(7071CEM)

Course Work
Information Retrieval

Module Leader: Seyed Mousavi

Student Name: Jyothi Yendamuri

SID: 11467683

Table of Contents:

Table of Figures:

Introduction:

Information retrieval (IR) is a process of extracting the most relevant data from semi-structured text documents based on using keywords supplied by users within an acceptable time. Because it is the root of software that allows literature search, the heart of a web search engine is feature or data extraction technology, which plays a significant part in biomedical research (Rohit,2021). Google vertical Search engine is the best and well-known example of information retrieval. Different components of information retrieval are indexer, crawler, query processor. Document representation, query representation, and framework to match and build a link between document and query representation are three important aspects of information retrieval. In this coursework two tasks are given. one is vertical search engine and another one is document clustering. The search operation for many sorts of keywords is effectively carried out using the web crawling technology. BFS algorithm determines the shortest route, allowing the search to be completed rapidly.

The second task is to determine the clustering procedure, for which we have selected technology as the category. The clustering procedure is performed with the aid of the K-means algorithm.

Aim:

The main goal of this research project is to better understand the information gathering process, which is demonstrated by a Coventry University School of Life Science (SLS) member, doing an experiment to retrieve a suitable paper/journal.

Tasks:

There are two tasks given in this coursework

1.Vertical Search Engine
2. Document Clustering

Programming Language:

In this coursework, Python language is used to complete the tasks. Python is one of the most widely used programming languages in the industry and has displaced several other computer programming languages. Python 3.8.8 version has been used.

Task 1: Vertical Search Engine

Search Engine:

A search engine is created to identify items in a repository or database that match terms or words entered by the user and is especially useful for discovering certain websites on the Internet. Search engine still widely used search engine on the planet. Search engine works through using crawling, indexing and ranking functions.

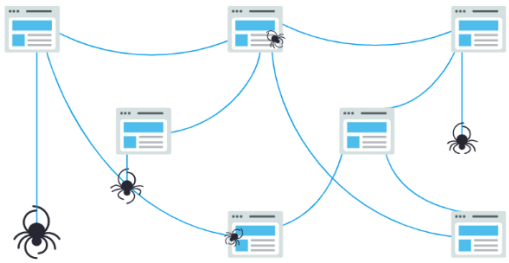

Figure 1: Bots fetching the webpages and links to find new url’s

Vertical Search Engine:

Vertical search engine is a most popular search engine, it purely concentrates on specific type of online material. Users can use vertical search engines to look for information within a subset of documents that are related to a specific topic. Building a Web crawler that differentiates relevant content from non-related documents is one of the most difficult aspects of establishing a vertical search engine (Pedro,2008). The vertical information zone can be organised by topic, media type, or content category. Ecommerce, online automobile sector, legal information, medical history, academic research, job hunting, and tourism are all common verticals.

In this report vertical search engine is created similar to google scholar to get the research papers or journals published by a member of the school of life sciences (SLS) at Coventry university. To create the vertical search engine crawler, indexer, query processor components are used.

1. Crawler:

A web crawler, also known as a spider or a search engine bot, gathers and analyses material from all across the web. The main purpose of a bot is to learn what each web site on the internet is about, so that information can be obtained when needed. Crawling is the scientific term for automatically visiting a website and getting information using a software program, which is why they're called "web crawlers.

The goal of a crawler in vertical search engine is to find more extremely necessary documents before unrelated sites, in order to reduce the amount of index time. Our usual or default crawling technique is breadth-first, in which we scan a set of pre-determined number of links from the first sites. Links authorize us to index connections from unrelated pages in the hopes of finding clusters of relevant pages that might otherwise not be reached.

Here search will be performed for the author who is a member of School of Life Sciences at Coventry University and the other search is performed for the papers. first our text should be vectorized then it should be stored. initially all required libraries are imported.

Beautiful Soup: “Beautiful soup” python library is used for getting the information from documents. Beautiful Soup is a Python tool for parsing and extract information from HTML and XML files. It connects with your chosen parser to provide fluent navigation, search, and modification of the parse tree.

The crawler will be called using the search key sent from the user after the depth of the search URL has been verified by the python script. The method parse will be used to parse the content of the URL.

We can see the below screen shot main execution starts from here. It is asking the user to he want to run the crawler or not. If user enter 1 then it run the crawler.

```
crawler_input=int(input("Press 1 to run crawler and 2 to continue from saved output file"))
if (crawler_input==1):
    webcrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif (crawler_input==2):
    if(os.path.exists("output.csv")):
        print("Search started from initially saved Crawler optput")
    else:
        print("You need to crawler first.")
else:
      print("invalid response.")
```

```
Press 1 to run crawler and 2 to continue from saved output file 1
```

Figure 2: Running the crawler

If user do not want to run the crawler then he will press 2. Then searching will be taken from the saved output file ” output.csv”.

```
: crawler_input=int(input("Press 1 to run crawler and 2 to continue from saved output file"))
if (crawler_input==1):
    webcrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif (crawler_input==2):
    if(os.path.exists("output.csv")):
        print("Search started from initially saved Crawler optput")
    else:
        print("You need to crawler first.")
else:
      print("invalid response.")
```

```
Press 1 to run crawler and 2 to continue from saved output file2
Search started from initially saved Crawler optput
```

Figure 3: Searching continued from saved output file

Suppose if I need paper from physical activity, if I do not run the crawler my search will take from this (ouput.csv) file. If I run the crawler this file will be created again. In order to make this file we must run the crawler one time. if crawler will not run one time then this file will not be created. So, we need to run the file in order to save the file.

If user enter 3 then it will show below response.

```
In [*]: crawler_input=int(input("Press 1 to run crawler and 2 to continue from saved output file"))
if (crawler_input==1):
    webcrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif (crawler_input==2):
    if(os.path.exists("output.csv")):
        print("Search started from initially saved Crawler optput")
    else:
        print("You need to crawler first.")
else:
      print("invalid response.")
```

```
Press 1 to run crawler and 2 to continue from saved output file 3
```

Figure 4: Testing crawler to given invalid number

```
: crawler_input=int(input("Press 1 to run crawler and 2 to continue from saved output file"))
if (crawler_input==1):
    webcrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif (crawler_input==2):
    if(os.path.exists("output.csv")):
        print("Search started from initially saved Crawler optput")
    else:
        print("You need to crawler first.")
else:
      print("invalid response.")
```

```
Press 1 to run crawler and 2 to continue from saved output file3
invalid response.
```

Figure 5: Invalid response

Its show's invalid response. Because user either enter one or two to run the crawler and search the information form saved output file. If we delete the output file and then run it information about all these tags taken from the website.

https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences/publications

After clicking the site we ask the crawler to go to the publications. Here many of them not from Coventry university. So we are fetching only member of the school of life sciences at Coventry university.

These webpages are designed in html. So i want to tell my crawler to read the all information.

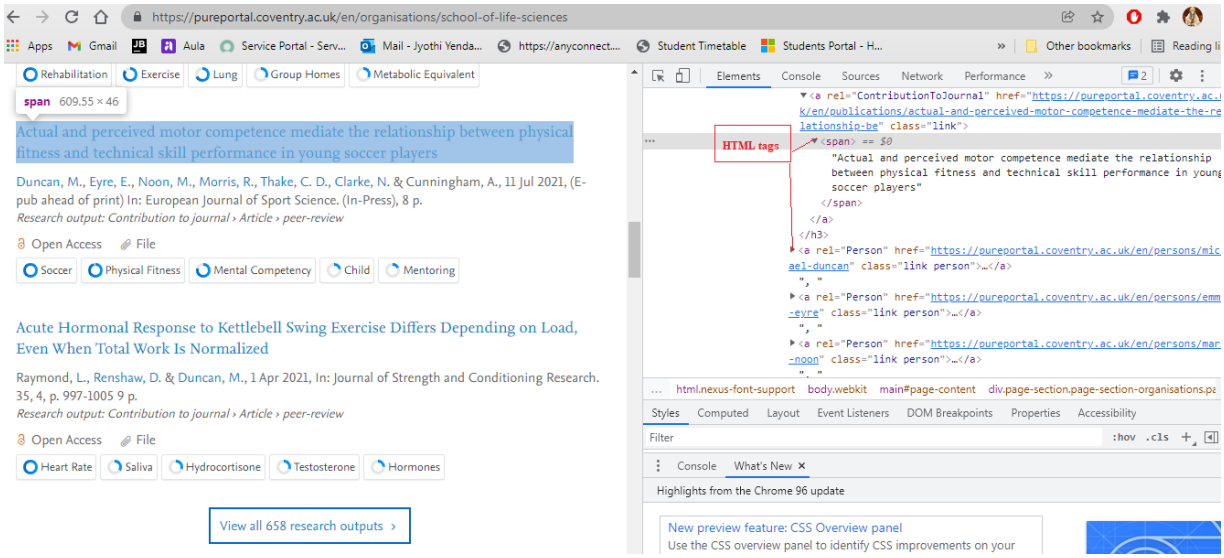

Figure 6: Website tags information

So, using this information crawler will read all these tags.

After searching the information next we will go to the next page. similarly, we will tell the crawler to after reading the information go to the next button. It will keep on continue searching information, saving output file, go to next page until find the search result. When if not find the search button then code will be stopped.

The search function mainly used to search the keywords to ensure the result is accurate. The needed fields are paper's title, author name of the paper, published year, and URL link to access the paper's information and link to view the author's details will be displayed.

Breadth first search:

BFS (breadth-first search) is a technique for visualizing data, searching networks, and traversing structures. The method examines and identifies all the important nodes in a network in a fast and accurate breadthwise manner. This approach explores all nodes nearby to a single node in a network. Keep in mind that BFS connects to each of these nodes one at a time. The BFS algorithm determines the shortest route, allowing the search to be completed rapidly. The search is carried out with the help of the Breadth First Search Algorithm (BFS).

This technique improves the discovery of the quickest route in a network, allowing for a speedy search when a keyword is provided. The BFS begins by searching the initial keyword before moving on to the neighbouring nodes. The BFS works in a structured manner, beginning with one of the vertices on the fringe. The vertex is chosen based on the vertex that was most recently encountered. This feature keeps the entire origin's priority queue open, making its actions less space efficient.

The BFS implementation involves identifying the sample's vertices as visited or not visited. The procedure must be repeated until the queue is shortened. During the keyword search process, this action allows you to identify the quickest and shortest path

2. Indexer

Indexer is an essential method in Information Retrieval (IR) technology. Because it is the initial stage in information retrieval and helps in effective manner, it is the best- and well-known characteristic of the IR process. Indexing summarises documents to their most important words. Information specification, tokenization of documents, processing of document keywords, and index creation are the four steps of indexing. The index can be stored as a direct index, inverted index and document index are data structures. Indexing is the process of organising content or collection that may be retrieved quickly.

The indexer must determine if it is upgrading existing items or creating new items. The data collecting policy usually coincides with traversal.

Inverted Index:

Inverted index has been implemented for this task. An inverted index is a technique that compiles a collection of terms or words as well as references to documents that include those elements. Then, through a process referred as tokenisation, search engines compress words to their main message, lowering the number of resources required to store and retrieve the information.

Data Pre-processing:

First, we need to clear duplicates in order to improve the search. The following steps done to prepare the data.

2.1 Lowercase:

Lower is a built-in Python function for dealing with strings. The .lower() function accepts no parameters and converts every capital letter to lowercase to produce lowercased characters from the provided input. It returns the exact string if the input string contains no uppercase characters. here we used the lower () python function to display the title of the papers and all sentences in lowercase format. This is the best- and well-known function to convert the string or characters to lowercase.

2.2 Tokenization:

Tokenization is the primary step in every text-processing application in Python. This could be used for statistical analysis, parsing, spell-checking, counting, and database building, among other things. Tokenizing data simply implies dividing the text's content. The MIT licence applies to the tokenizer. Methods implemented in tokenization are split function, regular function using NLTK, spaCy library among others. In this task we used word tokenize function to split the sentence into tokens as required.

2.3 Remove Punctuations:

This remove punctuation technique must be used with the string. To remove all punctuations from the string this technique will be used. Here first string of punctuation is defined. Then if function is used to check if string has no punctuations then it will print the string. else if string has punctuations it removes the punctuations from string using no punctuation method in the program. Finally, cleaned up data will be displayed.

2.4 Removal of Stop Words:

We imported stop words method to delete the no stop words during query processing. The output list is empty if all the query phrases are eliminated during stop word processing. When all the query words are stop words, stop word removal is deactivated to guarantee that search results are delivered. Here sentences are filtered clearly to check the stop words. To install the package, we can use pip install stop words.

After applying all pre processing steps we get the below results:

```
72]: #Applying Pre-processing steps
import re
import string
processed_journal = []
for t in df['Journal name']:
    # Removing Unicode
    clean_title = re.sub(r'[^\x00-\x7F]+', '', t)
    # Removing Mentions
    clean_title = re.sub(r'@\w+', '', clean_title)
    # Converting into lowercase
    clean_title = clean_title.lower()
    # Removing punctuations
    clean_title = re.sub(r'[%s]' % re.escape(string.punctuation), '', clean_title)
    #removing stop words
    stop = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s*')
    text = stop.sub('', clean_title)
    processed_journal.append(text)

print(processed_journal)
```

```
['comparison physical activity homebased centrebased pulmonary rehabilitation randomised controlled secondary analysis', 'act
ual perceived motor competence mediate relationship physical fitness technical skill performance young soccer players', 'acut
e hormonal response kettlebell swing exercise differs depending load even total work normalized', 'associations selfreported
sleep duration mortality employed individuals systematic review metaanalysis', 'blood cancer care resource limited setting co
vid19 outbreak single center experience sri lanka', 'british soccer academy personnel perceive psychological technicaltactica
l attributes important contributors development', 'changes joint kinematics dynamic postural stability free restricted arm mo
vements children', 'comparative analysis reported physical activity leisure centres members versus general population spain',
'crosscultural comparison fundamental movement skills 9 10 year old children england china', 'delivery approaches within exer
cise referral schemes survey current practice england', 'developing powerful athletes part 2 practical applications', 'resist
ed sled towing improve physical qualities elite youth soccer players differing maturity status', 'dose response effects bwf s
huttle time programme childrens actual perceived fundamental movement skill competence', 'effectiveness structured physical a
ctivity interventions evaluation physical activity levels adoption retention maintenance adherence rates systematic review me
taanalysis', 'effect barbell load vertical jump landing forcetime characteristics', 'effect sex fatigue quiet standing dynami
```

Figure 7: Applied all pre-processing steps

2.5 Porter Stemming:

The Stemming will removes the suffixes from an Original sentence to retrieve its stem, which is extremely important in the field of information retrieval (IR).This was introduced in 1980 and it is most popular stemming.

```
#Stemming and removing stop words
preprocessed_corpus=[]
for i in processed_journal:
    token_words=word_tokenize(i)
    stem_words=[]

    for word in token_words:

        if word not in stopwords.words("english"):

            if(word.isalpha()):

                stemmer = PorterStemmer()
                stem_words.append(stemmer.stem(word))
                doc=' '.join(stem_words)

    preprocessed_corpus.append(doc)

print(preprocessed_corpus)
```

```
['comparison physic activ homebas centrebas pulmonari rehabilit randomis control secondari analysi', 'actual perceiv motor co
mpet mediat relationship physic fit technic skill perform young soccer player', 'acut hormon respons kettlebel swing exercis
differ depend load even total work normal', 'associ selfreport sleep durat mortal employ individu systemat review metaanalys
i', 'blood cancer care resourc limit set outbreak singl center experi sri lanka', 'british soccer academi personnel perceiv p
sycholog technicaltact attribut import contributor develop', 'chang joint kinemat dynam postur stabil free restrict arm movem
ent children', 'compar analysi report physic activ leisur centr member versu gener popul spain', 'crosscultur comparison fund
ament movement skill year old children england china', 'deliveri approach within exercis referr scheme survey current practic
england', 'develop power athlet part practic applic', 'resist sled tow improv physic qualiti elit youth soccer player differ
matur statu', 'dose respons effect bwf shuttl time programm children actual perceiv fundament movement skill compet', 'effect
structur physic activ intervent evalu physic activ level adopt retent mainten adher rate systemat review metaanalysi', 'effec
t barbel load vertic jump land forcetim characterist', 'effect sex fatigu quiet stand dynam balanc lower extrem muscl stiff',
'enhanc implement sustain fundament movement skill intervent uk ireland lesson collect intellig engag stakehold', 'fundament
movement skill profici among british primari school children analysi behavior compon level', 'fundament movement skill percei
v compet fit key factor associ technic skill perform boy play grassroot soccer', 'habitu caffein consumpt affect ergogen coff
```

Figure 8: Applied stemming and removing the stop words

2.6 TF-IDF:

There are many numbers of words in a document however that appear frequently but aren't necessarily significant. Different terms in English like "the," "is," "of," and so on. We might add these terms to a list of stop words and remove them before analysing the documents. A list of stop words are not particularly sophisticated way of regulating term frequency for frequently used terms.

Inverse Document Frequency: Inverse document frequency (idf) reduces the weight of often used terms while increasing the weight of less frequently used words in a collection of documents. The tf-idf statistic is used to determine the importance of a word in a collection of documents, such as a single story in a collection of stories or a single website in a collection of websites.

After applying the tf idf we get the below result:

```
tfidf
```

	0	1	2	3	4	5	6	7	8	9	...	503	504	505	506	507	508	509	510	511	512
abil	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
absolut	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
abund	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
academ	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
academi	0.0	0.000000	0.0	0.0	0.0	0.243868	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
young	0.0	0.276335	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
youth	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
zimbabw	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
zr	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
zumba	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1853 rows × 513 columns

Figure 9: TF-Idf results screenshot

Here we have 1853 rows and 513 columns. 0 to 512 columns represent the papers. We can see the above output those are the 1853 words came from document.

3. Query Processor:

A query processor is one of the most important parts of a relational database, in which information is recorded in rows and tables.

In query processor we are expecting query from the user and it pre-processed. Then it is converted into a vector. So we can perform high similarity. Example here we are searching the keywords like" comparison physical activity "then it converted into vector.

```
for word in token_words:

    if word not in stopwords.words("english"):

        if(word.isalpha()):

            stemmer = PorterStemmer()
            stem_words.append(stemmer.stem(word))
            doc=' '.join(stem_words)

preprocessed_query.append(doc)

print(preprocessed_query)
#q1 = [q1]
q_vec = vectorizer.transform(preprocessed_query).toarray().reshape(tfidf.shape[0],)
print(q_vec.shape)
```

```
Enter topic for searching comparison physical activity
```

Figure 10: Searching Keyword

```
q1 = input("Enter topic for searching")
preprocessed_query=[]
#for i in q1:

token_words=word_tokenize(q1.lower())
stem_words=[]

for word in token_words:

    if word not in stopwords.words("english"):

        if(word.isalpha()):

            stemmer = PorterStemmer()
            stem_words.append(stemmer.stem(word))
            doc=' '.join(stem_words)

preprocessed_query.append(doc)

print(preprocessed_query)
#q1 = [q1]
q_vec = vectorizer.transform(preprocessed_query).toarray().reshape(tfidf.shape[0],)
print(q_vec.shape)
```

```
Enter topic for searchingcomparison physical activity
['comparison physic activ']
(1853,)
```

Figure 11: Searched Results are displayed and vectorised

Finally, my query is vectorised and matching the all vector.

Next, we will check the cosine similarity it will compare my query vector to all my existing vector till the tf-idf model.

```
def calc_cosine(v1,v2): # receives two vectors of the same length as lists and returns their cosine similarity
    return calc_inner(v1,v2) / (calc_length(v1)*calc_length(v2))
v0 = q_vec

k=[]
for v in range(0,512):
    x=calc_cosine(v0, tfidf[v])
    if(x>0.0):

        k.append(x)


print(k)
```

```
[0.38887508636750034, 0.09584433279605527, 0.16319551395037285, 0.21566571978510868, 0.09221078756515483, 0.3045337717362575,
0.17487546447825975, 0.10629034130721622, 0.1765362400756082, 0.1871839565458115, 0.11651112481040807, 0.24619392434061407, 0.1
00684237373845542, 0.14526144351945855, 0.2145268357872224, 0.17346810879654267, 0.08168152844316139, 0.17498361608546628, 0.128
56684438300026, 0.20222272020987245, 0.1919104518911627, 0.07404220666557465, 0.15832731226567187, 0.16976420261346623, 0.17656
46361800349, 0.2930242062766063, 0.2459087899887641, 0.11264969219775658, 0.1792038820243797, 0.16659307151031436, 0.1835383701
7419095, 0.16321150476375093, 0.10962407081388544, 0.24629546795127855, 0.08308751054065479, 0.07717584103759487, 0.22768949363
88438, 0.267839231025679, 0.09764468408660867, 0.16236956730419969, 0.06551263168179312, 0.11378769576474856, 0.357287066820040
2, 0.18973013578632947, 0.16876196381236971, 0.15118954295867892, 0.09068890612346987, 0.18004333030844064, 0.1524907098221987
3, 0.06742787426982882, 0.19039278693312683, 0.06777054588104885, 0.19709922627131, 0.1862128455789979, 0.07766208818723132, 0.
0976964875254054, 0.16393604375730095, 0.07163729287999093, 0.16399258466967126, 0.17941728417214128, 0.10664549603665306, 0.24
173731654155578, 0.24096038333476721, 0.19070271736340325, 0.07581988415863987, 0.07680507999888712, 0.26237457627754623, 0.218
38322480638886, 0.1346341017541538, 0.21829857206604442, 0.07225839489343915, 0.08629030065520518, 0.07484952631410524, 0.24943
512998719708, 0.2116976497850543, 0.10781461501267861, 0.08443888049666277, 0.07780392131029912, 0.07394829266930722, 0.0746333
7627055491, 0.09736182266989192, 0.08799954859315251, 0.06926867257449867]
```

Figure 12: Checking similarity

Here, all similarity results are displaying till 512 papers. If we don't want too many papers used enumerate function to display all papers an indexing order. Below screenshot shows the enumerate function and 20 papers are displayed in indexing order.

```
a= (sorted( [(x,i) for (i,x) in enumerate(k)], reverse=True ) [:20] )
print(a)
```

```
[(1.0000000000000002, 0), (0.47848346224252214, 37), (0.3242140591101352, 87), (0.306858926770661, 71), (0.2619453277870722, 7
7), (0.25079253949255464, 76), (0.2145795209328353, 75), (0.19776952184532362, 66), (0.17355043959158575, 47), (0.1631349975525
7435, 43), (0.1627991006870629, 28), (0.1507642477052811, 35), (0.14163577233611127, 2), (0.13967265236354912, 38), (0.13894003
896763393, 52), (0.12999391885641778, 49), (0.11851259626062023, 64), (0.11842559678575772, 5), (0.11394981352358347, 30), (0.1
0649529753332114, 32)]
```

Figure 13: Used enumerate function to reverse the order

If the search keyword is found then it displays all author information like Title of the paper, link, date of the paper, author name, link to author profiles.

```
if(a==[]):
    print("No related journal found")
else:
    for i in range(len(a)):
        x=a[i][1]

        print("Title of Paper:",df.iloc[x,2])
        print("Link of Paper:",df.iloc[x,3])
        print("Date of Paper Published:",df.iloc[x,4])
        print("Author names:",df.iloc[x,0])
        print("Link to Author Profiles:",df.iloc[x,1])
        print("----------------------------------------")
```

```
Title of Paper: A Comparison of Physical Activity Between Home-Based and Centre-Based Pulmonary Rehabilitation: A Randomised
Controlled Secondary Analysis
Link of Paper: https://pureportal.coventry.ac.uk/en/publications/a-comparison-of-physical-activity-between-home-based-and-cen
tre-b
Date of Paper Published: 26 Oct 2021
Author names: ['Horton, E.', 'Sewell, L.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/elizabeth-horton', 'https://pureportal.coventry.ac.u
k/en/persons/louise-sewall']
----------------------------------------
Title of Paper: The transitioning feature between uncooked and cooked cowpea seeds studied by the mechanical compression test
Link of Paper: https://pureportal.coventry.ac.uk/en/publications/the-transitioning-feature-between-uncooked-and-cooked-cowpea
-seed
Date of Paper Published: Mar 2021
Author names: ['Munialo, C. D.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/claire-munialo']
----------------------------------------
Title of Paper: British Soccer Academy Personnel Perceive Psychological and Technical/Tactical Attributes as the Most Importa
nt Contributors to Development
Link of Paper: https://pureportal.coventry.ac.uk/en/publications/british-soccer-academy-personnel-perceive-psychological-and-
```


Figure 14: Search results

Finally, checking if the given words/keywords are not match what will happen. Suppose user given a unrelated word then it will print the “No related Journal Found”.

If we search unrelated word it will show “No related journal found”.

```
if(a==[]):
    print("No related journal found")
else:
    for i in range(len(a)):
        x=a[i][1]

        print("Title of Paper:",df.iloc[x,2])
        print("Link of Paper:",df.iloc[x,3])
        print("Date of Paper Published:",df.iloc[x,4])
        print("Author names:",df.iloc[x,0])
        print("Link to Author Profiles:",df.iloc[x,1])
        print("----------------------------------------")
```

```
No related journal found
```

Figure 15: No related search Found screenshot

Task -2: Document Clustering:

Document clustering is the process of arranging documents into different clusters. The unsupervised categorization of documents into groups is known as document clustering in which papers within a cluster are similar, but documents with in separate clusters are different. Web browsers, online articles, media articles, and other text documents are examples of documents.

Here we are reading the information from BBC websites. Three websites (Politics,Education,Science) are mainly choosing to read the information. read parser function is used to read the headlines from chosen websites. After running the code, we get the 144 news headlines.

```
import feedparser
text=[]

Politics = feedparser.parse("http://feeds.bbci.co.uk/news/politics/rss.xml")
Education = feedparser.parse("http://feeds.bbci.co.uk/news/education/rss.xml")
Science = feedparser.parse("http://feeds.bbci.co.uk/news/science_and_environment/rss.xml")

for i in Politics.entries:
    x= i.summary
    text.append(x)

for i in Education.entries:
    x= i.summary
    text.append(x)

for i in Science.entries:
    x= i.summary
    text.append(x)

print(len(text))
print(text)
```

```
144
['The Labour MP accuses the government of stoking a culture war that will harm anyone "slightly different".', 'The south-eas
t London election was prompted by the death of former MP James Brokenshire.', 'But government ministers are accused of givin
g mixed messages on events over the festive period.', 'Alex Bourne says the former health secretary had no involvement is hi
s company winning NHS work.', 'Grenfell survivors and government minister Michael Gove criticise the F1 team over a sponsors
hip deal.', 'Maroš Šefčovič says it is "time to get medicines across the finish line" but again no deal is struck.', 'The Ba
rking MP held ministerial posts and chaired the powerful Public Accounts Committee.', 'Sir Desmond Swayne raises concerns ab
out changes to the code of conduct governing MPs behaviour.', 'The addresses of more than 1,000 people, including Sir Elton
John and Gabby Logan, were shared online.', 'The government acted too slowly to stop fraudsters exploiting the Covid loan sc
heme, a report says.', 'Reports suggesting a US decision to maintain tariffs on UK steel is linked to Brexit are rejected.',
"Sir Jeffrey Donaldson says Covid is no joking matter in response to Sammy Wilson's tweet.", 'Lord McKenzie of Luton has bee
n described as "one of the town's most ardent servants".', 'Sir Richard Leese has stepped down from the role of Manchester C
ity Council leader after 25 years.', 'Three Conservatives, including ex-leader Sir Iain Duncan Smith, give up paid roles adv
ising firms.', 'The empty brown fields of winter countryside could be transformed under new land subsidy rules for Englan
```

Figure 16: Displaying all headline results

Then applying some pre-processing techniques like tokenization and stop words to clean the data and it converts into vector.

```
#Preprocessing
ps = PorterStemmer()
clean_text = []
for i in text:
    word_tokens = word_tokenize(i.lower())
    n = ""
    for w in word_tokens:
        if w not in stopwords.words("english"):
            if (w.isalpha()):
                n += ps.stem(w) + " "
    clean_text.append(n)
print(clean_text)
```

```
['labour mp accus govern stoke cultur war harm anyon slightli differ ', 'london elect prompt death former mp jame brokenshir ',
'govern minist accus give mix messag event festiv period ', 'alex bourn say former health secretari involv compani win nh work
', 'grenfel survivor govern minist michael gove criticis team sponsorship deal ', 'maroš šefčovič say time get medicin across f
inish line deal struck ', 'bark mp held ministeri post chair power public account committe ', 'sir desmond swayn rais concern c
hang code conduct govern mp behaviour ', 'address peopl includ sir elton john gabbi logan share onlin ', 'govern act slowli sto
p fraudster exploit covid loan scheme report say ', 'report suggest us decis maintain tariff uk steel link brexit reject ', 'si
r jeffrey donaldson say covid joke matter respons sammi wilson tweet ', 'lord mckenzi luton describ one town ardent servant ',
'sir richard lees step role manchest citi council leader year ', 'three conserv includ sir iain duncan smith give paid role adv
is firm ', 'empti brown field winter countrysid could transform new land subsidi rule england ', 'hereav famili group call apol
```

Figure 17: Applied pre-processing techniques

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
data = vectorizer.fit_transform(clean_text)
print(data.todense())
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Figure 18: TF-IDF Vectorizer

K-means Algorithm

K-means algorithm is the one of the most often used algorithms that support clustering process. The K-means method, which is essentially an iterative algorithm, is one of the most prevalent algorithms that assist the clustering process. The information is assigned to the clusters with the shortest centroid distance. The first step in the k means algorithm's main operation is to define the number of clusters. The number of clusters must next be initialised, which is accomplished by shuffling the dataset and then picking the K data point for the centroids at random. K-means cluster is applied. Here we taken the k value as 3 because three categories of news websites are chosen.

```
from sklearn.cluster import KMeans
K = 3
model = KMeans(n_clusters=K)
model.fit(data)

print(model.labels_)
```

```
[2 2 2 2 2 2 2 2 0 2 0 2 0 0 0 1 0 0 2 2 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 1 0 0
 0 2 0 0 2 2 0 0 0 2 0 0 0 0 2 2 0 0 2 0 2 2 2 2 0 0 0 2 0 0 0 0 0 1 2 0 2
 0 1 2 0 0 0 0 2 2 2 2 0 0 0 2 1 0 2 2 2 2 0 0 0 0 0 0 2 2 2 0 0 2 1 1 0 2
 0 2 0 1 2 1 0 1 2 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 2]
```

Figure 19: K-means cluster

Results:

In this document clustering we are asking the user to enter a string for which user need to know cluster. that input will be pre processed and it will assign to the cluster.

Example here we are searching sentence like" face mask covering is important"

```
#Taking input from user
q = input("Enter a string for which you need to know cluster: ")

#pre-processing
word_tokens = word_tokenize(q)
x = ""
for w in word_tokens:
    if w not in stopwords.words("english"):
        x += ps.stem(w) + " "
print(x)

#predicting cluster
Y = vectorizer.transform([x])
res = model.predict(Y)
print("cluster is: ", res)
```

```
Enter a string for which you need to know cluster: face mask covering are important
```

Figure 20: Searching user given input

```
#Taking input from user
q = input("Enter a string for which you need to know cluster: ")

#pre-processing
word_tokens = word_tokenize(q)
x = ""
for w in word_tokens:
    if w not in stopwords.words("english"):
        x += ps.stem(w) + " "
print(x)

#predicting cluster
Y = vectorizer.transform([x])
res = model.predict(Y)
print("cluster is: ", res)
```

```
Enter a string for which you need to know cluster: face mask covering are important
face mask cover import
cluster is:  [2]
```

Figure 21: Cluster value generated screenshot

Finally we get the Search keyword “face mask cover import” cluster result as 2.

Conclusion:

As given in the task 1 vertical search engine is created similarly to the Google Scholar successfully. This technique will retrieve articles published by SLS members by matching the user's conditions after entering the key term in the relevant search section. Web crawling is a technology that helps in search operations by successfully collecting a copy of the viewed sites for subsequent usage. Indexer is an essential method in Information Retrieval technology. Inverted index has been used for this task. This technique compiles a collection of text elements and references to documents that include those elements. Some pre-processing techniques are applied in order to improve the search results. Next query process has done successfully, and my query is vectorised and matching the whole vector. The document clustering procedure is carried out in the second task using the K-means cluster method, and the cluster value is generated as a result.

Appendix:

Program Link: Google Drive:

https://drive.google.com/file/d/1UwPBAlghXaP-ubKoBK_9y7Md25TYN76_/view?usp=sharing

Git hub Link:

https://github.com/JyothiYendamuri/Information-Retrieval-Coursework/blob/main/Information%20Retrieval%20Source%20Code.ipynb

Video Link:

Coursework Task 1 and Task2 results are shared in video:

https://drive.google.com/file/d/1kk4WZj9ofXpfH0bol55iCzEEkbFBRU9P/view?usp=sharing

References:

Rohit, S. (2021). *Information Retrieval System Explained: Types, Comparison and Components.* Retrieved from: https://www.upgrad.com/blog/information-retrieval-system-explained/

Pedro, H. (2008). *Developing Web Crawlers for Vertical Search Engines.* Retrieved from survey of current research. Western Washington University. https://cedar.wwu.edu/cgi/viewcontent.cgi?article=1001&context=computerscience_stupubs

Ashna, J. (2019). *Information Retrieval using Boolean Query in Python*. Retrieved from voice https://medium.com/voice-tech-podcast/information-retrieval-using-boolean-query-in-python-e0ea9bf57f76

Moz. *How Search Engine Work: Crawling, Indexing and Ranking*. Retrieved from Moz pro. https://moz.com/beginners-guide-to-seo/how-search-engines-operate

Vivek, C. Nisha, T. *Document clustering using K-means*. Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/6112875

William, S. *TF-IDF from scratch in python on a real-world dataset*. Retrieved from towards Data science: https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089