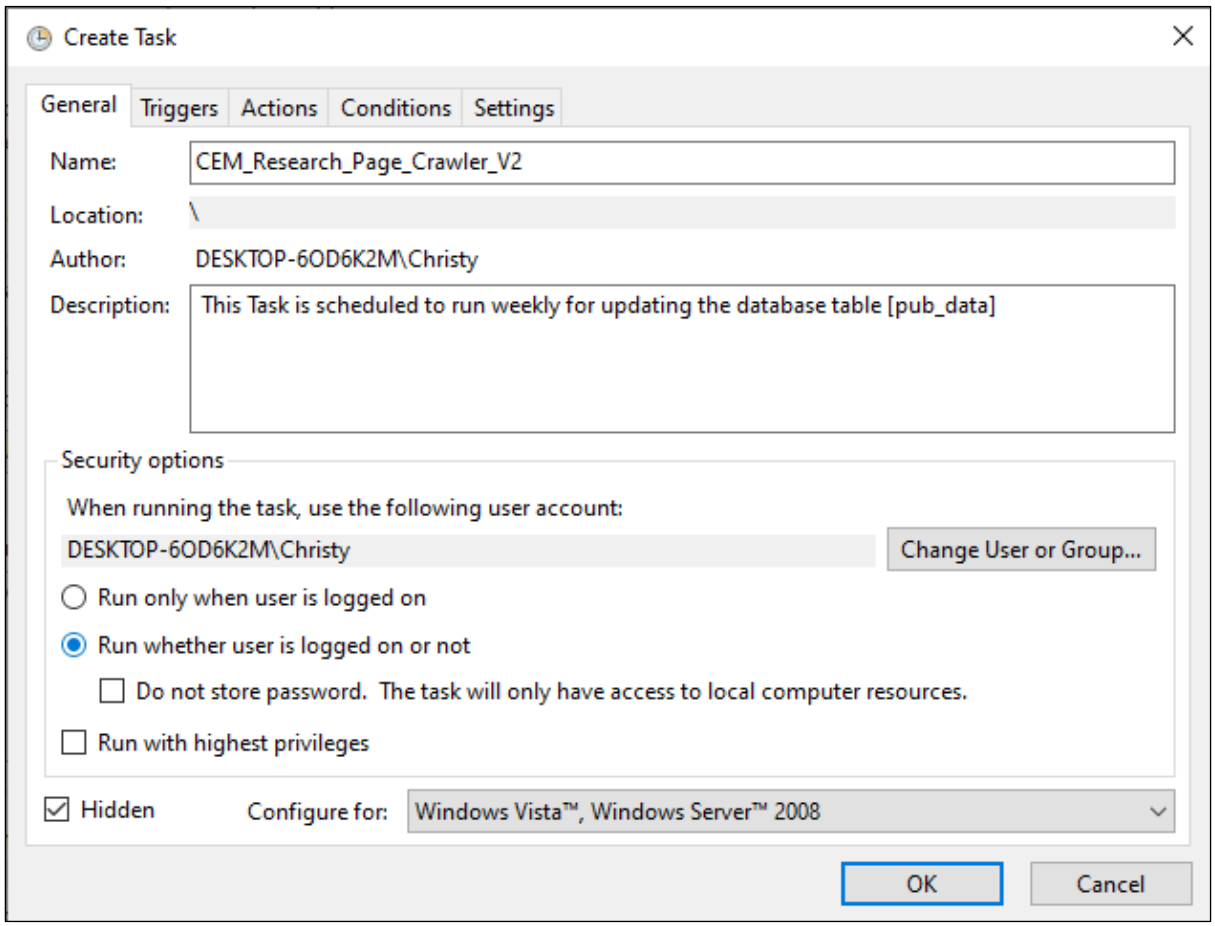
As the crawling and query processing scripts are different, we can schedule the crawling script to execute every week or daily to fetch and update the database/file.

I will be using Windows Task scheduler to schedule the runs



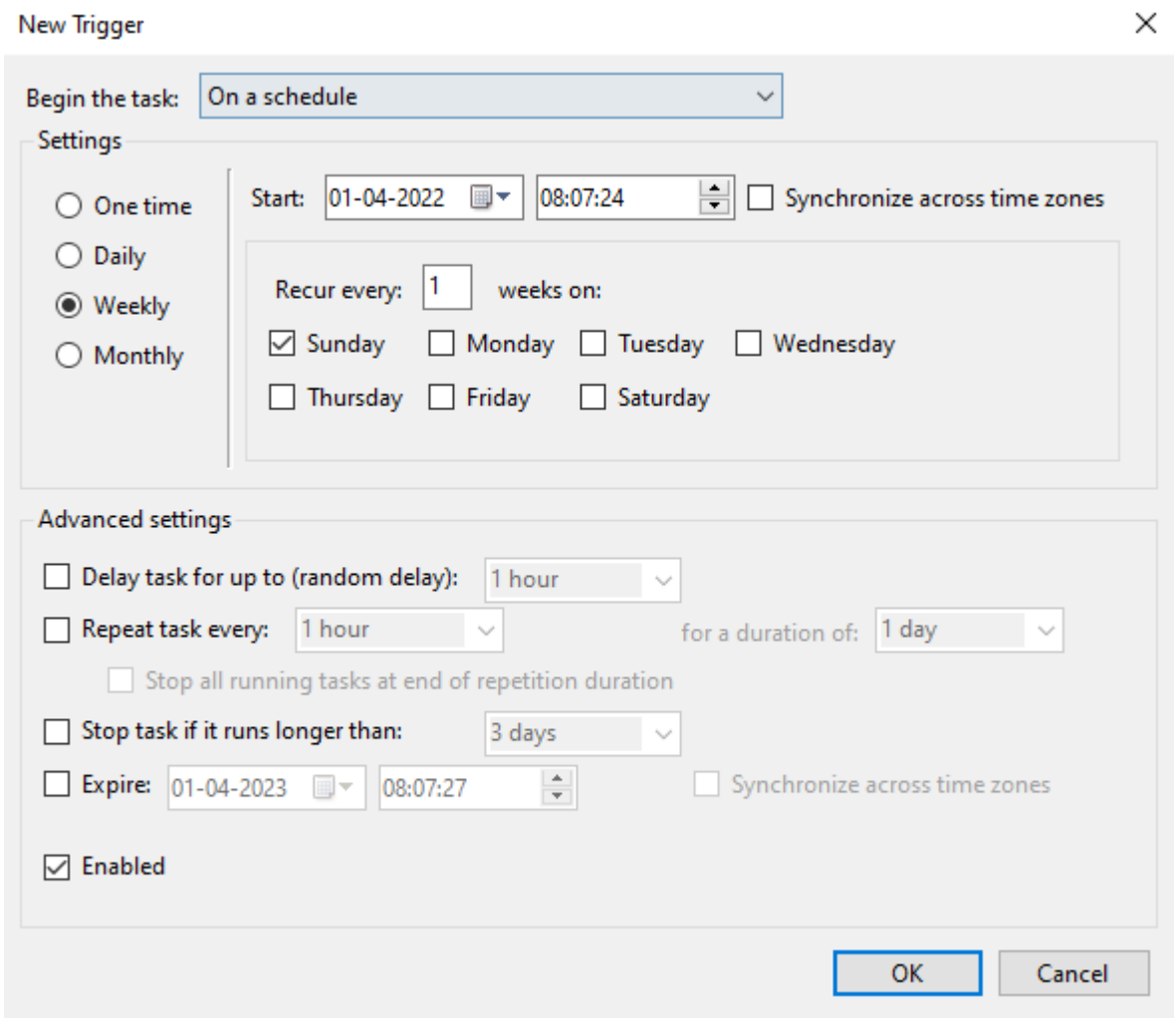
Step 1: Create a new task

Click on the “Create Task” icon to create a new task and select the “Hidden” option to run the Task in background.



Step 2: Creating new “Trigger”

Navigate to “Triggers” tab and click on “New” to schedule the run/execution



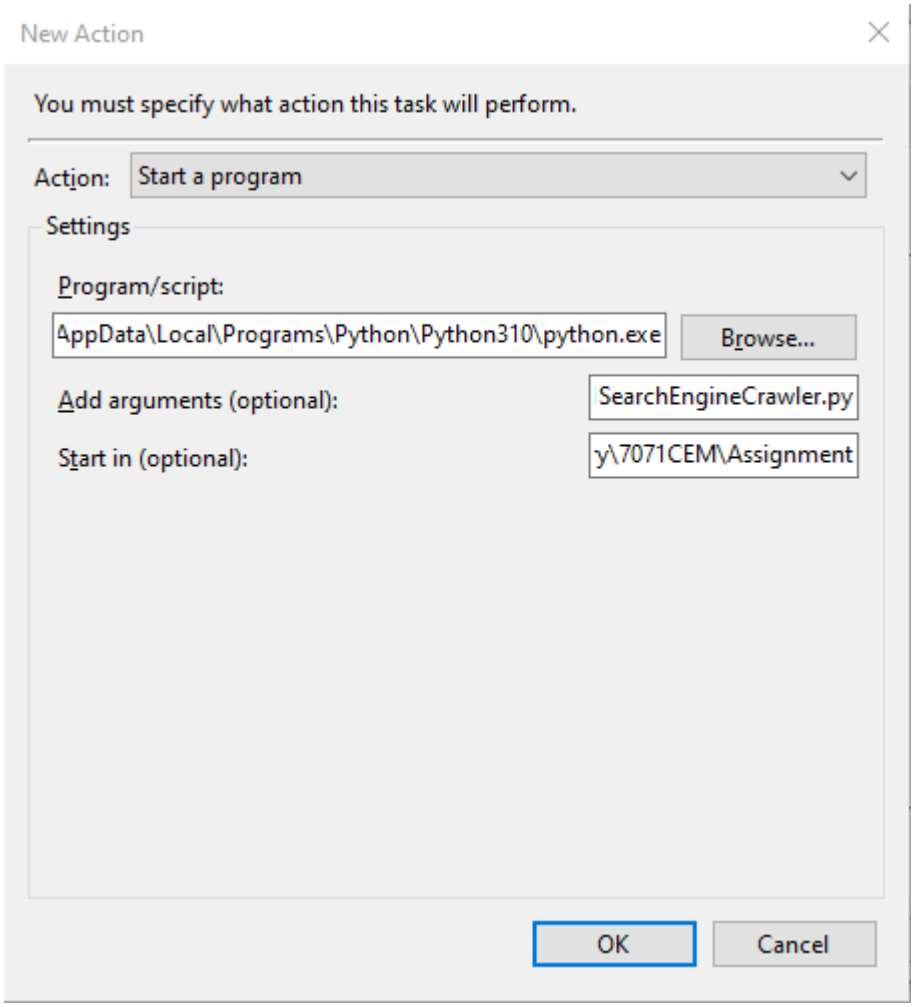
Step 3: Creating new “Action”

Navigate to Actions tab and click on “New” button to create an Action

Program/Script path will be the directory where Python is installed, example: …Christy\AppData\Local\Programs\Python\Python310\python.exe

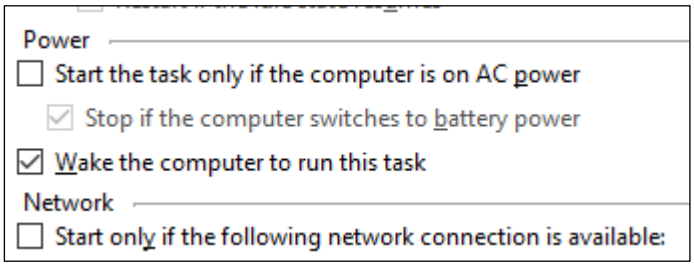
Add Arguments will be the file name with .py extension

Start in, will be the location of your .py[crawler] file



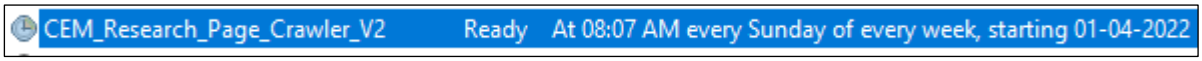
Step 4: Configure Condition and Settings tabs

In condition tab, select “wake the computer to run this task”, so that system will wake up to execute to the task when in sleep mode.



Settings tab can be altered or left default according to the user’s choice.

Click on “OK” to enable the settings



Task2. Document Clustering

In this task I have created a script that can fetch any amount of data from the rss pages using "feedparser" library and performs document clustering using k-means algorithm.

Clustering is a collection of methods for dividing data into groups, or clusters.

Step 1: Generic function to extract feeds

Here I have developed a generic function that can extract title and summary from feeds by passing url of the feed as input.

```
8     import feedparser
```

```
12    def fetch_title_summary(link):
13        feed = feedparser.parse(link)
14        feed_title_summary = []
15        for entry in feed.entries:
16            feed_title_summary.append(entry.title+' '+entry.summary)
17            # feed_title_summary.append(entry.title)
18            # feed_title_summary.append(entry.summary)
19
20        return feed_title_summary
```

Step 2: Extracting feeds for Sports, Science and Business categories

Sports:

```
26    sky_summary = fetch_title_summary("https://www.skysports.com/rss/12040")
27    mirror_summary = fetch_title_summary("https://www.mirror.co.uk/sport/?service=rss")
28    yardbarker_summary = fetch_title_summary("https://www.yardbarker.com/rss/rumors")
29    thesporting_summary = fetch_title_summary("https://thesporting.blog/blog?format=RSS")
30    indepedent_summary = fetch_title_summary("http://www.independent.co.uk/sport/rss")
31    sportingnews_summary = fetch_title_summary("https://www.sportingnews.com/us/rss")
32    sportskeeda_summary= fetch_title_summary("https://www.sportskeeda.com/feed")
```

```
35    sports_feed = sky_summary + mirror_summary + \
36                yardbarker_summary + thesporting_summary + indepedent_summary +\
37                            sportingnews_summary + sportskeeda_summary
38    print("Total Docs for Sports :",len(sports_feed))
```

```
In [69]: print("Total Docs for Sports :",len(sports_feed))
Total Docs for Sports : 177
```

Business:

```
43    cnbc_summary = fetch_title_summary("https://www.cnbc.com/id/19746125/device/rss/rss.xml")
44    economictimes_summary = fetch_title_summary("https://economictimes.indiatimes.com/rssfeedsdefault.cms")
```

```
46     business_feed = cnbc_summary + economictimes_summary
47     print("Total Docs for Business :",len(business_feed))
```

```
In [71]: print("Total Docs for Business :",len(business_feed))
Total Docs for Business : 105
```

Science:

```
newScientist_summary = fetch_title_summary("https://www.newscientist.com/feed/home/?cmpid=RSS%7CNSNS-Home")
sciencedaily_summary = fetch_title_summary("https://www.sciencedaily.com/rss/")
american_global_summary = fetch_title_summary("http://rss.sciam.com/ScientificAmerican-Global")
```

```
science_feed = newScientist_summary + sciencedaily_summary  +\
                american_global_summary
print("Total Docs for Science :", len(science_feed))
```

```
In [73]: print("Total Docs for Science :", len(science_feed))
Total Docs for Science : 185
```

Total documents taken:

```
final_docs = sports_feed + business_feed + science_feed
print("Total docs for Sports, Business and Science feeds :", len(final_docs))
```

```
Total Docs for Sports : 177
Total Docs for Business : 105
Total Docs for Science : 185
Total docs for Sports, Business and Science feeds : 467
```

Step 3: Reading Data through .txt file

As the docs fetched using feedparser didn't give me a good purity score, I extracted the data manually from various data and combined it to a .txt file manually and feed the same to the KMeans algorithm.

```
# ============================================================================
# Data from csv file
# ============================================================================

f = open('sports.txt', 'r')
sports_data = f.read().split("\n")

f = open('business.txt', 'r')
business_data = f.read().split("\n")

f = open('science.txt', 'r')
science_data = f.read().split("\n")

final_docs = sports_data + business_data + science_data
len(final_docs)

# while '' in final_docs:
#     final_docs.remove('')

print("Total docs for Sports, Business and Science feeds :", len(final_docs))
```

Step 4: Pre-Processing the data fetched

Data fetched using feedparser is pre-processed by removing the stop words and stemming the terms.

```
66    import nltk
67    nltk.download("stopwords")
68    from nltk.corpus import stopwords
69
70    sw = stopwords.words('english')
71    # Other languages that stopwords support
72    print("Other languages :",stopwords.fileids())
73
74    # Tokenizing, removing stop words and stemming:
75    # nltk.download("punkt")
76    from nltk.tokenize import word_tokenize
77    from nltk.stem import PorterStemmer
78    import re
```

```
In [77]: print("Other languages :",stopwords.fileids())
Other languages : ['arabic', 'azerbaijani', 'bengali',
'danish', 'dutch', 'english', 'finnish', 'french', 'german',
'greek', 'hungarian', 'indonesian', 'italian', 'kazakh',
'nepali', 'norwegian', 'portuguese', 'romanian', 'russian',
'slovene', 'spanish', 'swedish', 'tajik', 'turkish']
```

```
105   def pre_processing_query(query_list):
106       import re
107
108       # Tokenizing, removing stop words and stemming:
109
110       filtered_words = []
111       for doc in query_list:
112           doc = re.sub('\W+',' ', doc) #Removing special characters
113           tokens = word_tokenize(doc)
114           tmp = ""
115           for w in tokens:
116               if w not in sw:
117                   tmp += (ps.stem(w) + " ").lower()
118           filtered_words.append(tmp)
119
120       return filtered_words
```

Step 5: Vectorizing the pre-processed data

Constructing vectors using TfidfVectorizer

```
 96  from sklearn.feature_extraction.text import TfidfVectorizer
 97  vectorizer = TfidfVectorizer()
 98  X = vectorizer.fit_transform(filtered_docs)
 99  print(X.todense())
100  print(X.shape)
```

```
In [80]: print(X.shape)
(467, 3745)
```

Creating a dataframe to look at the words in the matrix. This gives us an idea about the words present in the matrix.

```
102  import pandas as pd
103  df = pd.DataFrame(X.toarray(), columns = vectorizer.get_feature_names())
104  print(df)
```

```
106  for column_name in vectorizer.get_feature_names():
107      print(column_name, end=" ")
```

Constructing vectors using CountVectorizer

```
109  from sklearn.feature_extraction.text import CountVectorizer
110  vectorizer = CountVectorizer()
111  X = vectorizer.fit_transform(filtered_docs)
112  print(X.todense())
113  print(X.shape)
```

```
In [86]: print(X.shape)
(467, 3745)
```

Step 6: Creating a model for KMeans

Creating KMeans cluster using Sklearn library. Number of clusters taken in 3(k) and vectorizer ised is **TfidfVectorizer**.

```
118  from sklearn.cluster import KMeans
119
120  K = 3
121  model = KMeans(n_clusters=K)#, init='k-means++', max_iter=100, n_init=1)
122  model.fit(X)
123
124  print("cluster no. of input documents, in the order they received:")
125  print(model.labels_)
```

Step 7: Predicting for new queries/docs

```
query = ["iceland fifth asteroid observ impact earth "]
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Science")
print("Predicted:",prediction)
print("\n")
```

```
In [118]: query = ["iceland fifth asteroid observ impact earth "]
     ...: Y = vectorizer.transform(pre_processing_query(query))
     ...: prediction = model.predict(Y)
     ...: print("News Feed: Science")
     ...: print("Predicted:",prediction)
     ...: print("\n")
News Feed: Science
Predicted: [0]
```

```
query = ["we put England under some good pressure but we"]
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Sports")
print("Predicted:",prediction)
print("\n")
```

```
In [119]: query = ["we put England under some good pressure but we"]
     ...: Y = vectorizer.transform(pre_processing_query(query))
     ...: prediction = model.predict(Y)
     ...: print("News Feed: Sports")
     ...: print("Predicted:",prediction)
     ...: print("\n")
News Feed: Sports
Predicted: [2]
```

```
query = ['FT" and "Financial Times" are trademarks of the Financial Times']
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Business")
print("Predicted:",prediction)
print("\n")
```

```
In [120]: query = ['FT" and "Financial Times" are trademarks of the
Financial Times']
     ...: Y = vectorizer.transform(pre_processing_query(query))
     ...: prediction = model.predict(Y)
     ...: print("News Feed: Business")
     ...: print("Predicted:",prediction)
     ...: print("\n")
News Feed: Business
Predicted: [1]
```

Step 8: Using Elbow method to determine the number of clusters (Real Python, n.d.)

Determining the k value using 10 clusters

```
250     # ELBOW Method
251
252     kmeans_kwargs = {"init": "random",
253                      "n_init": 10,
254                      "max_iter": 300,
255                      "random_state": 42}
256
257     # A list for SSE values [Sum of squared errors]
258     sse = []
```

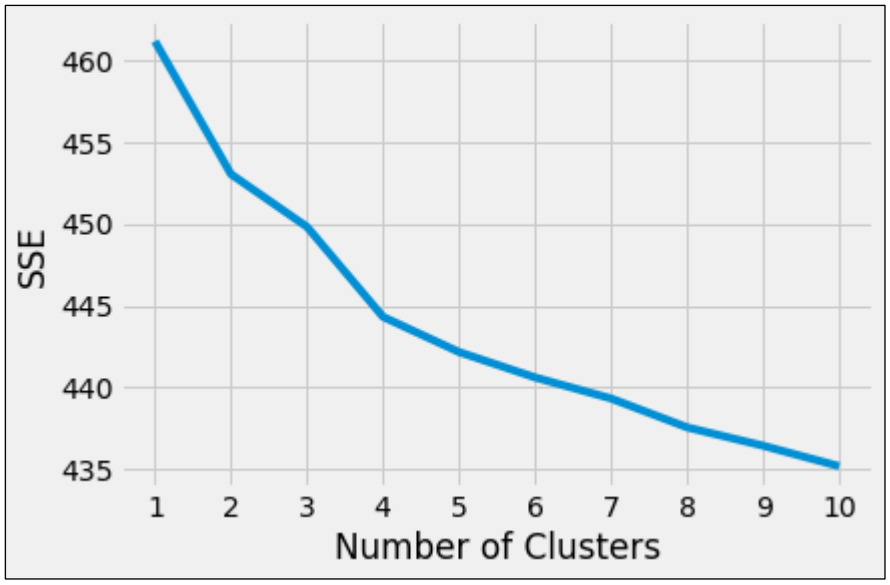
```
260     # ================================================================
261     # # checking output of sse with 10 clusters
262     # ================================================================
263     for k in range(1, 11):
264         kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
265         kmeans.fit(X)
266         sse.append(kmeans.inertia_)
267     print(len(sse))
268     print(sse)
```

```
In [134]: for k in range(1, 11):
     ...:     kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
     ...:     kmeans.fit(X)
     ...:     sse.append(kmeans.inertia_)
     ...: print(len(sse))
     ...: print(sse)
10
[461.1423453290763, 453.03164383237083, 449.81732795129176,
444.3227924145457, 442.1742851663413, 440.6234649364259,
439.3382304842381, 437.5754670873497, 436.45629544629446,
435.206303206919]
```

Plotting elbow graph using matplotlib

```
219     import matplotlib.pyplot as plt
220     from kneed import KneeLocator
```

```
277     plt.style.use("fivethirtyeight")
278     plt.plot(range(1, 11), sse)
279     plt.xticks(range(1, 11))
280     plt.xlabel("Number of Clusters")
281     plt.ylabel("SSE")
282     plt.show()
```




```
283     # ================================================
284     # identifying the elbow point programmatically:
285     # ================================================
286     knee_loc = KneeLocator(range(1, 11), sse,
287                            curve="convex",
288                            direction="decreasing")
289     print("elbow value :", knee_loc.elbow)
```

```
In [136]: knee_loc = KneeLocator(range(1, 11), sse,
     ...:                         curve="convex",
     ...:                         direction="decreasing")
     ...: print("elbow value :", knee_loc.elbow)
elbow value : 4
```

From the above output we can see that the actual number of clusters for given data is 4.

Step 9: Evaluating the model using Purity

Here I am evaluating the model by calculating the **Purity** of the model. As I have taken 3 clusters for my data, I will calculate the sum of correct labels in each cluster divided by the total no. of docs.

```
# ================================================================================
# Calculating purity
# ================================================================================
l1=[]
l2=[]
l3= []
l1 = [1] * len(sports_data)
l2 = [2] * len(business_data)
l3 = [0] * len(science_data)

P1 = 0
P2 = 0
P3 = 0
count = 0
for labels in model.labels_:
    print(labels)
    print("count:", count)
    if(count < len(l1) and labels ==1):
        P1 += 1
        print("Incremented P1")
    elif(count >len(l1) and count <(len(l2)+len(l1)) and labels == 2):
        P2 += 1
        print("Incremented P2")
    elif(count >(len(l2)+len(l1)) and labels == 0):
        P3 += 1
        print("Incremented P3")
    count +=1
```

```
purity = (P1+P2+P3)/len(model.labels_)
print("Purity: ", purity)
```

```
In [44]: purity
Out[44]: 0.53125
```

Purity of 0.53 indicates that only 53% of the data was predicted correctly.

Conclusion:

Accuracy of this model is very low and cannot rely on this technique/algorithm to predict to which group the document belongs to. Accuracy for this model can be increased if the volume of the data and the length of the document[sentence] is increased.

Also, this model was tested with CountVectorizer and TfidfVectorizer. Accuracy for both the vectorizer seems to be low.

Appendix:

Source Code

Task1: SearchEngineCrawler.py script

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 25 05:40:15 2022

@author: Christy
"""
#
================================================================================
==
# Fetching the CRAW DELAY from robots.txt file
#
================================================================================
==

def fetch_crawl_delay():
    import urllib.robotparser
    robo_praser = urllib.robotparser.RobotFileParser()
    robo_praser.set_url("https://pureportal.coventry.ac.uk/robots.txt")

    robo_praser.read()

    crwl_delay = robo_praser.crawl_delay("*")
    print('Crawl_Delay :',crwl_delay, '\n')

#
================================================================================
==


#
================================================================================
==
# Checking if the url obeys the robots.txt conditions
#
================================================================================
==



def link_verify(patterns,url):
    import re
    for pattern in patterns:
        try:
            re.compile(pattern)
            # print(f"'{pattern}' is a VALID regex pattern")
            if re.search(pattern, url):
                # print(f"  '{url}' MATCHES the pattern",pattern)
```

```
                return True
            else:
                print(f"  '{url}' DOES NOT match the pattern",pattern)

        except re.error:
            print(f"'{pattern}' is NOT a valid regex pattern")

    return False

# url = "https://pureportal.coventry.ac.uk/en/organisations/school-of-
computing-electronics-and-maths/publications/?format=rs"
# link_verify([r'/*[?]+export=xls',r'/*[?]+format=rss'], url)

#
==========================================================================
==

#
==========================================================================
==
# Reading the data from json file
#
==========================================================================
==
# load json module

def read_file(fileName):
    import json as js
    try:
        #trying to open a file in read mode
        o = open(fileName,"r")

        # returns JSON object as a dictionary
        file = js.load(o)
        o.close()
        print("File found and read")
    except FileNotFoundError:
        print("File does not exist")
        file = {}

    return file

#
==========================================================================
==

#
==========================================================================
==
# Function to verify the data exist in index/database
#
==========================================================================
==

def update_key(key):

    if(key in pub_data_retrv.keys()):
```

```
            print("Key already exists")
        else:
            pub_data_retrv.update({key:{}})
            print("New Key created")

    def update_values(key, value, title):
        if(key in pub_data_retrv[title].keys() and (pub_data_retrv[title][key])
    != value):
            pub_data_retrv[title].update({key:value})
            print("Updated", 'value for the key: [',key,'] with title', title )

        elif(key in pub_data_retrv[title].keys() and
    (pub_data_retrv[title][key]) == value):
            # print("Value is the same as the one fetched from file")
            a='' # do nothing!!!!!!!!!!!!!!!!!!!

        elif(key not in pub_data_retrv[title].keys()):
            pub_data_retrv[title].update({key:value})
            # print("New key created within the title key:",title)

    #
    ================================================================================
    ==

    #
    ================================================================================
    ==
    # Fetching Profile details of School CEM
    #
    ================================================================================
    ==

    def fetch_profiles():
        # import bs4
        from bs4 import BeautifulSoup
        import requests
        if(link_verify(patterns,prof_url) == False):
            prof_page = requests.get(prof_url)
            # print(page.text)

            prof_soup = BeautifulSoup(prof_page.text, "html.parser")
            # print(soup)

            # Fetching the total page count
            prof_page_count = int(prof_soup.select("nav.pages>ul>li:nth-last-
    child(2) a.step")[0].text)
            print('Total pages to crawl :',prof_page_count)

            # Iterating through each page to fetch the Profile details
            count=0
            for i in range (0, prof_page_count): # page count

                current_page_url = prof_url+'?page='+str(i)
                print("************************************************")
                print("Profile info from Page : ",i+1)
                print(current_page_url)
                print("************************************************")
```

```
            if(link_verify(patterns,current_page_url)== False):
                page =  requests.get(current_page_url)
                soup = BeautifulSoup(page.text, "html.parser")
                profiles = soup.select("ul.grid-results h3.title
a.link.person")
                for profile in profiles:
                    # Fetching Profile link
                    prof_links.add(profile.get('href'))
                    prof_data.update({profile.get('href'):{}})

                    # Fetching name of the profile
                    print(profile.text)

prof_data[profile.get('href')].update({'Name':profile.text})
            count += 1
        print('\n')

    print("No. of pages crawled: ", count)
    print("No. of profiles fetched :", len(prof_data))

#
==========================================================================
==


#
==========================================================================
==
# Fetching Publications from all pages
# Edge case: https://pureportal.coventry.ac.uk/en/publications/sense-
enabled-mixed-reality-museum-exhibitions-2
#
==========================================================================
==

# Global variables

# pub_data = {}

def crawl_and_update():
    import re
    from bs4 import BeautifulSoup
    import requests
    if(link_verify(patterns,url)== False):
        page = requests.get(url)
        # print(page.text)

        soup = BeautifulSoup(page.text, "html.parser")
        # print(soup)

        # Fetching the total page count
        page_count = int(soup.select("nav.pages>ul>li:nth-last-child(2)
a.step")[0].text)
        print('Total pages to crawl :',page_count)

        # Iterating through each page to fet the Publication details
```

```
        count = 0
        pub_id = 1
        for i in range (0, page_count): # page count

            inner_start = datetime.now()

            current_page_url = url+'?page='+str(i)
            print("************************************************")
            print("Publication info from Page : ",i+1)
            print(current_page_url)
            print("************************************************")

            if(link_verify(patterns,current_page_url)== False):
                page =  requests.get(current_page_url)
                soup = BeautifulSoup(page.text, "html.parser")
                publications = soup.select("ul.list-results h3.title
a.link")
                for publication in publications:
                    print(pub_id)

                    # --Title
                    # print('--Title :',publication.text)
                    title = re.sub('\W+','_', publication.text)
                    update_key(title)
                    # print(title, publication.text)
                    update_values('Title', publication.text, title)

                    # --Publication link
                    # print('--Publication link:',publication.get('href'))
                    update_values('Publication_link',
publication.get('href'),title)

                    if(link_verify(patterns,publication.get('href'))==
False):
                        pub_page = requests.get(publication.get('href'))
                        pub_soup = BeautifulSoup(pub_page.text,
"html.parser")

                        # --Publication Year
                        Pub_year = pub_soup.select("tr.status td
span.date")[0].text.split(' ')[-1]
                        # print("--Publication Year :", Pub_year)
                        update_values('Publication_Year', Pub_year, title)

                        # --Author Names
                        Author_names =
pub_soup.select("p.relations.persons")[0].text
                        # print("--Authors:", Author_names)
                        update_values('Authors', Author_names.split(','),
title)

                        # --Profile Links
                        # print("--Profile Links:")
                        temp_pub_links = []
                        if(len(pub_soup.select("div.introduction.no-metrics
p a.link.person")) == 0):
```

```
                            print("** None of the co-authors is a member of
the school CEM")
                            pub_skipped.add(str(pub_id)+'--'+title)
                        else:
                            author_count =
len(pub_soup.select("div.introduction.no-metrics p a.link.person"))
                            pub_valid = False
                            for author in
pub_soup.select("div.introduction.no-metrics p a.link.person"):
                                # print(author.text, ':',
author.get('href'))
                                temp_pub_links.append(author.get('href'))
                                if(pub_valid == False and
(author.get('href') in prof_links)):
                                    pub_valid = True
                            if(pub_valid == False):
                                print("*** None of the authors have profile
link from CEM ")
                                pub_skipped.add(str(pub_id)+'--'+title)
                            else:
                                valid_titles.update({pub_id:title})
                            update_values('Profile_Links', temp_pub_links,
title)

                        # --Abstract
                        # print("--Abstract:")
                        if(len(pub_soup.select("div.textblock")) == 0):
                            print("** This publication does not have an
Abstract", '\n')
                            pub_without_abtrt.add(str(pub_id)+'--'+title)
                        else:
                            #
print(pub_soup.select("div.textblock")[0].text)
                            update_values('Abstract',
pub_soup.select("div.textblock")[0].text, title)

                    if((datetime.now() - inner_start).seconds <
crwl_delay):
                        print("** Waiting for 1 second to load next
publication link")
                        time.sleep(1)

                    pub_id += 1
                    # print("\n")
            count += 1

            if((datetime.now() - inner_start).seconds < crwl_delay):
                print("** Waiting for 1 second to navigate to next page")
                time.sleep(1)

    print("No. of pages crawled: ", count)
    # write_obj.close()

# print("Total execution time :", (datetime.now() - outer_start), '\n')
```

```
#
================================================================================
==
# Saving the data to a .JSON file
#
================================================================================
==

def write_file(filename, data_dict):
    import json as js
    # create json object from dictionary
    js_obj = js.dumps(data_dict)

    # open file for WRITING, "w"
    file_upd = open(filename,"w")

    # write json object to file
    file_upd.write(js_obj)

    # close file
    # file_upd.close()

    print("File written to local disk succesfully")

    return file_upd

# Checking the data in json file
# test_title = re.sub('\W+','_', "Finite size scaling and the zeroes of the
partition function in the Φ44 model")
# print(pub_data_retrv[test_title])
# print(pub_data_retrv[test_title]['Abstract'])

#
================================================================================
==

# def sam_func():
#     print("Hi i am from Task1SearchEngineCrawler")

if __name__ == "__main__":

    from datetime import datetime
    import time

    outer_start = datetime.now()
    print("Started Timer at :",outer_start)
    crwl_delay = 0

    fetch_crawl_delay()
    print("************************************************************")

    patterns = [r'/*[?]+export=xls',r'/*[?]+format=rss']

    link_verify(patterns,"www.example.com/?export=xls")
    link_verify(patterns,"www.example.com/?format=rss")
    link_verify(patterns,"www.example.com/?for=rss")
```

```
    pub_data_retrv = {}
    pub_data_retrv = read_file('pub_data.json')

    print("************************************************************")

    prof_data={}
    prof_links = set()
    prof_url = "https://pureportal.coventry.ac.uk/en/organisations/school-
of-computing-electronics-and-maths/persons/"

    fetch_profiles()
    print("************************************************************")

    pub_skipped = set()
    pub_without_abtrt = set()
    valid_titles ={}
    title =''
    url = "https://pureportal.coventry.ac.uk/en/organisations/school-of-
computing-electronics-and-maths/publications/"

    crawl_and_update()
    print("************************************************************")

    print("No of publications that doen't have authors of CEM'
:",len(pub_skipped))
    print("No of publications without Abstract :",len(pub_without_abtrt),
'\n')

    pub_data_retrv_write = pub_data_retrv

    del_count = 0;
    for key in pub_skipped:
        try:
            pub_data_retrv_write.pop(key.split('--')[1])
            del_count += 1
        except KeyError:
            print('Key not found :', key.split('--')[1])

    print("del_count :",del_count)


    wrt_obj = write_file("pub_data.json",pub_data_retrv_write)
    wrt_obj.close()

    print("************************************************************")

    print("Total execution time :", (datetime.now() - outer_start), '\n')
```

Task 1: Search_Engine_Query_Search.py script

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 28 12:39:45 2022

@author: Christy
"""
```

```
#
================================================================================
==
# Pre-processing the data before creating Inverted index
#
================================================================================
==

# function to check if key exists in the given dictionary
def verify_key(key, dict_name):
    if key in dict_name:
        return True
    else:
        return False

def pre_processing(docs,crawl_filtered_docs):
    import re

    # Tokenizing, removing stop words and stemming:

    import nltk
    nltk.download("stopwords")
    from nltk.corpus import stopwords
    nltk.download("punkt")
    from nltk.tokenize import word_tokenize
    from nltk.stem import PorterStemmer
    sw = stopwords.words('english')
    ps = PorterStemmer()

    if(len(crawl_filtered_docs)>len(pub_data_retrv.keys())):
        crawl_filtered_docs =
crawl_filtered_docs[0:len(pub_data_retrv.keys())]
        # print(crawl_filtered_docs)

    temp_filtered_docs = []
    for doc in docs:
        doc = re.sub('\W+',' ', doc) #Removing special characters
        tokens = word_tokenize(doc)
        tmp = ""
        for w in tokens:
            if w not in sw:
                tmp += (ps.stem(w) + " ").lower()
        temp_filtered_docs.append(tmp)

    crawl_filtered_docs += temp_filtered_docs #problem

    # crawl_filtered_docs = temp_filtered_docs
    return crawl_filtered_docs

def pre_processing_query(query_list):
    import re

    # Tokenizing, removing stop words and stemming:

    import nltk
    nltk.download("stopwords")
```

```
    from nltk.corpus import stopwords
    nltk.download("punkt")
    from nltk.tokenize import word_tokenize
    from nltk.stem import PorterStemmer
    sw = stopwords.words('english')
    ps = PorterStemmer()
    filtered_words = []
    for doc in query_list:
        doc = re.sub('\W+',' ', doc) #Removing special characters
        tokens = word_tokenize(doc)
        tmp = ""
        for w in tokens:
            if w not in sw:
                tmp += (ps.stem(w) + " ").lower()
        filtered_words.append(tmp)

    return filtered_words

#
================================================================================
==
# Creating Inverted Index matrix
#
================================================================================
==

def inverted_index_func(docs,inverted_index):

    for i, doc in enumerate(docs):
        for term in doc.split():
            if term in inverted_index:
                inverted_index[term].add(i)
            else:
                inverted_index[term] = {i}

    # print(inverted_index)


def find_posting_list(query,inverted_index):
    # to be CREATED
    posting_list = set()
    words_to_search = set(query[0].split())
    # print(words_to_search)
    if(len(words_to_search) != 0):
        for word in words_to_search:
            try:
                for index in inverted_index[word.strip()]:
                    posting_list.add(index)
            except KeyError:
                print("No posting_list found for word : ",word)

    return posting_list

#
================================================================================
==
# Rank Retreival  - Vector Space Model
```

```
#
================================================================================
==

def vector_space(filtered_docs):
    from sklearn.feature_extraction.text import TfidfVectorizer
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(filtered_docs)
    return X.todense()

def inner(vec1, vec2):
    prod = 0
    for i in range(len(vec1)):
        prod += vec1[i] * vec2[i]
    return prod

def length(vec):
    import math
    return math.sqrt(inner(vec, vec))

def cosine(vec1, vec2):
    len1 = length(vec1)
    len2 = length(vec2)
    prod = inner(vec1, vec2)
    return prod / (len1 * len2)

# Converting scipy.sparse.csr.csr_matrix to list

def convert_matrix_list(matrix):
    import numpy as np
    np_matrix = matrix.flatten()
    np_array = np.squeeze(np.asarray(np_matrix))
    samp_list = np_array.tolist()
    return samp_list


def vector_space_model(vector_lists_query):

    # calculate the cosine values to rank the documents
    rank_values = []
    list_titles = []
    for i in range (0,(len(vector_lists_query)-1)):
        list_titles.append(list(pub_data_retrv.keys())[i])

rank_values.append(cosine(vector_lists_query[(len(vector_lists_query)-1)],
                                    vector_lists_query[i]))

    print("rank_values length:", len(rank_values))
    print("list_titles length:", len(list_titles))

    return rank_values,list_titles

# Storing the ranks of the document as pandas datframe
def sort_rank(rank_values, list_titles, posting_list_indexes):
    import pandas as pd
    rank_df = pd.DataFrame({
        'Index' : posting_list_indexes,
```

```
            'Title' : list_titles,
            'Rank' : rank_values
            })

        rank_df = rank_df.sort_values(by='Rank', ascending=False)
        rank_df = rank_df.reset_index(drop=True)

        return rank_df

    def tkinter_GUI(inverted_index):
        import tkinter as tk
        from PIL import Image, ImageTk
        from tkinter.scrolledtext import ScrolledText
        from tkinter import END
        import re

        root = tk.Tk()
        root.title(' Seacrh Engine || Publications of School of CEM || -
    Christy Jacob')

        canvas = tk.Canvas(root, width=1100, height=600)
        canvas.grid(columnspan=3, rowspan=7)

        # Inserting logo
        logo = Image.open('coventry-uni-logo.png')
        logo = ImageTk.PhotoImage(logo)
        logo_label = tk.Label(image=logo)
        logo_label.image = logo
        logo_label.grid(column=1, row=0)

        # Search Instrctions

        search_inst = tk.Label(root, text="Enter your Search Query here")
        search_inst.grid(columnspan=3, column=0, row=1)

        text_box1 = tk.Text(root, height=1, width=80, padx=15, pady=15)
        text_box1.grid(columnspan=3, column=0, row=2)


        def search_query(pub_data_retrv, inverted_index):

            skrl_text = ScrolledText(root)
            skrl_text.config(width = 135, height = 18)

            from datetime import datetime

            outer_start = datetime.now()
            # print("Started Timer at :",outer_start)

            query = text_box1.get("1.0","end-1c")

            #
    ================================================================================
    ==
            #          # PRE-PROCESSING the query
```

```
        #
==========================================================================
==
        query_filtered_docs = pre_processing_query([query])
        # print(query_filtered_docs)

        # filtered_indexed_docs = []
        posting_list_indexes =
find_posting_list(query_filtered_docs,inverted_index )

    #
==========================================================================
==
        # Creating docs with a combination of the Title, Authors and
Abstract
        # Converting all strings to lowercase
        docs=[]

        for index in posting_list_indexes:
            temp_string = ''
            temp_string +=
(list(pub_data_retrv.keys())[index]).replace('_',' ').lower()+' ' #adding
title
            if(verify_key('Authors',list(pub_data_retrv.keys())[index])):
                temp_string +=
(''.join(list(pub_data_retrv.keys())[index]['Authors'])).lower()+' '
            else:
                temp_string += ''
            if(verify_key('Abstract',list(pub_data_retrv.keys())[index])):
                temp_string +=
(list(pub_data_retrv.keys())[index]['Abstract']).lower()
            else:
                temp_string += ''
            docs.append(temp_string)

        print("Docs created")
    #
==========================================================================
==

        # Pre-processing

        crawl_filtered_docs = []
        crawl_filtered_docs = pre_processing(docs,crawl_filtered_docs)
        # print(crawl_filtered_docs)
    #
==========================================================================
==


        if(len(re.sub('\W+','', query)) < 3):
            print("Please enter atleast three valid character to
search!!!")
            skrl_text.insert(END, "Please enter atleast three valid
character to search!!!"+ "\n")
            skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)
            # close()
```

```
        elif(len(posting_list_indexes) == 0):
            skrl_text.insert(END, "Sorry!, no results found."+"\n")
            skrl_text.insert(END, "Please add more valid keywords to search
Query"+"\n")
            skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)
        else:
            posting_list_indexes = list(posting_list_indexes)
            print("posting_list_indexes", posting_list_indexes)

            crawl_filtered_docs += query_filtered_docs
            print("Length of new filtered docs with Query
:",len(crawl_filtered_docs))

    #
==========================================================================
==
    #         # Vector space model
    #
==========================================================================
==

            # Creating vectors for each doc
            vector_matrix = vector_space(crawl_filtered_docs)
            print(vector_matrix.shape)
            rows, columns = vector_matrix.shape
            print("rows :", rows, " columns :", columns)

            vector_lists_query = []
            for matrix in vector_matrix:
                vector_lists_query.append(convert_matrix_list(matrix))

            rank_values, list_titles =
vector_space_model(vector_lists_query)

            # Storing the ranks of the document
            rank_df = sort_rank(rank_values, list_titles,
posting_list_indexes)

            # display result
            print("------------------------------------------------ \n")
            print("Searh query :",query)
            print("Search results \n")
            print("Total execution time :", (datetime.now() - outer_start),
'\n')
            skrl_text.insert(END, "Search results:"+ "\n")
            skrl_text.insert(END, "\n")
            skrl_text.insert(END, "Total execution time :"+
str((datetime.now() - outer_start))+"\n")
            skrl_text.insert(END, "\n")
            for i in list(rank_df['Index']):
                print(list(pub_data_retrv.values())[i]);
                skrl_text.insert(END, "[Index : "+str(i)+"]"+"\n")
                skrl_text.insert(END,
                     'Title :
'+str((list(pub_data_retrv.values())[i])['Title'])+ "\n")
                skrl_text.insert(END,
```

```
                    'Publication_link :
'+str((list(pub_data_retrv.values())[i])['Publication_link'])+ "\n")
                skrl_text.insert(END,
                    'Publication_Year :
'+str((list(pub_data_retrv.values())[i])['Publication_Year'])+ "\n")
                skrl_text.insert(END,
                    'Authors :
'+str((list(pub_data_retrv.values())[i])['Authors'])+ "\n")
                skrl_text.insert(END,
                    'Profile_Links :
'+str((list(pub_data_retrv.values())[i])['Profile_Links'])+ "\n")
                try:
                    skrl_text.insert(END,
                    'Abstract :
'+str((list(pub_data_retrv.values())[i])['Abstract'])+ "\n")
                except KeyError:
                    skrl_text.insert(END, "No Abstract in the Publication
Link"+"\n")
                skrl_text.insert(END, "\n")
                print("------------------------------------------------ \n")

            skrl_text.grid(columnspan=3, rowspan=3, column=0, row=4)

    # Search button
    search_text = tk.StringVar()
    search_btn = tk.Button(root,
                           command=lambda:search_query(pub_data_retrv,
                                                       inverted_index),
                           textvariable = search_text, bg="#339FFF",
                           fg="white", height=3, width=15)
    search_text.set("Search")
    search_btn.grid(column=1, row=3)

    canvas = tk.Canvas(root, width=800, height=100)
    canvas.grid(columnspan=3)

    root.mainloop()


#
================================================================================
==


#
================================================================================
==

if __name__ == "__main__":

    from datetime import datetime
    from Task1SearchEngineCrawler import read_file, write_file

    outer_start = datetime.now()
    # print("Started Timer at :",outer_start)

    pub_data_retrv = {}
```

```
    pub_data_retrv = read_file('pub_data.json')

    inverted_index = {}
    inverted_index = read_file('index_data.json')

#
================================================================================
==
#     # Updating inverted index
#
================================================================================
==
    p_docs=[]

    for key in pub_data_retrv.keys():
        temp_string = ''
        temp_string += key.replace('_',' ').lower()+' ' #adding title
        if(verify_key('Authors',pub_data_retrv[key])):
            temp_string +=
(''.join(pub_data_retrv[key]['Authors'])).lower()+' '
        else:
            temp_string += ''
        if(verify_key('Abstract',pub_data_retrv[key])):
            temp_string += (pub_data_retrv[key]['Abstract']).lower()
        else:
            temp_string += ''
        p_docs.append(temp_string)

    print("Docs created for creating inverted_index")
    inverted_index = {key: set(value) for key, value in
inverted_index.items()}
    inverted_index_func(p_docs,inverted_index)
    inverted_index = {key: list(value) for key, value in
inverted_index.items()}
    write_file('index_data.json', inverted_index)
    inverted_index = {key: set(value) for key, value in
inverted_index.items()}

    #Opening GUI
    tkinter_GUI(inverted_index)


    #
================================================================================
==
                                    # The END

    # Check for the efficieny using below words
    # asăvoa
    # '水下可见光通信空间分集系统'
    # '对数正态分布衰落下的光ofdm'
    # 'zažímalová'
    # βc
    # l_2
    # görtler
    # σ
```

```
    #
================================================================================
==
```

Task2: Document_clustering_v2.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 29 17:16:27 2022

@author: Christy
"""

import feedparser

# Source:https://blog.feedspot.com/category/?_src=home

def fetch_title_summary(link):
    feed = feedparser.parse(link)
    feed_title_summary = []
    for entry in feed.entries:
        feed_title_summary.append(entry.title+' '+entry.summary)
        # feed_title_summary.append(entry.title)
        # feed_title_summary.append(entry.summary)

    return feed_title_summary

# SPORTS

# link = "https://www.skysports.com/rss/12040"

sky_summary = fetch_title_summary("https://www.skysports.com/rss/12040")
mirror_summary =
fetch_title_summary("https://www.mirror.co.uk/sport/?service=rss")
yardbarker_summary =
fetch_title_summary("https://www.yardbarker.com/rss/rumors")
thesporting_summary =
fetch_title_summary("https://thesporting.blog/blog?format=RSS")
indepedent_summary =
fetch_title_summary("http://www.independent.co.uk/sport/rss")
sportingnews_summary =
fetch_title_summary("https://www.sportingnews.com/us/rss")
sportskeeda_summary=
fetch_title_summary("https://www.sportskeeda.com/feed")


sports_feed = sky_summary + mirror_summary + \
            yardbarker_summary + thesporting_summary + indepedent_summary
+\
                        sportingnews_summary + sportskeeda_summary
print("Total Docs for Sports :",len(sports_feed))


# BUSINESS
```

```
cnbc_summary =
fetch_title_summary("https://www.cnbc.com/id/19746125/device/rss/rss.xml")
economictimes_summary =
fetch_title_summary("https://economictimes.indiatimes.com/rssfeedsdefault.c
ms")

business_feed = cnbc_summary + economictimes_summary
print("Total Docs for Business :",len(business_feed))

# SCIENCE

newScientist_summary =
fetch_title_summary("https://www.newscientist.com/feed/home/?cmpid=RSS%7CNS
NS-Home")
sciencedaily_summary =
fetch_title_summary("https://www.sciencedaily.com/rss/")
american_global_summary =
fetch_title_summary("http://rss.sciam.com/ScientificAmerican-Global")

science_feed = newScientist_summary + sciencedaily_summary  +\
                american_global_summary
print("Total Docs for Science :", len(science_feed))


final_docs = sports_feed + business_feed + science_feed
print("Total docs for Sports, Business and Science feeds :",
len(final_docs))


#
==========================================================================
==
# Data from csv file
#
==========================================================================
==

f = open('sports.txt', 'r')
sports_data = f.read().split("\n")

f = open('business.txt', 'r')
business_data = f.read().split("\n")

f = open('science.txt', 'r')
science_data = f.read().split("\n")

final_docs = sports_data + business_data + science_data
len(final_docs)

# while '' in final_docs:
#     final_docs.remove('')

print("Total docs for Sports, Business and Science feeds :",
len(final_docs))
```

```
# PRE-PROCESSING

import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords

sw = stopwords.words('english')
# Other languages that stopwords support
print("Other languages :",stopwords.fileids())


filtered_docs = []
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
nltk.download("punkt")
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
sw = stopwords.words('english')
ps = PorterStemmer()

def pre_processing_query(query_list):
    import re

    # Tokenizing, removing stop words and stemming:

    filtered_words = []
    for doc in query_list:
        doc = re.sub('\W+',' ', doc) #Removing special characters
        tokens = word_tokenize(doc)
        tmp = ""
        for w in tokens:
            if w not in sw:
                tmp += (ps.stem(w) + " ").lower()
        filtered_words.append(tmp)

    return filtered_words

filtered_docs = pre_processing_query(final_docs)

print(filtered_docs)
print(len(filtered_docs))

# CONSTRUCTING VECTORS

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(filtered_docs)
print(X.todense())
print(X.shape)

# import pandas as pd
# df = pd.DataFrame(X.toarray(), columns = vectorizer.get_feature_names())
# print(df)

# for column_name in vectorizer.get_feature_names():
#     print(column_name, end=" ")
```

```
# from sklearn.feature_extraction.text import CountVectorizer
# vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(filtered_docs)
# print(X.todense())
# print(X.shape)

#
================================================================================
==
# # Using K-means for clustering
#
================================================================================
==
from sklearn.cluster import KMeans

K = 3
model = KMeans(n_clusters=K)#, init='k-means++', max_iter=100, n_init=1)
model.fit(X)

print("cluster no. of input documents, in the order they received:")
print(model.labels_)

# Trial 2
# model = KMeans(n_clusters=K,init='k-means++', max_iter=200, n_init=50)
# model.fit(X)

# print("cluster no. of input documents, in the order they received:")
# print(model.labels_)

#
================================================================================
==
# # PREDICTION
#
================================================================================
==

query = ["football is great "]
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Sports")
print("Predicted:",prediction)
print("\n")

query = ["help researchers understand precisely what happened."]
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Science")
print("Predicted:",prediction)
print("\n")

query = ['key yield spread, which inverted Monday for the']
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Business")
print("Predicted:",prediction)
```

```
print("\n")

query = ["Biological experiments are conducted somewhere"]
Y = vectorizer.transform(pre_processing_query(query))
prediction = model.predict(Y)
print("News Feed: Science")
print("Predicted:",prediction)
print("\n")

#
================================================================================
==
# Calculating purity
#
================================================================================
==
l1=[]
l2=[]
l3= []
l1 = [1] * len(sports_data)
l2 = [2] * len(business_data)
l3 = [0] * len(science_data)

P1 = 0
P2 = 0
P3 = 0
count = 0
for labels in model.labels_:
    print(labels)
    print("count:", count)
    if(count < len(l1) and labels ==1):
        P1 += 1
        print("Incremented P1")
    elif(count >len(l1) and count <(len(l2)+len(l1)) and labels == 2):
        P2 += 1
        print("Incremented P2")
    elif(count >(len(l2)+len(l1)) and labels == 0):
        P3 += 1
        print("Incremented P3")
    count +=1

purity = (P1+P2+P3)/len(model.labels_)
print("Purity: ", purity)



#
================================================================================
==
# Experiment 2: Elbow
#
================================================================================
==

import matplotlib.pyplot as plt
from kneed import KneeLocator
from sklearn.cluster import KMeans
```

```
from sklearn.metrics import silhouette_score

kmeans = KMeans(init="random",
                n_clusters=3,
                n_init=10,
                max_iter=300,
                random_state=42)
kmeans.fit(X)

# The lowest SSE value
print(kmeans.inertia_)

# Final locations of the centroid
print(kmeans.cluster_centers_)
print(kmeans.cluster_centers_.shape)


# The number of iterations required to converge
print(kmeans.n_iter_)

# cluster assignments are stored as a one-dimensional NumPy array
# First 5 predicted labels
print(kmeans.labels_[:5])
print(kmeans.labels_)

#
# Choosing the Appropriate Number of Clusters
#
# ELBOW Method

kmeans_kwargs = {"init": "random",
                 "n_init": 10,
                 "max_iter": 300,
                 "random_state": 42}

# A list for SSE values [Sum of squared errors]
sse = []

#
# # checking output of sse with 10 clusters
#
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)
print(len(sse))
print(sse)
```

```
#
==========================================================================
==
# ['fivethirtyeight',
#  'seaborn-pastel',
#  'seaborn-whitegrid',
#  'ggplot',
#  'grayscale']
#
==========================================================================
==
plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
#
==========================================================================
==
# identifying the elbow point programmatically:
#
==========================================================================
==
knee_loc = KneeLocator(range(1, 11), sse,
                       curve="convex",
                       direction="decreasing")
print("elbow value :", knee_loc.elbow)

#
==========================================================================
==
# # checking output of sse with 20 clusters
#
==========================================================================
==
sse = []
for k in range(1, 21):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)
print(len(sse))
print(sse)

plt.style.use("fivethirtyeight")
plt.plot(range(1, 21), sse)
plt.xticks(range(1, 21))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()

#
==========================================================================
==
# identify the elbow point programmatically:
```

```
#
================================================================================
==
kl = KneeLocator(range(1, 21), sse,
                 curve="convex",
                 direction="decreasing")
print("knee/elbow :", kl.elbow)

#
================================================================================
==
# # checking output of sse with 15 clusters
#
================================================================================
==
sse = []
for k in range(1, 16):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)
print(len(sse))
print(sse)

plt.style.use("fivethirtyeight")
plt.plot(range(1, 16), sse)
plt.xticks(range(1, 16))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()

#
================================================================================
==
# identify the elbow point programmatically:
#
================================================================================
==

kl = KneeLocator(range(1, 16), sse,
                 curve="convex",
                 direction="decreasing")
print("knee/elbow :", kl.elbow)
```

References:

w3schools, (n.d.). Python RegEx (w3schools.com). Retrieved on March 30, 2022, from https://www.w3schools.com/python/python_regex.asp

Real Python, (n.d). K-Means Clustering in Python: A Practical Guide – Real Python. Retrieved on March 31, 2022, from https://realpython.com/k-means-clustering-python/