Logo

Description automatically generated

# 7071CEM Information Retrieval

**Student name:** CHRISTY ABRAHAM JACOB

**Student ID:** 11760090

Table of Contents

[7071CEM Information Retrieval 1](#_Toc99722177)

[Task 1. Search Engine 4](#_Toc99722178)

[1. Steps to crawl and store the data fetched during web page crawl 4](#_Toc99722179)

[**Step 1**: Reading “robots.txt” file of School of CEM to fetch the crawl delay and the links that are disallowed to crawl 4](#_Toc99722180)

[**Step 2**: Defining a function to verify the links used to crawl the website 5](#_Toc99722181)

[**Step 3**: Fetching data from the database or file that was already crawled for the same page 6](#_Toc99722182)

[**Step 4**: Fetching profile names and links of all the profile that belongs to school of CEM 6](#_Toc99722183)

[**Step 5**: Fetching publication details in which at least one co-author is a current faculty of school of CEM 8](#_Toc99722184)

[**Step 6**: Updating/Storing the information fetched to database 11](#_Toc99722185)

[2. Steps to process the query and find the most relevant publication 12](#_Toc99722186)

[**Step 1**: Reading the crawl data and inverted index data from the database/file 12](#_Toc99722187)

[**Step 2**: Creating the GUI function which is responsible for the query fetching and processing 13](#_Toc99722188)

[**Step 3**: Creating a basic GUI for the search engine 13](#_Toc99722189)

[**Step 4**: Processing the crawled data to construct inverted index 15](#_Toc99722190)

[**Step 5**: Creating inverted index matrix 17](#_Toc99722191)

[**Step 6**: Fetching the query and pre-processing the query 17](#_Toc99722192)

[**Step 7**: Performing Ranked retrieval using Vector Space Model 19](#_Toc99722193)

[**Step 8**: Displaying the results 21](#_Toc99722194)

[**Step 9**: Checking the efficiency of the search engine by checking the search time 21](#_Toc99722195)

[3. Scheduling the crawling script to run every week to fetch the data and update the database/file in local repository 23](#_Toc99722196)

[**Step 1**: Create a new task 23](#_Toc99722197)

[**Step 2**: Creating new “Trigger” 24](#_Toc99722198)

[**Step 3**: Creating new “Action” 24](#_Toc99722199)

[**Step 4**: Configure Condition and Settings tabs 25](#_Toc99722200)

[Task2. Document Clustering 26](#_Toc99722201)

[**Step 1**: Generic function to extract feeds 26](#_Toc99722202)

[**Step 2**: Extracting feeds for Sports, Science and Business categories 26](#_Toc99722203)

[**Step 3**: Reading Data through .txt file 27](#_Toc99722204)

[**Step 4**: Pre-Processing the data fetched 27](#_Toc99722205)

[**Step 5**: Vectorizing the pre-processed data 28](#_Toc99722206)

[**Step 6**: Creating a model for KMeans 29](#_Toc99722207)

[**Step 7**: Predicting for new queries/docs 29](#_Toc99722208)

[**Step 8**: Using Elbow method to determine the number of clusters (Real Python, n.d.) 30](#_Toc99722209)

[**Step 9**: Evaluating the model using Purity 32](#_Toc99722210)

[Conclusion: 33](#_Toc99722211)

[Appendix: 34](#_Toc99722212)

[Source Code 34](#_Toc99722213)

[Task1: SearchEngineCrawler.py script 34](#_Toc99722214)

[Task 1: Search\_Engine\_Query\_Search.py script 41](#_Toc99722215)

[Task2: Document\_clustering\_v2.py 49](#_Toc99722216)

[References: 58](#_Toc99722217)

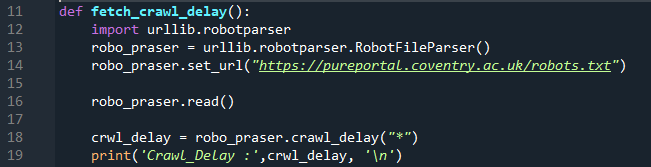
# Task 1. Search Engine

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

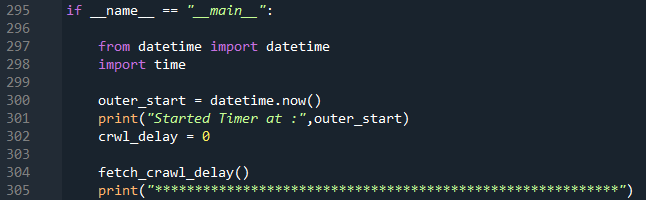
### **Step 1**: Reading “robots.txt” file of School of CEM to fetch the crawl delay and the links that are disallowed to crawl





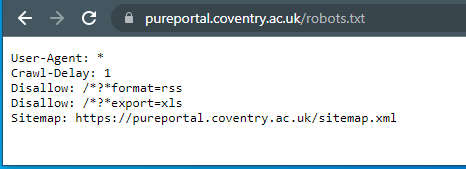
Above code parses the content in “robots.txt” file and assigns it to the variable named “crwl\_delay” initialised in the main method.

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.





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

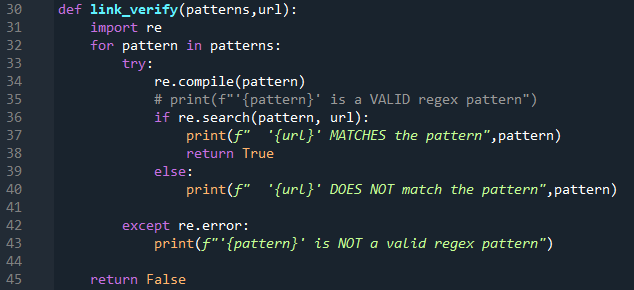


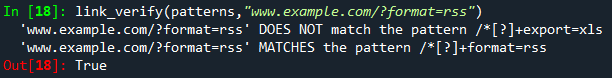
From <https://pureportal.coventry.ac.uk/robots.txt> we can notice that 2 links that contains /\*?\*format=rss and /\*?\*export=xls are disallowed. In-order to obey this rule, we need to create a function or develop a regular expression in python that verifies above url formats.

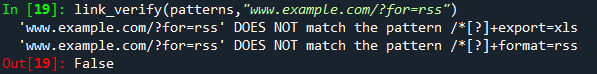


In the above regular expression, ‘\*’ verifies zero or more occurrences of the pattern towards the left, ‘+’ verifies one or more occurrences of the pattern and, characters that needs to be verified are defined inside a pair of ‘[]’ as shown in the above code snippet. (w3schools, n.d.)

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

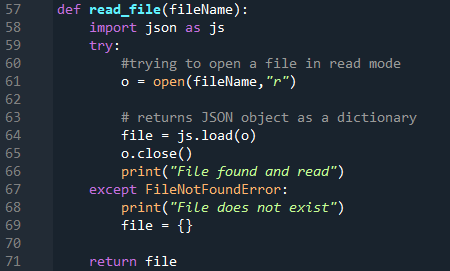






### **Step 3**: Fetching data from the database or file that was already crawled for the same page

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.







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.

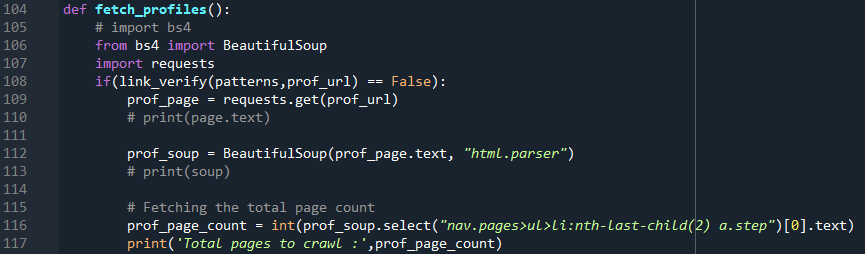
### **Step 4**: Fetching profile names and links of all the profile that belongs to school of CEM

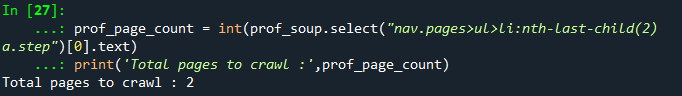
In the main program I declare the variables that is required to store the profile data. In this script I have used a dictionary to store the profile and link as a key-value pair for future reference in the script and have declared an empty set to store profile links so that the script doesn’t store duplicate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

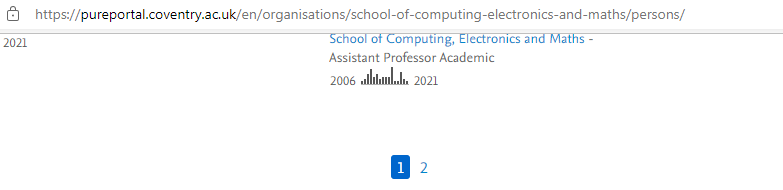


To fetch the page content and parse it we can use a python library named “BeautifulSoup” and using the object created for beautifulSoup we can access different html tags to fetch the desired content.

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.

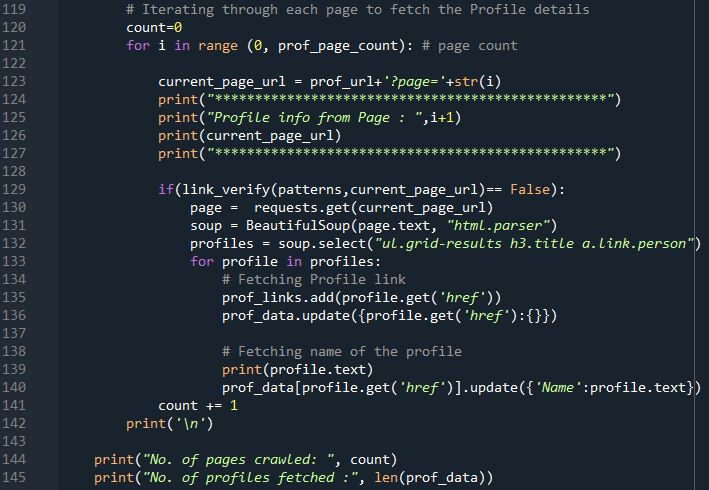


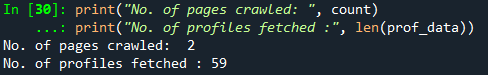




In the above code snippet, css path is used to identify the total pages in profile url and use that value to navigate from one page to another during crawling. This approach avoids hardcoding the values and the code works efficiently even if the page count is updated on the website in future.

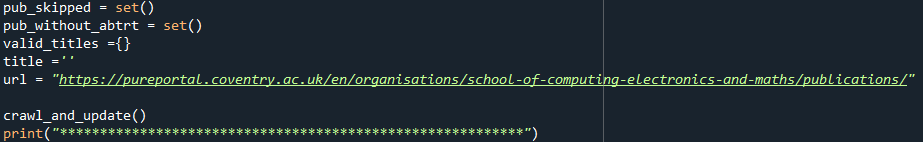
From the page count value obtained, we can iterate through the pages to fetch the profile links and name. Profile link is used as the key and name is stored as its value.

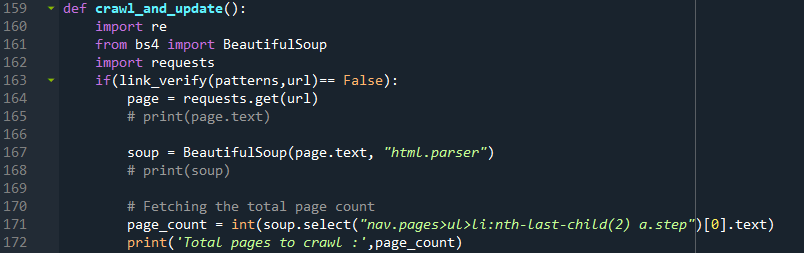


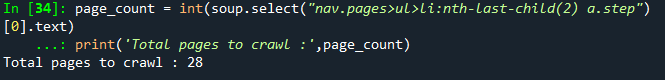


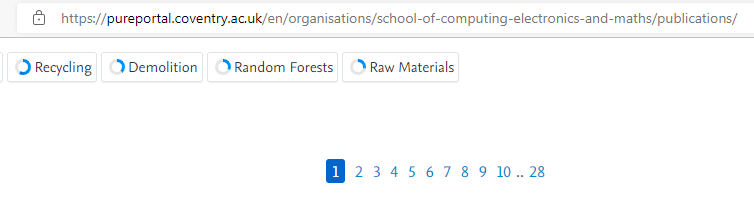
### **Step 5**: Fetching publication details in which at least one co-author is a current faculty of school of CEM

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.

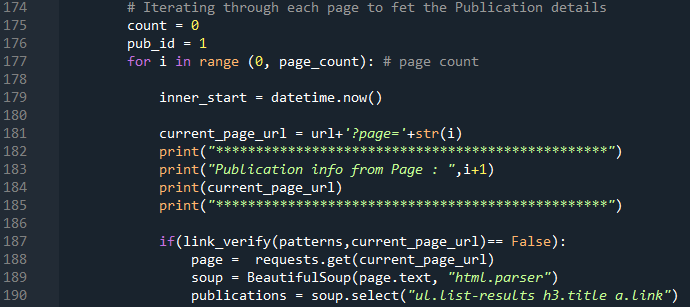




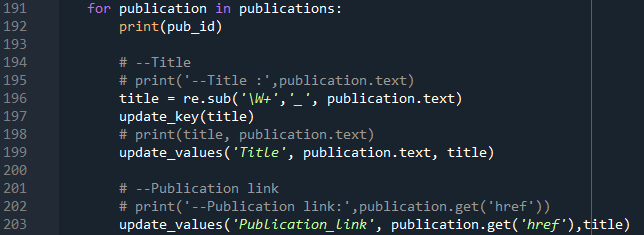




Script fetches publication link first in-order to navigate to the publication page and fetch other details like authors, author profiles, publication title, publication year and abstract.

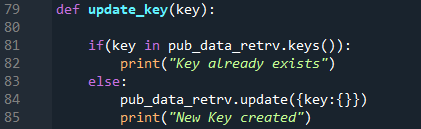


Once all the publication links are fetched from the current page, script iterates through each publication link as shown in the below code snippet.



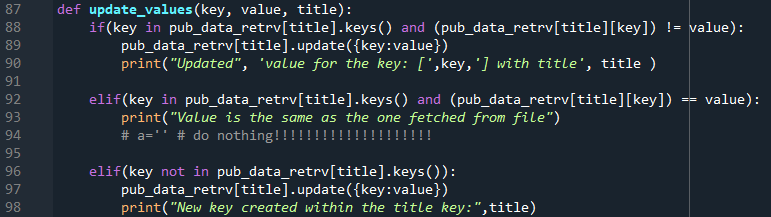
As the code to store and compare the values fetched is generic, its best practice to define a separate function that fetches and compares with the values that is fetched from the .json file or database.

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.



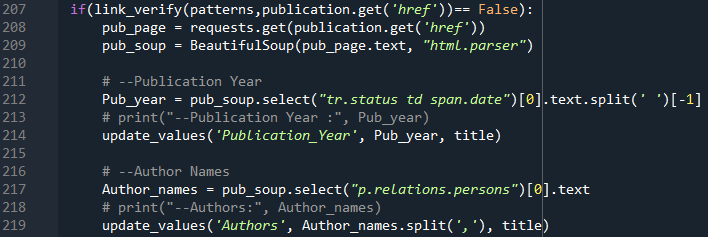


Similarly, update\_value() verifies the values for key passed as argument when the function is called.



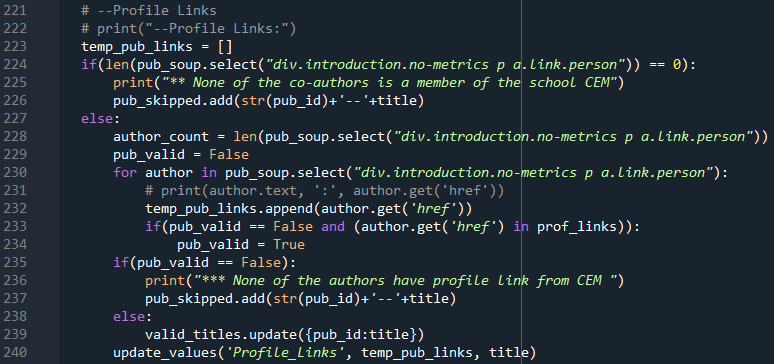


Next, script verifies if the publication link obeys the “robots.txt” file conditions. If it obeys, other details like publication year and author names are fetched.

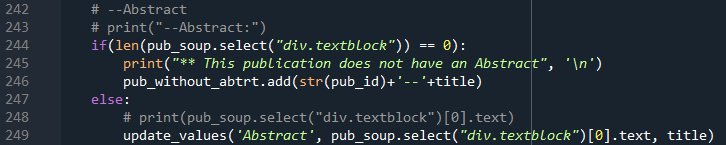


As there can be multiple profile links for a single publication, count of the profile links in a particular publication is fetched first and then the links are compared with the links fetched from the profile page to identify if the publication has a co-author who is currently a faculty of school of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

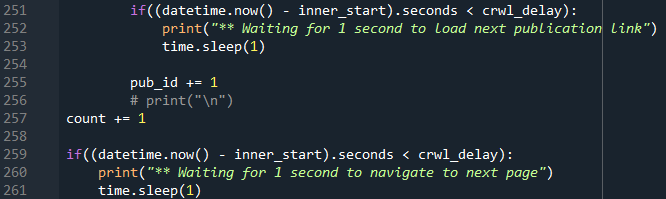
If the conditions are not satisfied, id/key of the publication is stored in “pub\_skipped” variable for future references.



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.

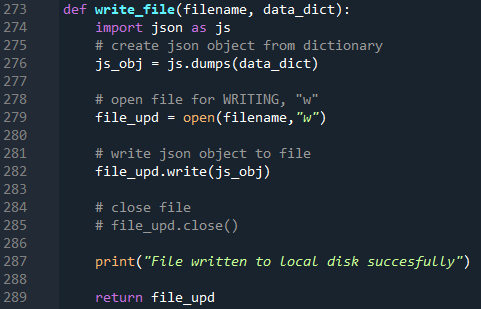


At the end of each “for” loop, script verifies the if the crawl delay specified in the “robots.txt” file is obeyed before hitting/sending GET request to the next url, as shown in the below code snippet.



### **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.





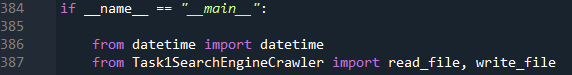


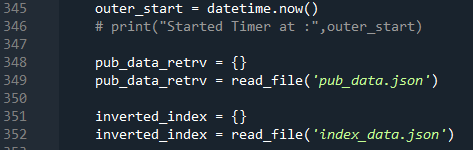
## 2. Steps to process the query and find the most relevant publication

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.

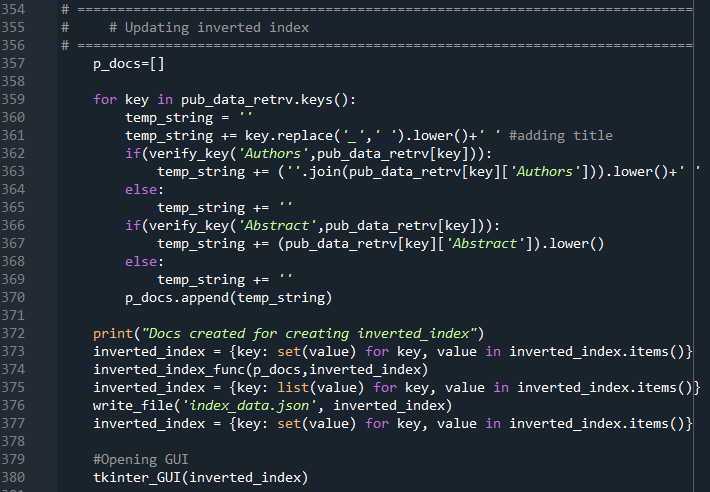






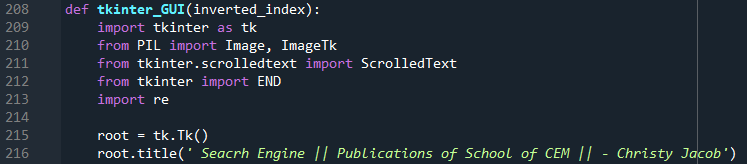


After reading the data from the pub\_data.json file, **inverted index is updated**, so that any new values in the pub\_data will be added/ created in the index\_data.json file



### **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.

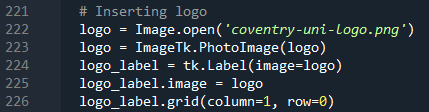


### **Step 3**: Creating a basic GUI for the search engine

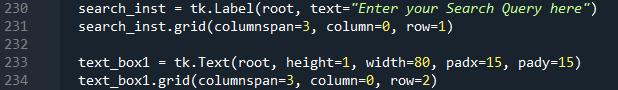
Before adding any GUI element, first step is to create a canvas to place the GUI components.



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.



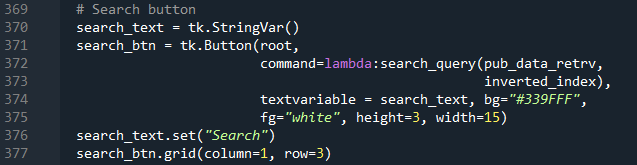
To make the GUI more attractive and user friendly, we can also add text and text\_boxes to enter the query and display the results.



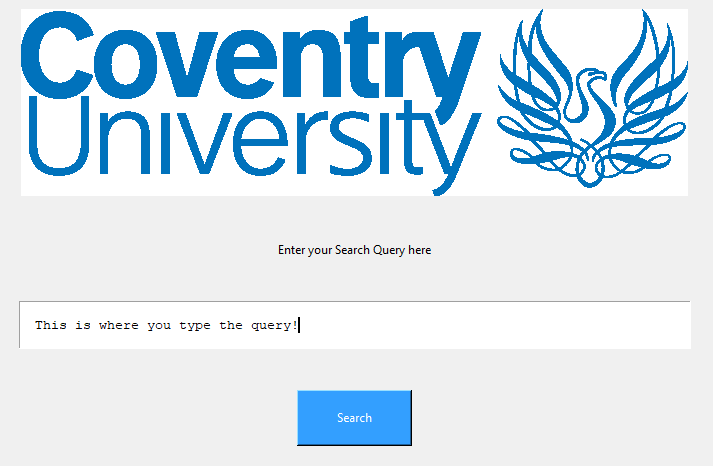
ScrolledText() method is used to create a text box instead of Text() function because, ScrolledText() automatically creates a scroll bar if the content exceeds the text\_box line count.



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

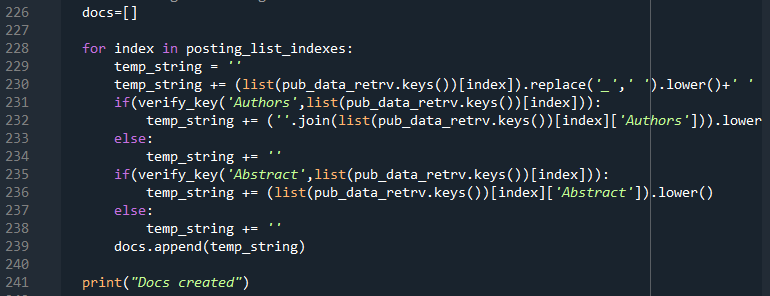


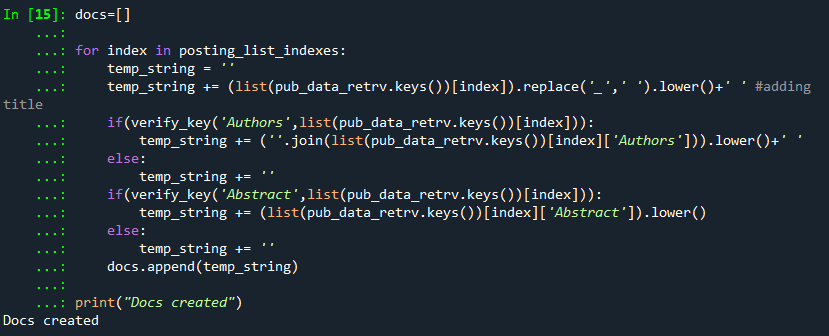
Below displayed is a screenshot of the GUI of the search engine before starting the search.



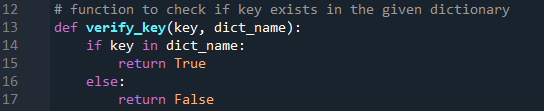
### **Step 4**: Processing the crawled data to construct inverted index

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.

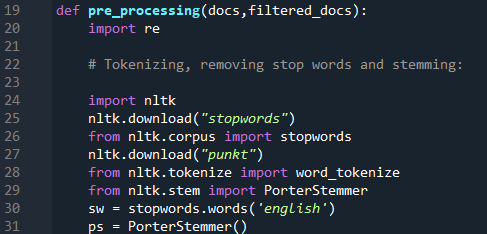




As abstract can be null, we need to verify if the key exists in the database for that publication, in-order to avoid errors.

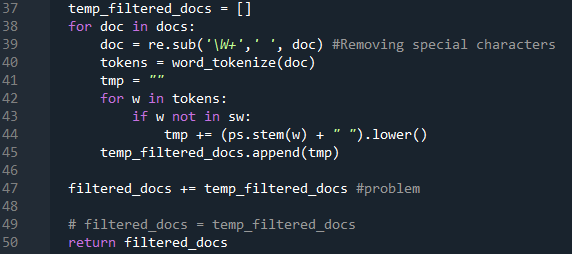


Values of column title, author names and abstract are combined to form a single document for each publication as shown in the above code snippet. These docs must be pre-processed before creating inverted index. In pre-processing, each doc is tokenized, stemmed, and stop words are removed. First, we must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



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

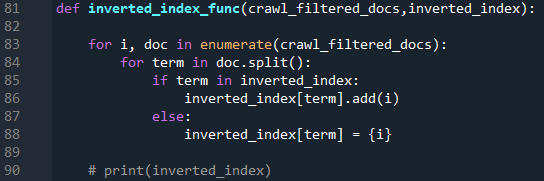
pre\_processing() method returns the filtered doc and stored in another variable which is passed as one of the arguments to calculate inverted index.





### **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.



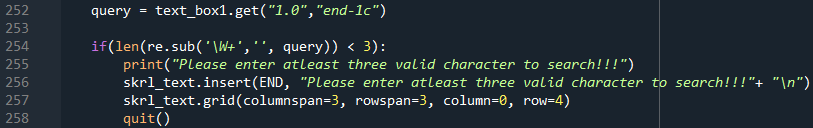


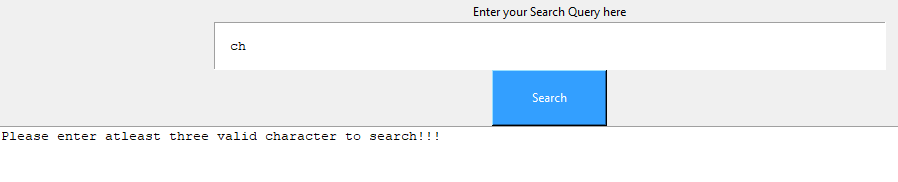
Updated inverted index is updated/written back to the database/file.



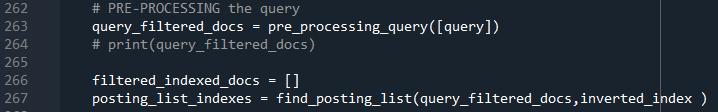
### **Step 6**: Fetching the query and pre-processing the query

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.

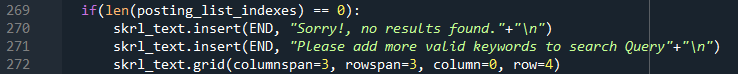


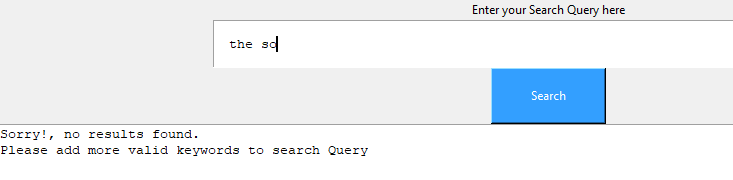


If the query is valid, it is pre-processed to find the posting list indexes.

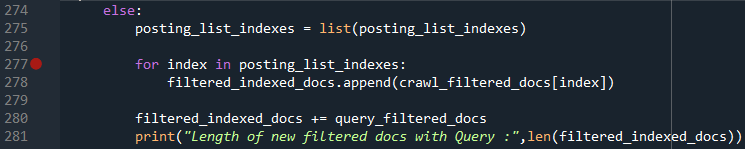


Next, the script searches if the pre-processed query has a posting list in the inverted index dictionary created. If it doesn’t find the posting list, another warning message is displayed, asking user to enter more keywords to get results.





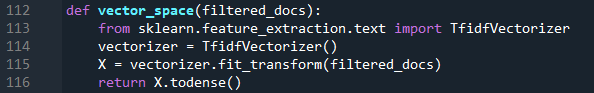
If the query is valid, it adds the filtered query with the filtered data fetched with respect to the posting list.

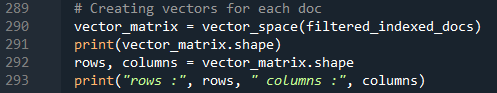


### **Step 7**: Performing Ranked retrieval using Vector Space Model

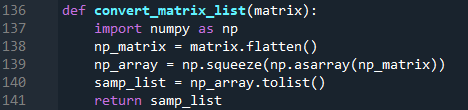
Filtered docs are converted to vectors using TfidfVectorizer library and the vector matrix is converted to list to calculate the rankings (cosine values) of each document.







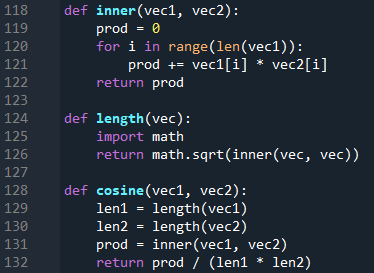


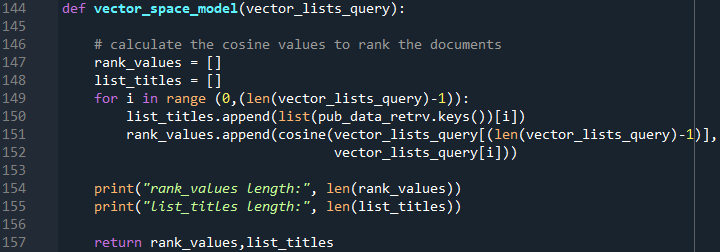




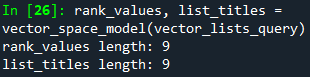


The rankings are then converted to a dataframe and sorted in descending order to display the search results, so that the most relevant documents (having greater cosine values) are listed first.

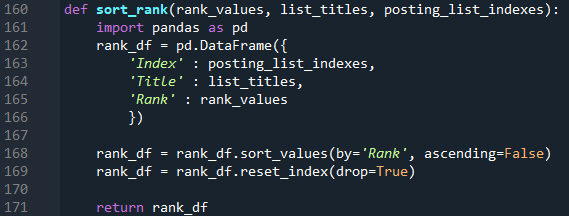








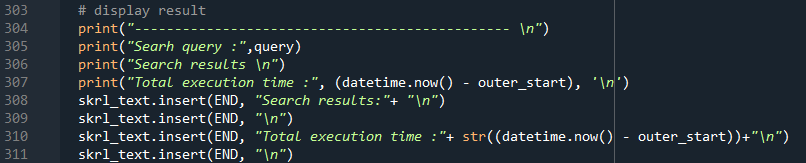
Creating and sorting pandas dataframe

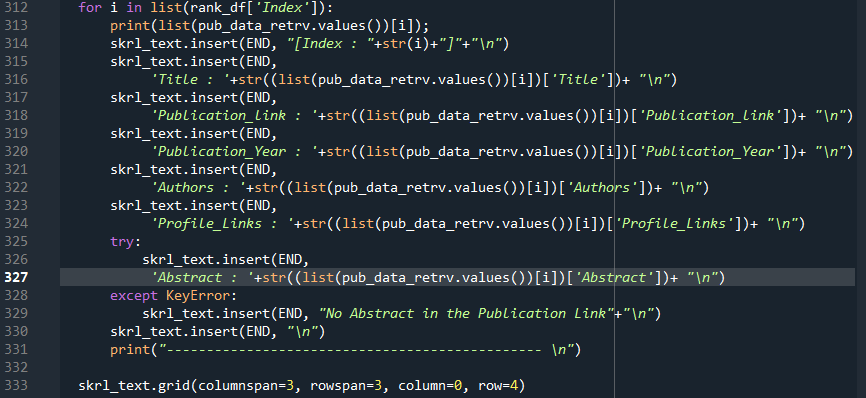


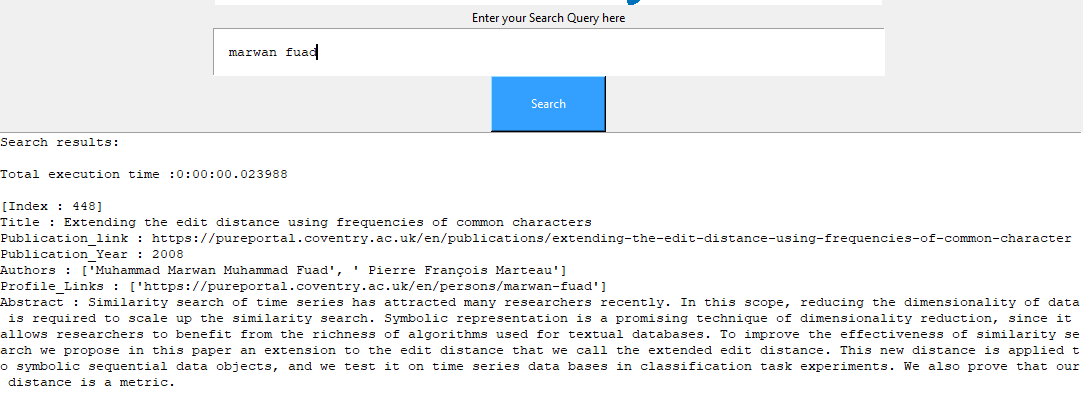


### **Step 8**: Displaying the results

Result is displayed in the scroll text field of tkinter

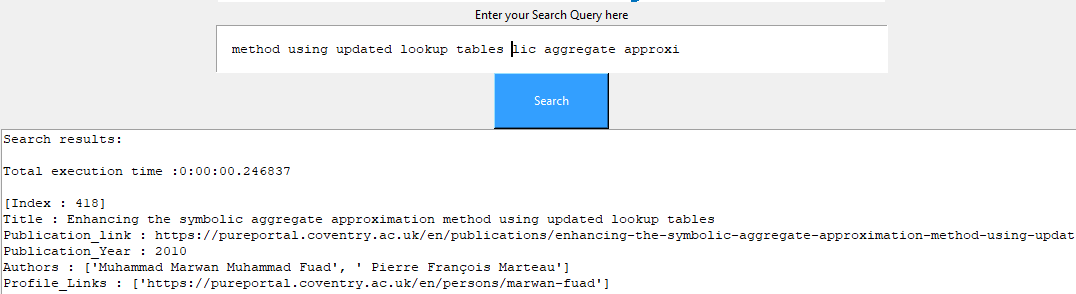


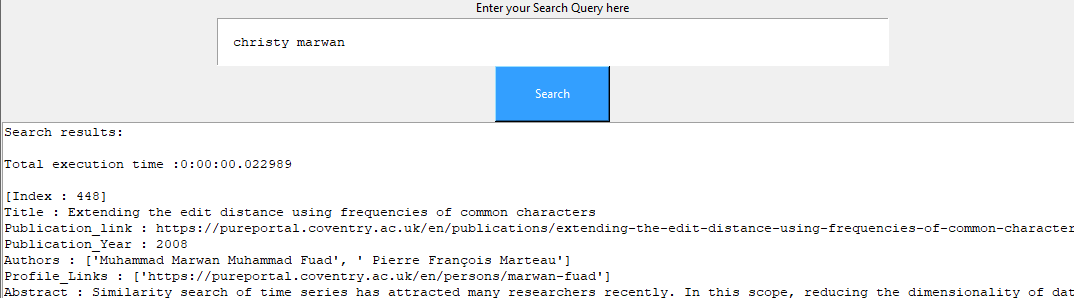


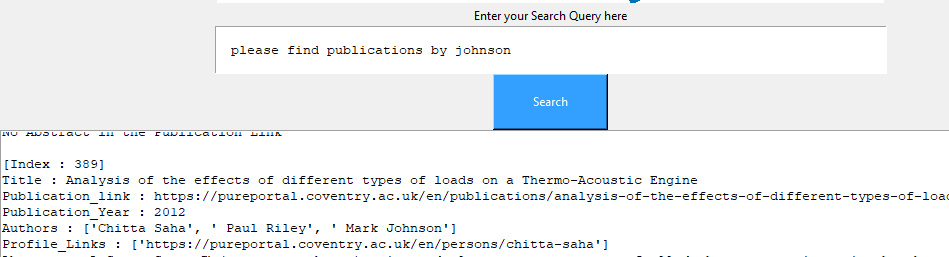


### **Step 9**: Checking the efficiency of the search engine by checking the search time

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





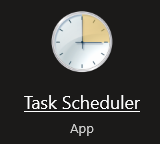




## 3. Scheduling the crawling script to run every week to fetch the data and update the database/file in local repository

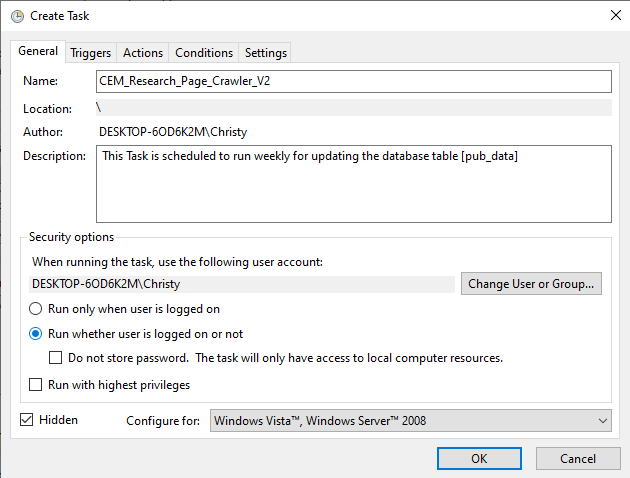
As the crawling and query processing scripts are different, we can schedule the crawling script to execute every week or daily to fetch and update the database/file.

I will be using Windows Task scheduler to schedule the runs



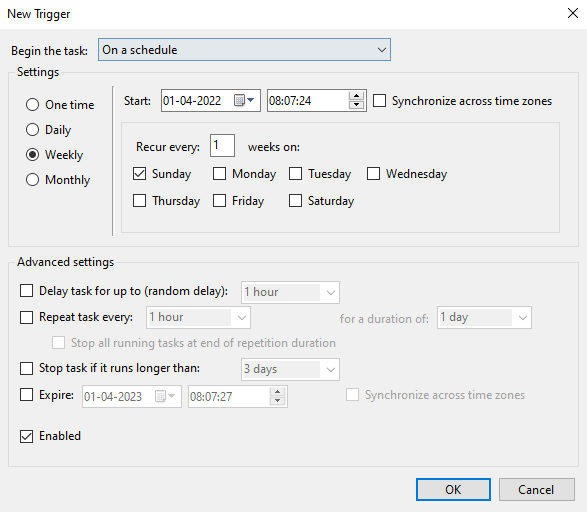
### **Step 1**: Create a new task

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



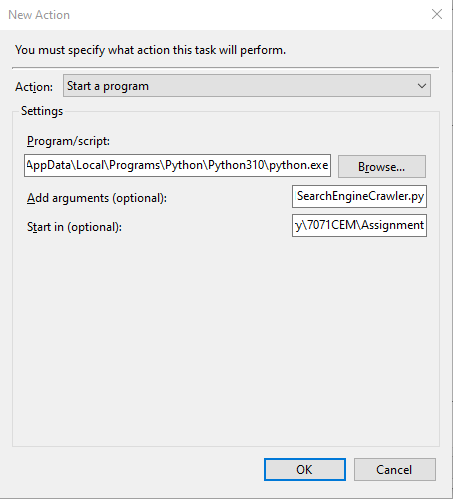
### **Step 3**: Creating new “Action”

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

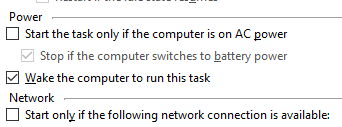
**Add Arguments** will be the file name with .py extension

**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

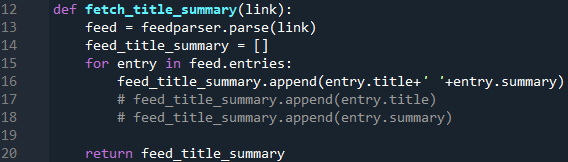
In this task I have created a script that can fetch any amount of data from the rss pages using “feedparser” library and performs document clustering using k-means algorithm.

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.

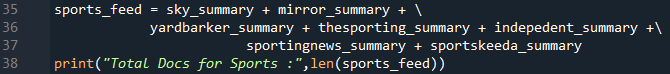




## **Step 2**: Extracting feeds for Sports, Science and Business categories

Sports:







Business:







Science:

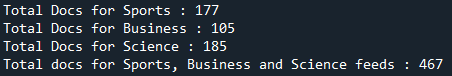






Total documents taken:





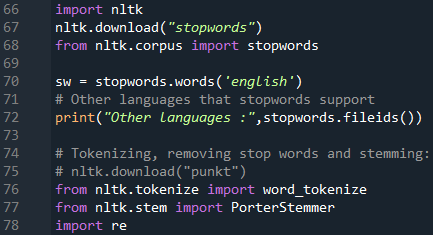
## **Step 3**: Reading Data through .txt file

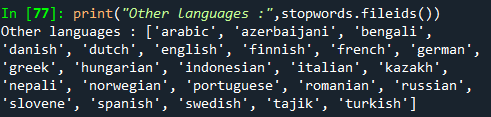
As the docs fetched using feedparser didn’t give me a good purity score, I extracted the data manually from various data and combined it to a .txt file manually and feed the same to the KMeans algorithm.

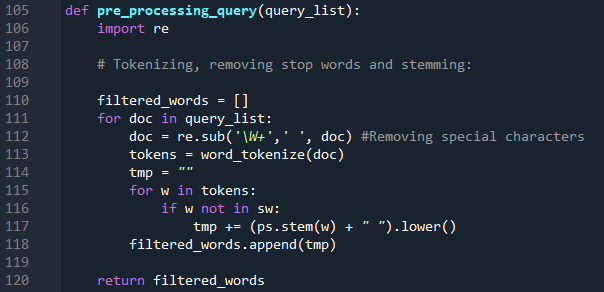


## **Step 4**: Pre-Processing the data fetched

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

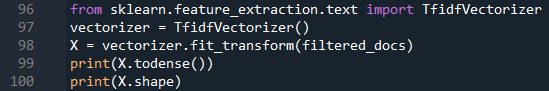






## **Step 5**: Vectorizing the pre-processed data

Constructing vectors using TfidfVectorizer



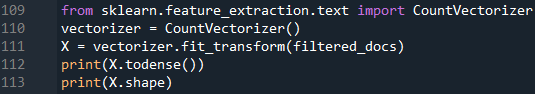


Creating a dataframe to look at the words in the matrix. This gives us an idea about the words present in the matrix.





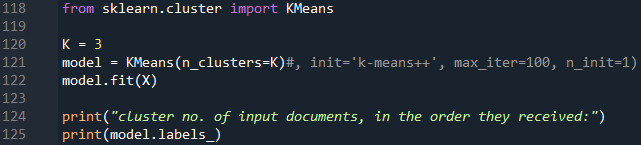
Constructing vectors using CountVectorizer



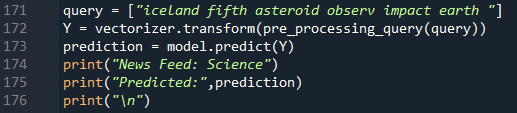


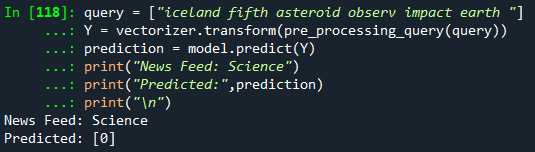
## **Step 6**: Creating a model for KMeans

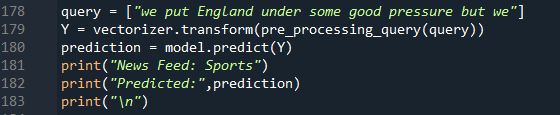
Creating KMeans cluster using Sklearn library. Number of clusters taken in 3(k) and vectorizer ised is **TfidfVectorizer**.

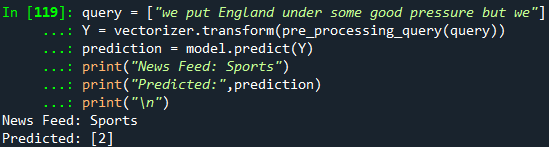


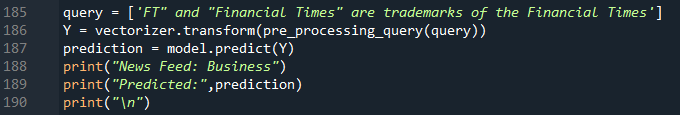
## **Step 7**: Predicting for new queries/docs

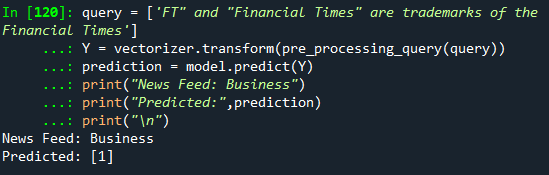






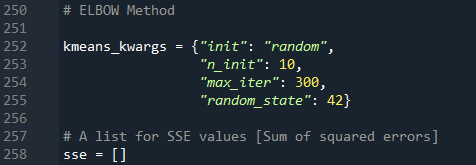


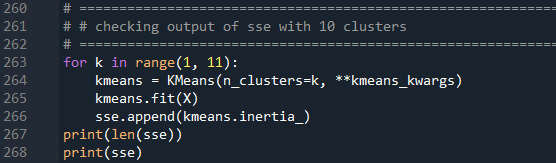


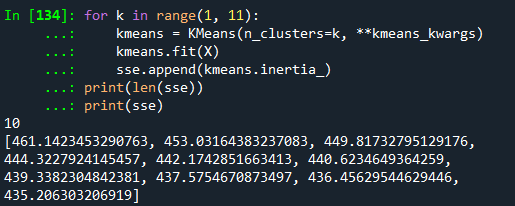


## **Step 8**: Using Elbow method to determine the number of clusters (Real Python, n.d.)

Determining the k value using 10 clusters

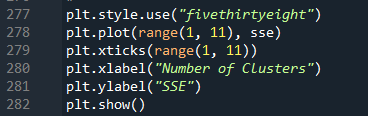


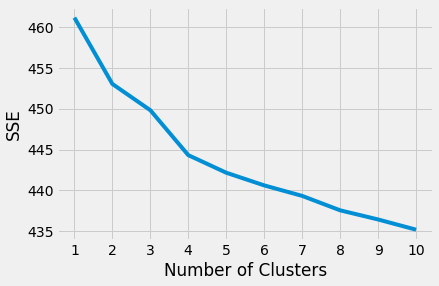


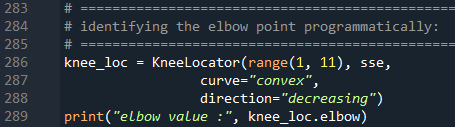


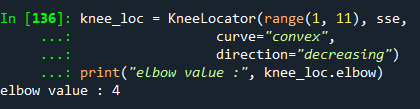
Plotting elbow graph using matplotlib











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.







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 Ф44 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.cms")

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%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))

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/>