

(7071CEM)

Coursework

Information Retrieval

MODULE LEADER: Dr. Seyed Mousavi

Student Name: Bharti Tayal

SID: 10895366

I can confirm that all work submitted is my own: Yes

Task 1. Search Engine:

A Search engine is used to search information over the webpages. Whenever a user enters any information to search, search engine scans the internet and returns the relevant websites. It breaks down the user query into keywords and then use algorithms to match those keywords with websites titles, content, keywords on internet. There are many search engines like Google, yahoo, Bing, etc.

Search Engine works in three following steps:

1. Crawling: The main part of crawler is to search the information on internet. It thoroughly scans all the URLs it finds.
2. Indexing: The next step after Crawling is to store the crawled information in an organised manner so that it can be displayed to the user according to the relevancy.
3. Ranking: It provides pieces of content that is used to sort the results by most relevant to least relevant.

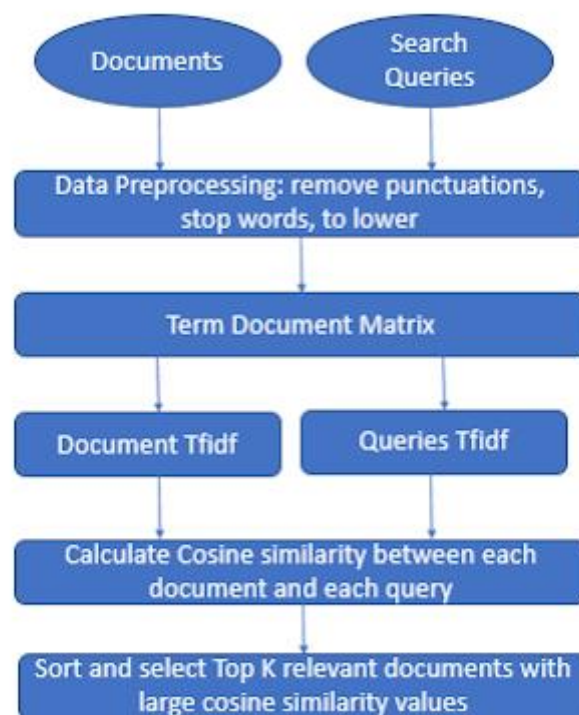


Fig1. Basic structure of Information Retrieval Architecture

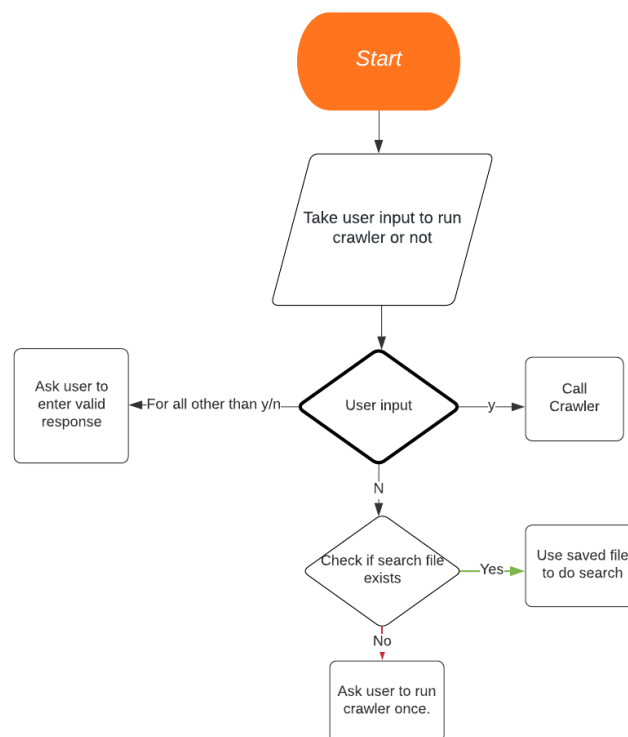
(Source: https://3.bp.blogspot.com/-Ix9pGH_mYNM/Wfgm064NpqI/AAAAAAAAAGCA/X-P4V2917Pgr5hPnRVvRkfRD6GwgiLIrACK4BGAYYCw/s400/information%2Bretrieval_2.PNG)

In this task, the aim is to design a search engine which can retrieve papers published by a member of the School of Life Sciences (SLS) at Coventry University only. The step-wise execution of the task is given below:

Step-1: Crawling:

The link to the page to be crawled is given below:

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



**Fig1. Flowchart showing how crawler will initialise
(Self-designed using Lucidchart)**

The above flow chart explains that initially, program is asking from the user to run the crawler or not. The crawler will **run manually**. If the user wants to run crawler, user can type “y” and crawler would start scouring webpage and results will be saved in csv file. If user enters “n”, the crawler would not run and program will start searching in already saved csv file. In order to create a csv file, the crawler should run once so that results of execution can be saved. For all other responses, program will ask user to give response either in y/n.

```

run_crawler=input("Do you want to run a crawler (y/n): ")
if (run_crawler.lower()=='y'):
    mycrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif(run_crawler.lower()=='n'):
    if(os.path.isfile('research.csv')):
        print("Search Engine initiated")
    else:
        print("No Crawler output exists, you need to run crawler first!")
else:
    print("Invalid input received. Please enter your results in y/n")
    
```

Fig2. Asking user to run Crawler or not

```
run_crawler=input("Do you want to run a crawler (y/n): ")
if (run_crawler.lower()=='y'):
    mycrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif(run_crawler.lower()=='n'):
    if(os.path.isfile('research.csv')):
        print("Search Engine initiated")
    else:
        print("No Crawler output exists, you need to run crawler first!")
else:
    print("Invalid input received. Please enter your results in y/n")
```

Do you want to run a crawler (y/n): n
No Crawler output exists, you need to run crawler first!

Fig.3. Output when user enters “n” and file not found

```
Do you want to run a crawler (y/n): y
fetching https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences/publications/
https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences/publications/?page=1
title of the paper is: A Comparison of Physical Activity Between Home-Based and Centre-Based Pulmonary Rehabilitation: A Ran
domised Controlled Secondary Analysis
link to the paper is: https://pureportal.coventry.ac.uk/en/publications/a-comparison-of-physical-activity-between-home-based
-and-centre-b
Date of publication is: 26 Oct 2021
authors' profile links:
Horton, E.
https://pureportal.coventry.ac.uk/en/persons/elizabeth-horton
Sewell, L.
https://pureportal.coventry.ac.uk/en/persons/louise-sewell
-----
title of the paper is: Actual and perceived motor competence mediate the relationship between physical fitness and technical
skill performance in young soccer players
link to the paper is: https://pureportal.coventry.ac.uk/en/publications/actual-and-perceived-motor-competence-mediate-the-re
lationship-be
Date of publication is: 11 Jul 2021
```

Fig.4. Output when user enters “y” and crawling starts

```
Do you want to run a crawler (y/n): n
Search Engine initiated
```

Fig.5. Output when user enters “n” and file found

```
run_crawler=input("Do you want to run a crawler (y/n): ")
if (run_crawler.lower()=='y'):
    mycrawler('https://pureportal.coventry.ac.uk/en/organisations/school-of-life-sciences',20)
elif(run_crawler.lower()=='n'):
    if(os.path.isfile('research.csv')):
        print("Search Engine initiated")
    else:
        print("No Crawler output exists, you need to run crawler first!")
else:
    print("Invalid input received. Please enter your results in y/n")
```

Do you want to run a crawler (y/n): yes
Invalid input received. Please enter your results in y/n

Fig.6. Output when user enters any other character than y or n

The figures mentioned above show different outputs received according to the user choices. The program will run crawler if user enters “y”.

```
def mycrawler(seed, maxcount):
    seed_url= seed+ "/publications/"
    Q = [seed_url]
    count = 0
    fills=[] #List to append records

    baseurl = "https://pureportal.coventry.ac.uk"

    while(Q!=[] and count < maxcount):
        url = Q[0] #accessing first element of queue
        Q = Q[1:] #reassigning queue from second element #removing first element
        print("fetching " + url)

        code = requests.get(url) #Reading page
        plain = code.text #HTML text of page
        #Passing HTML text to BeautifulSoup object to parse it and allow to search for any part of it
        s = BeautifulSoup(plain, "html.parser")

        next_pg=s.find("a", class_="nextLink")
        if(next_pg!= None):
            url_next_page=next_pg.get('href')

            #Normalisation
            if( url_next_page[0:7] != 'http://' and url_next_page[0:8]!='https://' ) :
                if(baseurl[len(baseurl) -1] == '/'):
                    url_next_page = baseurl + url_next_page
                else:
                    url_next_page = baseurl + '/' + url_next_page

            print("Link to next page is: ", url_next_page)
            #appending link to next page in queue
            Q.append(url_next_page)

        #fetching all information about all divisions with class result-container
        results=s.find_all("div", class_="result-container")

        #retrieving details about paper and storing in dictionary
        for paper in results:
            fill={}
            paper_title = paper.find("h3", class_="title")
            paper_link = paper.find("a", class_="link")
            published=paper.find("span", class_="date")

            if(paper_title.text!= None):
                print("title of the paper is: ",paper_title.text)
                print("link to the paper is: ",paper_link.get('href'))
                print("Date of publication is:", published.text)
                print("authors' profile links:")

                author_name= []
                author_profile=[]
                for authors in paper.find_all("a", class_="link person"):

                    auth = authors.get('href')
                    auth_name=authors.string
                    author_name.append(auth_name)
                    author_profile.append(auth)
                    print(auth_name)
                    print(auth)

                fill['title']=paper_title.text
                fill['Link to paper']= paper_link.get('href')
                fill['Date of Publish']= published.text
                fill['author name']=author_name
                fill['authors profile link']=author_profile
                fills.append(fill)

            print("-----")

        #writing information to csv file
        filename = 'research.csv'
        with open(filename, 'w', newline='', encoding="utf-8") as f:
            w = csv.DictWriter(f,['title','Link to paper','Date of Publish','author name','authors profile link'])
            w.writeheader()
            for fill in fills:
                w.writerow(fill)
            count=count+1
        print("Crawler ran successfully and results stored in Research.csv file ")
```

Fig.7. Crawler function

The code snippet above shows how the crawler will run. Initially, the seed url is passed which is then added '/publications/', where all the information about publications is stored. The crawler will start its

search from the seed url and do as instruct. The maxcount is the variable that will take care that crawler will stop hitting servers unnecessarily and preserve robots.txt rules. The maxcount is the hit rate. It is set such that if we increase it, no more new information is fetched.

Initially, an empty list is created which will hold the contents searched by the crawler (details of the publications) and will then later be used to write records in a csv file.

In the beginning, the HTML content of the page will be read by the crawler of the seed url which will then be passed to BeautifulSoup. BeautifulSoup will then parse the data from the HTML text which will then be used for web scrapping. The **Breadth-First-Strategy** is used to crawl the webpages. It will first crawl the seed url and fetch information from that link. When this link is passed to the crawler for fetching information, queue will pop that link. After reading information of current page, crawler will go to next page link and information will be fetched. The technique is executed using queue, which follows First-in-First-out strategy. The crawler will scan the webpages unless queue is empty, that is, when it will not find link to next page further.

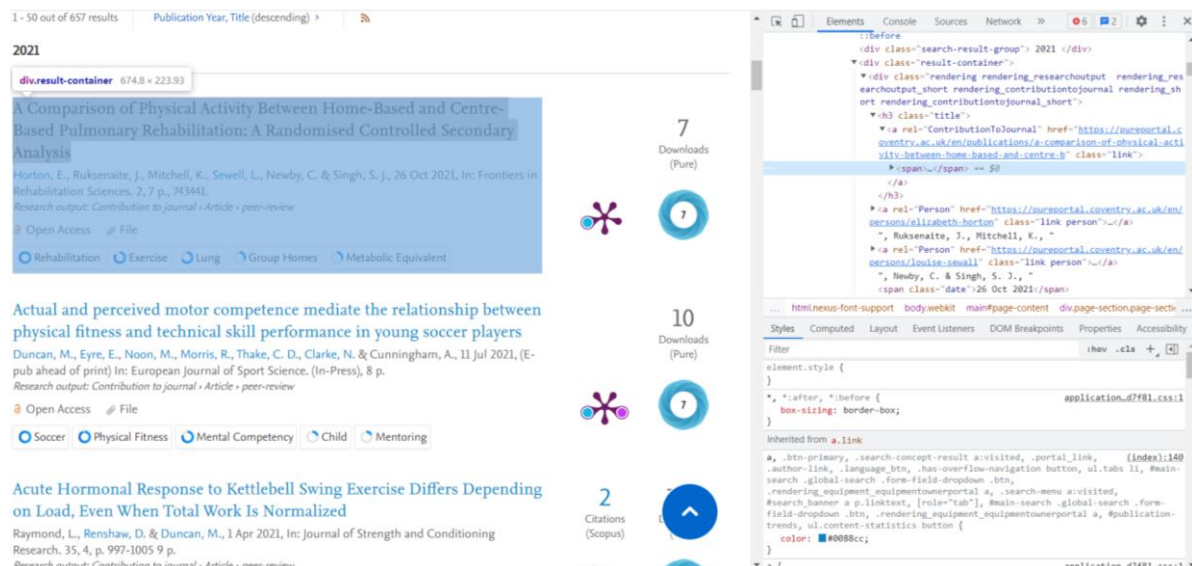


Fig.8. Inspecting Source of given seed url





















All the information regarding the classes and tags have been thoroughly searched from the webpage by inspecting the source. The details about the publications are stored in the division created using <div> tag and class as “Result-container”. From beautiful soup object, all the information contained in <div> tags is extracted whose class is result-container using ‘find_all’. From this extracted information, all the required details about the publications such as **name of the paper, link of the paper, date of publish, author names and their pureportal profile links** is fetched using their tags and classes. All the details about each paper is stored in dictionary in form of key-value pairs, which is then appended to the list and then finally written to the csv file. The csv file looks like:

Fig.9. Research.csv

```
#Reading csv file and storing contents in dataframe
df=pd.read_csv("research.csv", names=['title','Link to paper','Date of Publish','author name','authors profile link'],
               encoding='unicode_escape')
#Dropping first row containing labels only
df=df.iloc[1:,:]
print("No. of research papers retrieved: ", df.title.count())
df.head()
```

	title	Link to paper	Date of Publish	author name	authors profile link
1	A Comparison of Physical Activity Between Home...	https://pureportal.coventry.ac.uk/en/publicati...	26 Oct 2021	['Horton, E.', 'Sewell, L.']	['https://pureportal.coventry.ac.uk/en/persons...
2	Actual and perceived motor competence mediate ...	https://pureportal.coventry.ac.uk/en/publicati...	11 Jul 2021	['Duncan, M.', 'Eyre, E.', 'Noon, M.', 'Morris...']	['https://pureportal.coventry.ac.uk/en/persons...
3	Acute Hormonal Response to Kettlebell Swing Ex...	https://pureportal.coventry.ac.uk/en/publicati...	1 Apr 2021	['Renshaw, D.', 'Duncan, M.']	['https://pureportal.coventry.ac.uk/en/persons...
4	Associations Between Self-Reported Sleep Durat...	https://pureportal.coventry.ac.uk/en/publicati...	Jul 2021	['Roden, L. C.']	['https://pureportal.coventry.ac.uk/en/persons...
5	Blood cancer care in a resource limited settin...	https://pureportal.coventry.ac.uk/en/publicati...	17 Sep 2021	['Lin, T. T.']	['https://pureportal.coventry.ac.uk/en/persons...

After Analysing Research.csv, it was observed that many papers have been retrieved whose authors are not the part of the Coventry University. Hence, these rows were dropped from the dataframe as the task is to search from the publications of the CU authors only.

	title	Link to paper	Date of Publish	author name	authors profile link
22	Improving food safety culture in Nigeria: A re...	https://pureportal.coventry.ac.uk/en/publicati...	13 Aug 2021		
30	Rates of Bile Acid Diarrhoea After Cholecystec...	https://pureportal.coventry.ac.uk/en/publicati...	Aug 2021		
44	The Relationship between Training Load Measure...	https://pureportal.coventry.ac.uk/en/publicati...	25 Jun 2021		
52	A national survey exploring the influence of U...	https://pureportal.coventry.ac.uk/en/publicati...	13 Nov 2020		
66	Does ultrasound equally improve the quality of...	https://pureportal.coventry.ac.uk/en/publicati...	1 Feb 2020		
...
643	The uses of ultrasound in food technology	https://pureportal.coventry.ac.uk/en/publicati...	Nov 1996		
644	Use of crosslinked mucilage prepared from rure...	https://pureportal.coventry.ac.uk/en/publicati...	1996		
648	The Influence of Sonication on the Palladium-C...	https://pureportal.coventry.ac.uk/en/publicati...	May 1994		
656	The O-Alkylation of 5-Hydroxy Chromones. A Com...	https://pureportal.coventry.ac.uk/en/publicati...	1990		
657	NMR studies of some π -4-diene-rhodium(I) and m...	https://pureportal.coventry.ac.uk/en/publicati...	Apr 1989		

143 rows x 5 columns

7

```
#Removing papers whose authors are Not part of CU
df.drop(df.loc[df['author name']=="[]"].index, inplace=True)
print("No. of research papers left after removing papers from non CU authors: ", df.title.count())
df.head()
```

	title	Link to paper	Date of Publish	author name	authors profile link
1	A Comparison of Physical Activity Between Home...	https://pureportal.coventry.ac.uk/en/publicati...	26 Oct 2021	[Horton, E.', 'Sewell, L.]	https://pureportal.coventry.ac.uk/en/persons...
2	Actual and perceived motor competence mediate ...	https://pureportal.coventry.ac.uk/en/publicati...	11 Jul 2021	['Duncan, M.', 'Eyre, E.', 'Noon, M.', 'Morris...	https://pureportal.coventry.ac.uk/en/persons...
3	Acute Hormonal Response to Kettlebell Swing Ex...	https://pureportal.coventry.ac.uk/en/publicati...	1 Apr 2021	['Renshaw, D.', 'Duncan, M.]	https://pureportal.coventry.ac.uk/en/persons...
4	Associations Between Self-Reported Sleep Durat...	https://pureportal.coventry.ac.uk/en/publicati...	Jul 2021	['Roden, L. C.]	https://pureportal.coventry.ac.uk/en/persons...
5	Blood cancer care in a resource limited settin...	https://pureportal.coventry.ac.uk/en/publicati...	17 Sep 2021	['Lin, T. T.]	https://pureportal.coventry.ac.uk/en/persons...

The next step is to make the list of the title of the research papers so that preprocessing can be done on it before making the inverted index.

Fig.13. List of titles of papers published

8


```
#Data pre processing using Tokenizer, Stemmer and Stop Word removal process
#Importing necessary Libraries
from nltk.stem.porter import PorterStemmer
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

stemmer = PorterStemmer()
def preprocessing(doc):
    # changing sentence to lower case
    doc = doc.lower()
    print(doc)
    tokenizer = nltk.RegexpTokenizer(r'\w+') #removing punctuation
    # tokenize into words
    words = tokenizer.tokenize(doc)
    print(words)
    # removing stop words
    words = [word for word in words if word not in stopwords.words("english")]
    print(words)
    # stemming
    words = [stemmer.stem(word) for word in words]
    print(words)
    # join words to make sentence
    doc = " ".join(words)
    return doc
```

Fig.14. Pre-processing function

After pre-processing, stemmed_words are joined to make sentences which will then be used for tfidfVectoriser.

```
#Pre-processing the title list
documents = [preprocessing(document) for document in title_list]
print(documents)

a comparison of physical activity between home-based and centre-based pulmonary rehabilitation: a randomised controlled second
ary analysis
['a', 'comparison', 'of', 'physical', 'activity', 'between', 'home', 'based', 'and', 'centre', 'based', 'pulmonary', 'rehabil
itation', 'a', 'randomised', 'controlled', 'secondary', 'analysis']
['comparison', 'physical', 'activity', 'home', 'based', 'centre', 'based', 'pulmonary', 'rehabilitation', 'randomised', 'cont
rolled', 'secondary', 'analysis']
['comparison', 'physic', 'activ', 'home', 'base', 'centr', 'base', 'pulmonari', 'rehabilit', 'randomis', 'control', 'secondar
i', 'analysi']
actual and perceived motor competence mediate the relationship between physical fitness and technical skill performance in yo
ung soccer players
['actual', 'and', 'perceived', 'motor', 'competence', 'mediate', 'the', 'relationship', 'between', 'physical', 'fitness', 'an
d', 'technical', 'skill', 'performance', 'in', 'young', 'soccer', 'players']
['actual', 'perceived', 'motor', 'competence', 'mediate', 'relationship', 'physical', 'fitness', 'technical', 'skill', 'perfo
rmance', 'young', 'soccer', 'players']
['actual', 'perceiv', 'motor', 'compet', 'mediat', 'relationship', 'physic', 'fit', 'technic', 'skill', 'perform', 'young',
'soccer', 'player']
acute hormonal response to kettlebell swing exercise differs depending on load, even when total work is normalized
['acute', 'hormonal', 'response', 'to', 'kettlebell', 'swing', 'exercise', 'differs', 'depending', 'on', 'load', 'even', 'whe
n', 'total', 'work', 'is', 'normalized']
```

Fig.14. Pre-processed list of title of research paper

TfidfVectoriser will be used to compute the word counts, IDF values and Tf-idf scores. It is not of much importance here as only titles are extracted which do not have much occurrence of similar words. It is useful for ranking system and how to find relevancy between documents and query.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
tfidf_model = vectorizer.fit_transform(documents)
print(tfidf_model)
```

```
(0, 86)      0.2510479749364271
(0, 1437)    0.32884499926065125
(0, 346)     0.2627004242036015
(0, 1328)    0.2774239631842375
(0, 1361)    0.2833366573632885
(0, 1315)    0.2833366573632885
(0, 246)     0.32884499926065125
(0, 156)     0.43154256014688813
(0, 738)     0.3060908283119698
(0, 25)      0.17503264598968868
(0, 1213)    0.18599311743152444
(0, 311)     0.2627004242036015
(1, 1226)    0.22265667918749263
(1, 1498)    0.2289250826036151
(1, 1815)    0.27830350303518764
(1, 1194)    0.1925472260155497
(1, 1486)    0.23117648229576337
(1, 1626)    0.3130977189441367
(1, 612)     0.26871404163725554
(1, 1364)    0.283774625108655
(1, 980)     0.29658380888790614
(1, 312)     0.27830350303518764
(1, 1040)    0.283774625108655
(1, 1191)    0.28982266996672673
```

Fig.15. Tf-Idf model

```
print(tfidf_model.shape)
```

```
(515, 1822)
```

```
# print the full sparse matrix
print(tfidf_model.toarray())
```

```
[[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.]
```

```
title_vec=pd.DataFrame(tfidf_model.toarray(), columns = vectorizer.get_feature_names())
```

```
title_vec
```

	abil	absolut	abund	academ	academi	acceler	acceleromet	accelerometeri	accelerometri	access	...	write	year	yeast	young	youth	zimbabw	zi
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	0.000000	0.0	0.0	0.0
1	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	0.278304	0.0	0.0	0.0
2	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	0.000000	0.0	0.0	0.0
3	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	0.000000	0.0	0.0	0.0
4	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	0.000000	0.0	0.0	0.0
...

Fig.16. shape of Tf-Idf model , sparse matrix, dataframe

Here, the shape of model shows that there are 515 documents stored. Number of features are 1822 which are represented as columns in the dataframe.

Step-2: Indexing:

The documents to be indexed should be preprocessed first. The preprocessing done initially was for Tf-idf where stemmed words were joined. But for indexing, we need separate words. Inverted Indexing is used to reduce the response time the search engine takes for searching. With the help of inverted index, search engine does not have to match the information of query with the all the documents individually. It can find the documents containing token in inverted index. It contains all the preprocessed terms which are also known as tokens along with the

document id in which the token is present. So, the inverted index is a sequence of modified token and docid pairs.

Here, inverted index will be updated only when it will find either new term or new docid containing term. It is **not getting build from scratch** every time.

```
#Each word feature will be mapped against the doc ids where it is found.
#Search Pattern will use this indexed corpus for faster search.

from collections import defaultdict
from nltk.tokenize import sent_tokenize, word_tokenize

#Index is a map of word in documents where it is found in
inverted_index = defaultdict(set)
from nltk.corpus import stopwords
from nltk.stem import snowball
stwords = set(stopwords.words('english'))

# Maintain the reference to the document by its index in the corpus list
for docid, c in enumerate(title_list):
    #print(docid, c)
    for sent in sent_tokenize(c):
        for word in word_tokenize(sent):
            if (word.isalpha()):
                word_lower = word.lower()
            if word_lower not in stwords:
                stemmer = snowball.SnowballStemmer('english')
                word_stem = stemmer.stem(word_lower)

                # indexes will be incremented when it will have any new word rather than building from scratch
                if (word_stem in inverted_index) and (docid not in inverted_index[word_stem]):
                    inverted_index[word_stem].add(docid)
                # New term in inverted_index
            else:
                inverted_index[word_stem].add(docid)

print(len(inverted_index.keys()))
print(inverted_index)

1694
defaultdict(<class 'set'>, {'comparison': {0, 96, 162, 259, 323, 8, 73, 427, 142, 272, 370, 216}, 'physic': {0, 1, 257, 132,
7, 11, 13, 22, 24, 25, 26, 280, 28, 285, 419, 38, 39, 166, 424, 426, 172, 50, 51, 310, 57, 186, 319, 451, 71, 461, 334, 336,
213, 86, 216, 346, 348, 93, 222, 231, 235, 237, 112, 368, 114, 119, 121, 122, 253, 255}, 'activ': {0, 7, 13, 277, 22, 280, 2,
5, 26, 281, 285, 419, 38, 166, 424, 426, 172, 174, 50, 51, 435, 310, 57, 186, 444, 318, 447, 451, 196, 326, 199, 461, 334, 33,
6, 464, 213, 86, 475, 348, 93, 97, 484, 231, 487, 233, 234, 235, 377, 237, 489, 112, 368, 114, 499, 500, 119, 121, 122, 508,
253, 382, 255}, 'pulmonari': {0, 162, 334, 368, 370, 377, 447}, 'rehabilit': {0, 162, 100, 138, 334, 370, 377, 447}, 'randomi
s': {0, 162, 195, 164, 294, 118, 377, 317, 447}, 'control': {0, 385, 195, 164, 294, 231, 144, 115, 118, 216, 217, 317, 447},
'secondari': {0, 276, 125}, 'analysi': {0, 7, 490, 142, 17, 506, 439, 58}, 'actual': {1, 12, 217}, 'perceiv': {1, 5, 38, 363,
12, 143, 18}, 'motor': {1, 81, 114, 85, 21, 213, 217, 190, 95}, 'compet': {1, 394, 12, 18, 114, 21, 213, 439, 217, 95}, 'medi
at': {1, 450, 235, 114, 499, 443, 412}, 'relationship': {1, 34, 129, 257, 293, 174, 213, 310, 438}, 'fit': {1, 34, 257, 166,
172, 174, 112, 18, 340, 312, 317, 285}, 'technic': {176, 1, 18, 210}, 'skill': {1, 8, 12, 16, 17, 18, 144, 26, 414, 291, 306,
180, 53, 190, 191, 72, 81, 83, 85, 222, 240, 119, 249, 255}, 'perform': {1, 129, 387, 132, 260, 262, 263, 265, 139, 18, 147,
148, 151, 284, 286, 289, 36, 37, 433, 179, 314, 63, 64, 325, 330, 331, 333, 462, 335, 84, 90, 346, 350, 351, 226, 229, 357, 2
31, 109, 238, 112, 113, 252, 118, 251, 124, 383}, 'young': {1, 257, 36, 329, 142, 302, 304, 52, 217, 315}, 'soccer': {1, 257,
131, 132, 5, 260, 138, 11, 401, 18, 151, 32, 417, 34, 41, 49, 306, 56, 64, 72, 73, 216, 92, 353}, 'player': {1, 257, 131, 13
2, 264, 11, 401, 24, 31, 32, 415, 34, 35, 417, 39, 44, 49, 433, 62, 64, 72, 73, 205, 206, 216, 89, 92, 353, 102, 108}, 'acu
t': {65, 2, 292, 484, 230, 393, 489, 331, 492, 205, 464, 147, 148, 341, 312, 219, 284, 287}, 'hormon': {2, 318}, 'respons':
{129, 2, 389, 393, 12, 269, 146, 413, 288, 164, 173, 46, 434, 312, 195, 454, 207, 209, 339, 467, 341, 468, 227, 230, 492, 36
```

Fig.17. Building inverted Index

In this program, the inverted indexes are stored in the form of dictionary where keys represent token and values represent docids. The next step after forming the sequence is to sort them by terms.

```
#Sorting inverted index
sort_inverted_index=sorted(inverted_index.items())
sorted_inverted_index = defaultdict(set)
sorted_inverted_index=dict(sorted_inverted_index)
print(sorted_inverted_index)

{'abil': {371, 95}, 'absolut': {146}, 'abund': {390}, 'academ': {128, 292, 171, 139, 112, 127}, 'academi': {417, 34, 194, 5,
39, 264, 205, 433, 374, 407, 380, 89, 92, 415}, 'acceler': {64, 72, 35, 383}, 'acceleromet': {121, 321, 424, 237, 142, 119, 5
7}, 'accelerometer': {53}, 'accelerometri': {361, 93, 119}, 'access': {328, 329, 327}, 'acclim': {393, 244, 341, 309, 316, 2
84}, 'accord': {216, 486}, 'account': {30}, 'accumul': {467}, 'acetoxyacid': {487, 494, 471}, 'achiev': {292, 478, 342},
'achill': {428, 388, 399}, 'acid': {483, 198, 498, 182, 184, 59, 188}, 'acl': {400}, 'acn': {233, 196, 281}, 'across': {232,
301, 346, 62, 415}, 'act': {157}, 'actigraph': {57, 142}, 'action': {260}, 'activ': {0, 7, 13, 277, 22, 280, 25, 26, 281, 28
5, 419, 38, 166, 424, 426, 172, 174, 50, 51, 435, 310, 57, 186, 444, 318, 447, 451, 196, 326, 199, 461, 334, 336, 464, 213, 8
6, 475, 348, 93, 97, 484, 231, 487, 233, 234, 235, 377, 237, 489, 112, 368, 114, 499, 500, 119, 121, 122, 508, 253, 382, 25
5}, 'actwatch': {51}, 'actual': {1, 12, 217}, 'aculeatus': {305}, 'acut': {65, 2, 292, 484, 230, 393, 489, 331, 492, 205, 46
4, 147, 148, 341, 312, 219, 284, 287}, 'acyl': {434}, 'adapt': {107, 117, 383}, 'addit': {184, 278}, 'address': {375}, 'adeno
sin': {360, 449, 234}, 'adenovirus': {423}, 'adequ': {240}, 'adher': {13}, 'adhes': {385}, 'adipos': {36}, 'adjunct': {342},
'adjust': {137}, 'administr': {169}, 'adolesc': {416, 161, 132, 261, 235, 427, 429, 430, 112, 240, 274, 275, 401}, 'adopt':
{13}, 'adren': {490}, 'adult': {142, 273, 23, 27, 285, 286, 36, 38, 172, 46, 47, 175, 307, 52, 187, 315, 324, 70, 203, 204, 3
31, 219, 347, 93, 227, 361, 118, 246, 378, 510}, 'advanc': {399}, 'advers': {201}, 'aerob': {34, 340}, 'aeruginosa': {165},
'affect': {168, 330, 46, 19, 507, 247, 347}, 'africa': {496, 236}, 'african': {177}, 'africana': {233, 165}, 'age': {257, 13
4, 265, 285, 291, 39, 180, 53, 310, 57, 187, 190, 63, 321, 195, 324, 73, 85, 213, 88, 216, 247, 221, 97, 486, 108, 237, 113,
119, 249}, 'agent': {281}, 'aggreg': {384}, 'agreement': {279}, 'aid': {41}, 'air': {71}, 'al': {321, 297}, 'albin': {327, 32
8, 329, 236, 496, 177, 497, 504}, 'algarobia': {421}, 'alien': {405}, 'alkalin': {59}, 'alloy': {98}, 'alon': {351, 86, 95},
'alter': {97, 98, 134, 231, 508}, 'altitud': {488, 485}, 'amateur': {337}, 'ambient': {120, 439, 379, 391}, 'ambulatori': {47
```

Fig.18. Sorting inverted Index according to terms

Posting list is the list which contains the document ids in which the token is present. It can be a one document or than one documents. Generally, posting list is stored in the form of list or dictionary. In this program, the posting list is stored in the form of set which is similar to dictionary.

```
#Displaying Postings List from inverted index
def post_list(term):
    return sorted_inverted_index[term]
print(post_list("ambient"))

{120, 439, 379, 391}
```

Fig.19. Sorting inverted Index according to terms

Step-3: Query Processor:

The last step of the task is to process the user query. In this step, user has to enter the topic related to which he is interested in looking for papers. This query can be of single words or multiple words. It can be a combination of both upper case and lower case letters; may contain some stop words, punctuation, numbers, etc. In order to retrieve good and more results, the user query should be preprocessed similar to the process followed for the titles of the research paper.

```
#Here the search patterns are searched in OR condition.
def preprocess_and_search(query):
    matched_documents = set() #to store the information about papers with which the query matches
    #Pre-Processing user query
    for word in word_tokenize(query):

        if word.isalpha():
            word_lower = word.lower()

            if (word_lower not in stwords):
                word_stem = stemmer.stem(word_lower)
                query_stem.append(word_stem)

            # fetching docid of documents with which query matches using inverted index
            matches = sorted_inverted_index.get(word_stem)

            if matches:
                # The operator |= is a short hand for set union
                #Gathering all the matching docids in a set
                matched_documents |= matches #union signifies OR operation
                print("Documents found for ", word_stem, "are : ",matched_documents)

    print("User query after pre-processing ",query_stem)
    return matched_documents
```

Fig.20. Query Processor

After preprocessing the user input query, the stemmed words will be appended into an initialised list which will store all the preprocessed words of user input query. These stemmed_words are then used to find matches from the sorted inverted index. If it finds the match, i.e., the stemmed word gets matched with token in sorted inverted index, the respective docid will be stored in variable matches. If matches contain any docid, that docid will be added to matched_documents. The same process will be repeated for all the stemmed words.

Docids of all the stemmed words are being merged using union operations of sets which reflects OR Boolean logic. OR Boolean logic is used to retrieve as much as paper possible. The concept of OR Boolean logic is that if query has more than one word, the docids will be searched for every word and will be merged. Union operation of sets will ensure that there is no duplicity of docids in final matched set.

```
# the below search function will return the doc id which is aligned to the index of the corpus.
query_stem = []
searchstring=input("enter your search string ")
doc_id = preprocess_and_search(searchstring)
print("Final matching list : ", doc_id)
print("No of matching documents found: ",len(doc_id))

enter your search string Systematic Review and Meta-Analysis
Documents found for systemat are : {161, 3, 40, 43, 13, 48, 112, 274, 275, 84, 21, 276, 27, 223}
Documents found for review are : {3, 13, 80, 274, 275, 84, 21, 276, 218, 27, 223, 161, 40, 43, 110, 48, 112}
User query after pre-processing ['systemat', 'review']
Final matching list : {3, 13, 80, 274, 275, 84, 21, 276, 218, 27, 223, 161, 40, 43, 110, 48, 112}
No of matching documents found: 17
```

Fig.21. User enters a query having punctuation, stop words

From fig 21, it is evident that matches are found for all the stemmed words obtained after pre-processing. As it is clear from the fig, 14 matching docids were found for ‘systemat’ and 17 matching docids were found for ‘review’. The union of both the sets is taken and docids lying in both the sets are considered only once to avoid redundancy.

Step-4: Ranked Retrieval:

After query processing, it is very useful to calculate relevance score for each document so that retrieved papers are presented in descending order of their relevance.

```
#extracting tfidf scores for only matched docids and features
tf_idf_rank_list=list()
if(doc_id!= set()):
    for i in (doc_id):
        tf_idf_rank=0
        for j in range(len(query_stem)):
            try:
                q=query_stem[j]
                tf_idf_rank += title_vec.iloc[i][q]
            except KeyError:
                tf_idf_rank+=0
        tf_idf_rank_list.insert(i,tf_idf_rank)
    print(tf_idf_rank_list)
    print(len(tf_idf_rank_list))
```

```
[0.49974942335128447, 0.41889528459758363, 0.21548848334579268, 0.4760883016641112, 0.5284579706761955, 0.6133490674520764, 0.5040296212873823, 0.4464300467976165, 0.17992574434945852, 0.42080566101627737, 0.4617016620887658, 0.4760883016641112, 0.37174889885399653, 0.47378157802090665, 0.21859160149237505, 0.4828247160377729, 0.399013371750363]
17
```

Fig.22. Extracting tfidf scores for matched docids

The next step after obtaining docids, tf-idf scores are fetched from pre-designed models. The scores are fetched only for those where stem words of user query lie in the document. All the fetched Tf-idfs score are inserted in list which is further sorted in decreasing order so that most relevant document; document with higher tf-idf score is printed on the top.

```
sorted_tf_idf_rank_list=(sorted( [(x,i) for (i,x) in enumerate(tf_idf_rank_list)], reverse=True ))
print(sorted_tf_idf_rank_list)
```

```
[(0.6133490674520764, 5), (0.5284579706761955, 4), (0.5040296212873823, 6), (0.49974942335128447, 0), (0.4828247160377729, 15), (0.4760883016641112, 11), (0.4760883016641112, 3), (0.47378157802090665, 13), (0.4617016620887658, 10), (0.4464300467976165, 7), (0.42080566101627737, 9), (0.41889528459758363, 1), (0.399013371750363, 16), (0.37174889885399653, 12), (0.21859160149237505, 14), (0.21548848334579268, 2), (0.17992574434945852, 8)]
```

Fig.23. Sorting fetched tf-idf ranks

Sorted rank list will give the relevance of each document present in docid. For eg, in Fig23, the maximum match index found is (0.613349, 5) where 0.613349 signifies the Tf-idf score and 5 is the index position of document in docid. It means document available at 5th index position in docid is most relevant. After getting ranks for each document, the information using docid and dataframe containing information about paper is displayed.

```
#Papers fetched with relevancy
if (doc_id != set()):
    for i in range(len(sorted_tf_idf_rank_list)):
        x=sorted_tf_idf_rank_list[i][1]
        y=list(doc_id)[x]

        print("Title of Paper:",df.iloc[y,0])
        print("Link of Paper:",df.iloc[y,1])
        print("Date of Paper Published:",df.iloc[y,2])
        print("Author names:",df.iloc[y,3])
        print("Link to Author Profiles:",df.iloc[y,4])
        print("-----")
else:
    print("No match found")
```

Title of Paper: The Effects of Caffeine Ingestion on Measures of Rowing Performance: A Systematic Review and Meta-Analysis
Link of Paper: <https://pureportal.coventry.ac.uk/en/publications/the-effects-of-caffeine-ingestion-on-measures-of-rowing-performance>
Date of Paper Published: 8 Feb 2020
Author names: ['Duncan, M.', 'Tallis, J.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/michael-duncan', 'https://pureportal.coventry.ac.uk/en/persons/jason-tallis']

Title of Paper: What are the views of overweight and obese adolescents (12â17yrs) attending lifestyle treatment interventions: a qualitative systematic review
Link of Paper: <https://pureportal.coventry.ac.uk/en/publications/what-are-the-views-of-overweight-and-obese-adolescents-1217yrs-at>
Date of Paper Published: 2 Sep 2017
Author names: ['Jones, H.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/helen-jones']

Fig.24. Paper displayed as per relevance

Results are also published without rank retrieval to check the working of ranking.

```
#papers fetched without relevancy
if (doc_id != set()):
    for i in doc_id:
        print("Title of Paper:",df.iloc[i,0])
        print("Link of Paper:",df.iloc[i,1])
        print("Date of Paper Published:",df.iloc[i,2])
        print("Author names:",df.iloc[i,3])
        print("Link to Author Profiles:",df.iloc[i,4])
        print("-----")
else:
    print("No match found")
```

Title of Paper: Associations Between Self-Reported Sleep Duration and Mortality in Employed Individuals: Systematic Review and Meta-Analysis
Link of Paper: <https://pureportal.coventry.ac.uk/en/publications/associations-between-self-reported-sleep-duration-all-cause-and-c>
Date of Paper Published: Jul 2021
Author names: ['Roden, L. C.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/laura-roden']

Title of Paper: Effectiveness of Structured Physical Activity Interventions Through the Evaluation of Physical Activity Levels, Adoption, Retention, Maintenance, and Adherence Rates: A Systematic Review and Meta-Analysis
Link of Paper: <https://pureportal.coventry.ac.uk/en/publications/effectiveness-of-structured-physical-activity-interventions-through>
Date of Paper Published: Jan 2021
Author names: ['Willinger, N.', 'Jimenez, A.', 'Horton, E.']
Link to Author Profiles: ['https://pureportal.coventry.ac.uk/en/persons/nadja-willinger', 'https://pureportal.coventry.ac.uk/en/persons/alfonso-jimenez', 'https://pureportal.coventry.ac.uk/en/persons/elizabeth-horton']

Title of Paper: Rheological, tribological and sensory attributes of texture-modified foods for dysphagia patients and the elderly: A review

Fig.25. Paper displayed without relevance

After comparing results from Fig24 and fig25, it is observed that rank retrieval is a useful parameter in order to display results.

When user enters topic which is not found in inverted index, program will display no result found.


```
enter your search string Credit Card fraud
User query after pre-processing ['credit', 'card', 'fraud']
Final matching list : set()
No of matching documents found: 0
```

```
sorted_tf_idf_rank_list=(sorted( [(x,i) for (i,x) in enumerate(tf_idf_rank_list)], reverse=True ))
print(sorted_tf_idf_rank_list)
```

```
[]
```

```
#Papers fetched with relevancy
if (doc_id!= set()):
    for i in range(len(sorted_tf_idf_rank_list)):
        x=sorted_tf_idf_rank_list[i][1]
        y=list(doc_id)[x]

        print("Title of Paper:",df.iloc[y,0])
        print("Link of Paper:",df.iloc[y,1])
        print("Date of Paper Published:",df.iloc[y,2])
        print("Author names:",df.iloc[y,3])
        print("Link to Author Profiles:",df.iloc[y,4])
        print("-----")
else:
    print("No match found")
```

```
No match found
```

Fig.26. No paper information printed

Further Enhancements:

The search can be made more refined by using AND Boolean logic where only intersecting docids will be considered. The model can further be enhanced to search paper information from Author's name. Moreover, phase queries and longer phase queries can also be introduced. It can be given nice Graphical User Interface having more knowledge.

Task 2. Document Clustering:

Clustering is used to group a set of documents into two or more clusters. These clusters are different from each other. The documents in one cluster should be similar whilst documents which are not present within one cluster should be dissimilar.

It is an unsupervised learning technique in which classes or labels of the documents to be clustered is not known. In clustering, Distance measure is an important means using which outcomes of clustering can be evaluated. Suppose these are different documents which need to be clustered:

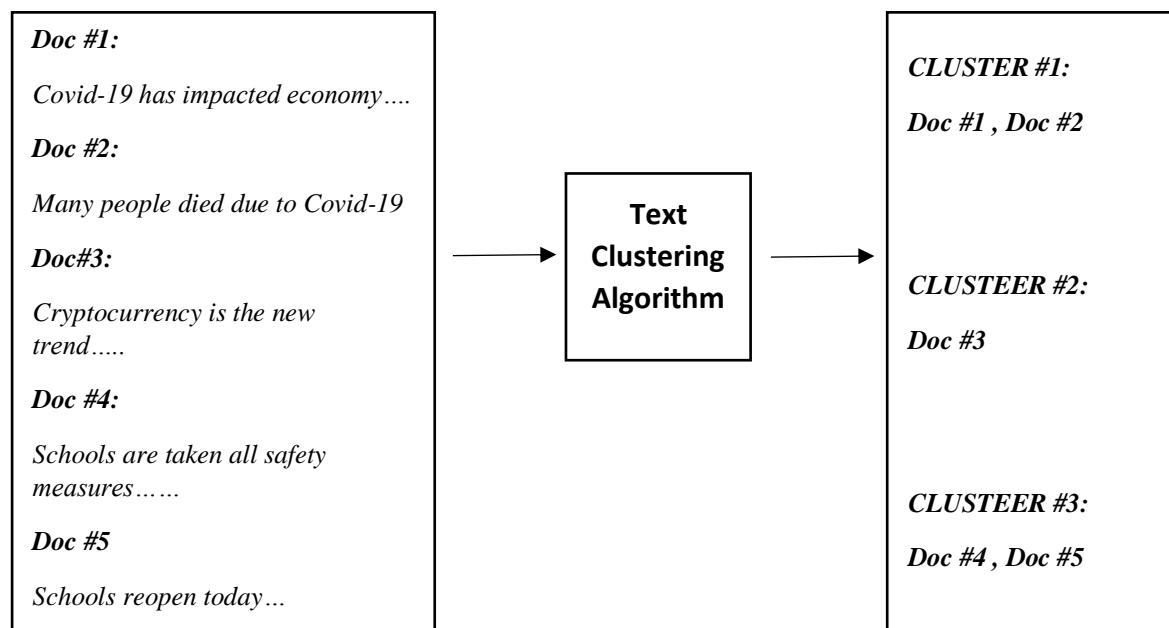


Fig. Clustering Algorithm Input and Output

Types of Clustering:

There are two types of clustering available based on how many clusters are assigned to each document: Hard Clustering, in which, only one cluster is assigned to each document, whilst, if a document is present in more than one cluster, it is Soft Clustering.



Fig. Hard and Soft Clustering

(Source: https://miro.medium.com/max/1313/1*bBYxRxPveZ8khii6Ge2T5Q.png)

Clustering can also be categorised as Flat Clustering and Hierarchical Clustering. There is no partitional clustering in Flat Clustering whereas when cluster splits itself into several clusters, it is known as Hierarchical Clustering.

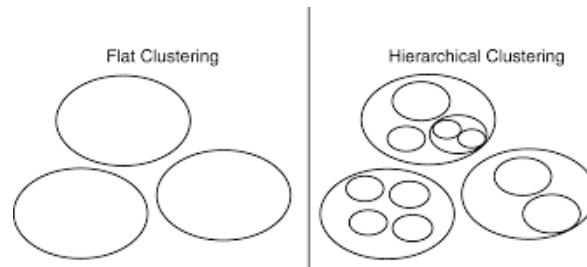


Fig. Flat and Hierarchical Clustering

(Source: https://media.springernature.com/original/springer-static/image/chp%3A10.1007%2F978-3-319-91815-0_9/MediaObjects/394318_1_En_9_Fig6_HTML.png)

Algorithm Used:

There are many algorithms available to do clustering depend on different types clustering. In this task, Clustering has been executed using KMeans algorithm which is a Flat and Hard Clustering technique.

Implementation:

The execution steps of this task are as follows:

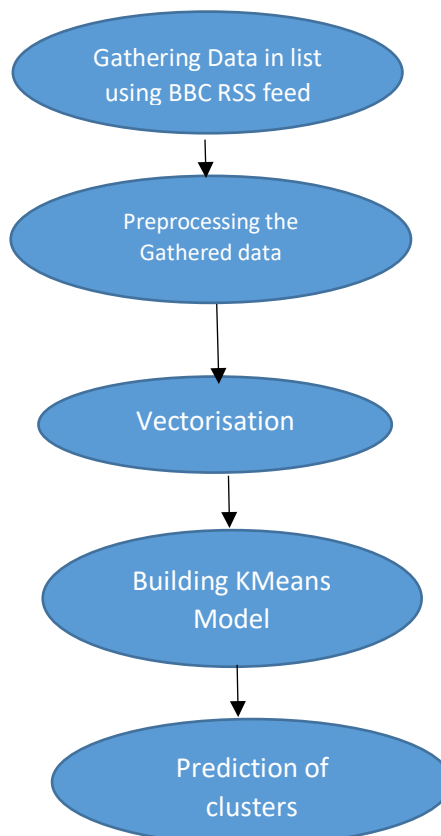


Fig. Execution Flow of Clustering

Step-1: Installing Packages and Gathering Data

Installing Feedparser package to read RSS feed

```
pip install feedparser
```

```
Requirement already satisfied: feedparser in c:\users\hp\anaconda3\lib\site-packages (6.0.8)
Requirement already satisfied: sgmllib3k in c:\users\hp\anaconda3\lib\site-packages (from feedparser) (1.0.0)
Note: you may need to restart the kernel to use updated packages.
```

Fig. Installing packages for fetching RSS feed

The data has been fetched from the BBC news website from its RSS feed using feedparser. News of five different categories : Business, Science, Arts, Technology, and Politics have been fetched and assigned labels 0,1,2,3,4 respectively and appended in a list Y. These labels will be helpful in predicting accuracy. These are just assumed labels as discussed above also, no human can predict how clusters are actually assigned by algorithm.

From the fetched data, its RSS summary is extracted so that we have limited and precise data to be clustered. These news headlines are then appended into a final list, lst. The number of documents fetched vary every time the program is run as news website is updated timely.

```
1. #Fetching news using RSS feed
2. #getting summary instead of full RSS feed
3. import feedparser
4. lst=[]
5. Y=[]
6. NewsFeed_business = feedparser.parse("http://feeds.bbc.co.uk/news/business/rss.xml")
7. NewsFeed_sci =
  feedparser.parse("http://feeds.bbc.co.uk/news/science_and_environment/rss.xml")
8. NewsFeed_arts =
  feedparser.parse("http://feeds.bbc.co.uk/news/entertainment_and_arts/rss.xml")
9. NewsFeed_tech = feedparser.parse("http://feeds.bbc.co.uk/news/technology/rss.xml")
10. NewsFeed_pol= feedparser.parse("http://feeds.bbc.co.uk/news/politics/rss.xml")
11.
12. for entry in NewsFeed_business.entries:
13.     x= entry.summary
14.     y=0
15.     lst.append(x)
16.     Y.append(y)
17.
18. for entry in NewsFeed_sci.entries:
19.     y= entry.summary
20.     lst.append(y)
21.     y=1
22.     Y.append(y)
23.
24. for entry in NewsFeed_arts.entries:
25.     z= entry.summary
26.     lst.append(z)
27.     y=2
28.     Y.append(y)
29.
30. for entry in NewsFeed_tech.entries:
31.     p= entry.summary
32.     lst.append(p)
33.     y=3
34.     Y.append(y)
35.
36. for entry in NewsFeed_pol.entries:
37.     p= entry.summary
38.     lst.append(p)
39.     y=4
```

```
40.     Y.append(y)
41.
42. print("no of news headlines fetched: ",len(lst))
43. print("number of labels is: ", len(Y))
44. print(lst)
```

Code Snippet. Preparing documents to be clustered

```
no of news headlines fetched: 174
['The British Retail Consortium says people should show consideration to store workers and other shoppers.', 'The airline has temporarily stopped flights after more of its staff were forced to isolate.', 'Critics claim publicly available data shows worrying failures on the 737 Max since its return to service.', 'Chip shortages lead to production dropping to the lowest level recorded for October since 1956.', 'Analysts say this year's sale will be the biggest to date, but retailers won't offer as many discounts.', 'The boss of the world's largest fertiliser producer says gas prices are responsible for higher food prices.', 'Wife of a former Nissan boss on trial in Japan says he is a forgotten victim of the Carlos Ghosn affair.', 'Strikes or protests are planned in 20 countries, on one of the busiest days of the year for retail.', 'Black Friday is back and Americans, flush with Covid cash, are going out to spend.', 'It is unclear whether the placement scheme for 16-24 year olds is working, says National Audit Office.', 'It comes as China's technology industry regulator reviews compliance with new privacy rules.', 'Steve Finch of toy retailer Bopster, explains the pressures his firm is under for Black Friday this year.', 'Six southern African countries are put on the red list, amid "deep concern" over a new variant of the virus.', 'The UK energy market has seen 24 companies collapse since September after wholesale gas prices rose.', 'TNT owner FedEx is sorry as hundreds say they have faced long waits to receive travel documents.', 'It was the most used during a year of lockdowns that saw passenger numbers drop 78% due to Covid.', 'The energy firm, with 1.7 million customers, has been put into a special form of administration.', 'Parties are also likely to be in venues outside city centres, says the owner of Harvester and All Bar One.', 'Jamie Dimon has apologised after saying that his Wall Street bank would outlast China's ruling party.', 'It is the latest such move as soaring prices threaten to destabilise the post-pandemic recovery.', 'The move comes as tensions grow between the US and China over the status of Taiwan and trade issues.', 'Browns of Chester served the city for 241 years before it closed when the
```

Fig. News document

Step-2: Preprocessing the fetched Data:

The data fetched using RSS feed will be now preprocessed. Basic pre-processing techniques are applied; Changing case to lower case, tokenization, removal of stop words, unicodes, and punctuation, Stemming.

#Pre-processing the fetched feed

```
filtered_docs = []
for doc in lst:
    tokens = word_tokenize(doc)
    tmp = ""
    for w in tokens:
        if w not in stopwords:
            if(w.isalpha()):
                stemmer = snowball.SnowballStemmer('english')
                tmp += stemmer.stem(w) + " "
    filtered_docs.append(tmp)

print(filtered_docs)
```

```
['the british retail consortium say peopl show consider store worker shopper ', 'the airlin temporarili stop flight staff for c isol ', 'critic claim public avail data show worri failur max sinc return servic ', 'chip shortag lead product drop lowest level record octob sinc ', 'analyst say year sale biggest date retail wo offer mani discount ', 'the boss world largest ferti lis produc say gas price respons higher food price ', 'wife former nissan boss trial japan say forgotten victim carlo ghosn a ffair ', 'strike protest plan countri one busiest day year retail ', 'black friday back american flush covid cash go spend ', 'it unclear whether placement scheme year old work say nation audit offic ', 'it come china technolog industri regul review c omplianc new privaci rule ', 'steve finch toy retail bopster explain pressur firm black friday year ', 'six southern african countri put red list amid deep concern new variant virus ', 'the uk energi market seen compani collaps sinc septemb wholesal gas price rose ', 'tnt owner fedex sorri hundr say face long wait receiv travel document ', 'it use year lockdown saw passeng number drop due covid ', 'the energi firm million custom put special form administr ', 'parti also like venu outsid citi cent r say owner harvest all bar one ', 'jami dimon apologis say wall street bank would outlast china rule parti ', 'it latest mov e soar price threaten destabilis recoveri ', 'the move come tension grow us china status taiwan trade issu ', 'brown chester serv citi year close chain collaps may ', 'portug bike factori busi sinc april cycl surg popular ', 'peter done betfr ceo pen insula share busi advic ceo secret ', 'south africa main electr compani eskom plan switch use coal renew energi ', 'how forme r world war two pilot came second cycl competit older peopl ', 'a host firm develop mini nuclear reactor question remain safe ti cost ', 'covid push mani us ditch screen time hobbi knit among popular ', 'covid chang shop habit shift make one biggest p urchas ', 'covid led increas parent gurus use social media support new mum dad ', 'alway supplier take need say offcut boss n ish parekh ', 'small farm africa struggl compet commerci agricultur new platform help ', 'how footbal sticker maker surviv th iev tycoon competit wrestler ', 'a grow number tech firm develop robot say could replac bar staff ', 'xbox boss phil spencer
```

Fig. Preprocessed news

Step-3: Vectorization:

Machine Learning algorithms use vectorised features (i.e. words should be converted into vectors) while doing NLP. In this program, TF-IDF is used to construct the vectors. In this process, not only words are being converted into vectors but also an entire document is being translated into a vector. Vectorised feature names are also stored in a variable words so that it can be used further.

```
#TfidfVectorisation
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(filtered_docs)
print(X.todense())
words = vectorizer.get_feature_names()
print(words)

[[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.]]
['absolut', 'academ', 'access', 'accus', 'acoust', 'act', 'action', 'activist', 'adapt', 'administr', 'adopt', 'advanc', 'adv
ertis', 'advic', 'aerosol', 'affair', 'africa', 'african', 'after', 'age', 'agenda', 'agreement', 'agricultur', 'ahead', 'ai
m', 'airlin', 'album', 'algorithm', 'all', 'also', 'alway', 'amaz', 'ambit', 'ambiti', 'american', 'amid', 'among', 'amount',
'analyst', 'ancient', 'andrew', 'anim', 'anniversari', 'apologis', 'applaud', 'approach', 'april', 'arcan', 'archer', 'area',
'argentina', 'around', 'art', 'artifici', 'as', 'assess', 'asteroid', 'astronom', 'attach', 'attract', 'audit', 'avail', 'ave
ng', 'averag', 'aw', 'back', 'bad', 'ban', 'band', 'bank', 'bar', 'bargain', 'base', 'bateson', 'bbc', 'becom', 'began', 'beg
in', 'behind', 'best', 'bestsel', 'betfr', 'better', 'bexley', 'big', 'biggest', 'bike', 'bill', 'black', 'blind', 'block',
'blue', 'boat', 'book', 'boom', 'boost', 'bopster', 'bori', 'boss', 'boy', 'brazil', 'bristol', 'britain', 'british', 'broa
d', 'broken', 'brokenshir', 'brought', 'brown', 'budziak', 'build', 'busi', 'busiest', 'but', 'buy', 'by', 'cage', 'call', 'c
ame', 'camp', 'campaign', 'canada', 'cancel', 'cancer', 'cape', 'carbon', 'care', 'carlo', 'carolin', 'case', 'cash', 'cast
l', 'celebr', 'centr', 'centuri', 'ceo', 'chain', 'chair', 'chang', 'channel', 'chapter', 'character', 'chart', 'check', 'chemi
c', 'chester', 'chief', 'children', 'china', 'chines', 'chip', 'choic', 'christma', 'citi', 'claim', 'claimant', 'class', 'cl
```

Fig. Vectorisation

Step-4: Building model:

The K-means clustering algorithm is used as it is the most appropriate model used for clustering.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
wcss = []
for i in range(1,7):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=100,n_init=15,random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,7),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.savefig('elbow.png')
plt.show()
```

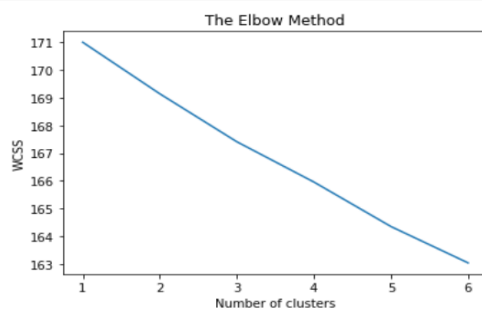


Fig. Using Elbow method

Here, elbow method was also used additionally to predict suitable number of clusters so that if more clusters are added, there is no further improvement. Generally, the plot looks like the elbow. The x-axis value of the point of the corner of the elbow gives the suitable number of clusters. For this model, curve is not good as expected but very slight curves can be observed at K=3,4 and 5. So, all these K-values will be used to build the clusters and then results will be compared. Some words are also displayed from each cluster to have an idea what sort of words it has considered similar.

```
#Clustering with K=5
import numpy as np
from sklearn.cluster import KMeans
# trying to make 5 clusters as there are five categories: Business, Science, art, technology and politics
K = 5
model_K5 = KMeans(n_clusters=K)
model_K5.fit(X)

print("cluster no. of input documents, in the order they received:")
print(model_K5.labels_)

# We look at 5 the clusters generated by k-means.
common_words = model_K5.cluster_centers_.argsort()[:, :-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))

cluster no. of input documents, in the order they received:
[2 0 0 0 2 4 4 2 0 2 3 2 3 1 0 2 3 0 0 0 0 2 4 0 0 4 0 0 1 3 0 3 4 1 0 0 0
 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 0 0 2 3 1 0 0 0 0 0 1 0 4 0 0 0 0 3 0 0 0
 0 3 0 0 0 1 0 0 0 0 2 0 0 0 2 3 0 3 0 0 0 0 0 0 3 2 1 2 3 2 3 0 2 0 0 1 4
 0 4 0 0 0 0 0 0 4 0 3 3 0 0 1 0 0 1 1 1 0 2 3 0 0 0 0 0 0 4 2 0 0 0 0 0 0
 0 0 3 0 0 1 0 4 3 2 0 3 4 0 0 0 4 0 0 0 1 0 0 0 0 2]
0 : the, say, parti, energi, leader, use, week, help, one, labour, cost, bbc, johnson, green, plan, move, remain, call, back, h
ope, govern, take, bori, mps, best
1 : chang, climat, could, live, the, uk, glasgow, summit, need, implic, way, come, made, year, practic, risk, seen, take, cris
i, know, viral, went, new, emiss, determin
2 : year, retail, day, busiest, may, giant, say, one, first, plan, strike, uk, the, protest, covid, countri, rebellion, due, ne
xt, explain, close, music, look, show, it
3 : new, put, health, variant, say, rule, covid, complianc, privati, struggl, commerci, compet, farm, platform, regul, industr
i, agricultur, dad, media, support, increas, led, gurus, mum, parent
4 : world, peopl, how, second, cycl, competit, former, pilot, older, war, came, two, govern, price, boss, say, race, creat, aro
und, generat, lobbi, group, tighter, approach, hull
```

Fig. Clustering with K=5

```
#Clustering with K=4
import numpy as np
from sklearn.cluster import KMeans
# trying to make 4 clusters
K = 4
model_K4 = KMeans(n_clusters=K)
model_K4.fit(X)

print("cluster no. of input documents, in the order they received:")
print(model_K4.labels_)

# We look at 4 clusters generated by k-means.
common_words = model_K4.cluster_centers_.argsort()[:, :-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))

cluster no. of input documents, in the order they received:
[1 0 0 0 1 1 1 3 0 1 1 1 3 2 1 1 2 1 1 1 0 1 0 0 2 0 0 0 3 2 3 2 0 1 0 2 0
 0 0 2 3 3 3 3 3 0 3 0 3 3 0 3 2 0 0 1 1 0 2 0 0 2 0 3 0 3 0 2 2 2 1 0 0 0
 0 1 2 1 0 3 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 3 1 0 0 0 1 2 3 1 2
 1 0 0 2 0 2 0 2 0 0 2 2 0 1 0 0 1 0 1 2 1 1 0 1 1 1 0 0 1 0 0 3 1 1 0
 0 0 3 1 0 3 0 1 0 0 0 0 0 1 0 0 0 0 0 0 3 0 0 1 0 0]
0 : the, week, govern, peopl, minist, found, one, how, remain, call, sinc, back, rule, bori, bbc, news, johnson, constitu, tran
sport, covid, technolog, go, make, trade, second
1 : say, year, the, it, parti, rule, new, work, retail, health, come, secretari, leader, offic, due, price, first, china, restr
ict, travel, uk, fan, peopl, could, cost
2 : use, energi, compani, africa, plan, new, south, help, piec, comput, commerci, platform, compet, struggl, farm, agricultur,
electr, coal, main, renew, eskom, switch, global, gurus, increas
3 : chang, climat, countri, need, could, the, live, one, year, practic, implic, way, strike, glasgow, busiest, protest, tackl,
made, day, summit, thing, say, retail, flood, plan
```

Fig. Clustering with K=4

```
#Clustering with k=3
import numpy as np
from sklearn.cluster import KMeans
# trying to make 3 clusters
K = 3
model_K3 = KMeans(n_clusters=K)
model_K3.fit(X)

print("cluster no. of input documents, in the order they received:")
print(model_K3.labels_)

# We look at 3 clusters generated by k-means.
common_words = model_K3.cluster_centers_.argsort()[:, :-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))

cluster no. of input documents, in the order they received:
[1 1 2 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0
0 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 2 1 1 0 0 1 1 0 1 0 0 0 2 0 1 0 0 1 0
0 0 0 1 1 1 0 1 1 1 1 0 0 0 2 1 1 0 1 1 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 1 1 1 1 1 2 1 0 0 0 1 0 0 0 0 2 1 0 2 0 2
0 2 0 0 1 1 0 2 2 2 2 1 0 1 1 0 0 1 0 2 1 0 1 2 0 0]
0 : say, new, use, plan, year, one, rule, technolog, covid, retail, it, govern, cost, energi, peopl, make, small, countri, fir
m, review, week, how, africa, mps, world
1 : the, chang, climat, could, say, live, made, glasgow, take, uk, come, health, way, summit, grow, part, trade, year, need, mu
sic, risk, us, issu, leader, found
2 : first, johnson, minist, bori, secretari, term, public, set, year, say, come, scotland, omicron, migrant, transport, return,
ban, consult, look, crisi, variant, claim, care, england, labour
```

Fig. Clustering with K=3

All the three models predict one cluster for each document which proves K-means is a hard and flat clustering technique.

Step-5: Predicting Clusters:

After building the model, some random data containing 5 documents of each category as same as news headlines are passed to all the three models to see how they group it. The data is preprocessed in a similar manner as news document. Afterwards, its vectorised features will be created which is passed to predict clusters to all the three designed models with different k values.

```
1. #testing document
2. #trying with different categories of news to check what clusters we get
3. #Business, Science, art, technology, Politics
4. #these testing document contains queries containing stop words, Punctuations, mixture of
   Upper case and lower case letters
5. test_doc = ['The UK energy market has seen 24 companies collapse since September after
   wholesale gas prices rose',
6.             'Sir Andrew Pollard warns against possible moves to reverse planned
   investment in science',
7.             'Adele album became 2021 fastest seller',
8.             'Huge fines on makers of insecure smart devices.',
9.             'PM under pressure has to act on migrant sea crossings.'
10.            ]
11.
12. #Preprocessing on test document
13. filtered_test_docs = []
14. for doc in test_doc:
15.     tokens = word_tokenize(doc)
16.     tmp = ""
17.     for w in tokens:
18.         if w not in stwords:
19.             if(w.isalpha()):
20.                 tmp += stemmer.stem(w) + " "
21.     filtered_test_docs.append(tmp)
22.
23. print("User query after preprocessing is: ",filtered_test_docs)
```

Code Snippet. Predicting clusters for test document

Fig. Predicted clusters for test document

Fig. Displaying assigned and predicted labels

23

The performance of the model can be further enhanced by collecting more amount of the data so that algorithm can work efficiently and better results will be achieved.

```
from sklearn.metrics import classification_report
y_true = Y
y_pred = model_K5.labels_
target_names = ['Business', 'Science', 'art', 'technology', 'Politics']
print(classification_report(y_true, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
Business	0.14	0.23	0.18	35
Science	0.30	0.33	0.31	42
art	0.12	0.15	0.13	20
technology	0.23	0.19	0.20	27
Politics	0.26	0.12	0.16	50
accuracy			0.21	174
macro avg	0.21	0.20	0.20	174
weighted avg	0.22	0.21	0.21	174

Fig. Classification report for K=5 clustering model

Appendix:

The code and video can be found here:

Video- <https://drive.google.com/file/d/1DmI2JvnY6GFUjfFZ4K4KBWgrMLW4EAW7/view>

Video has been uploaded on Aula also.

Code-

<https://colab.research.google.com/drive/1GWNPXLoFCj2T3WWMA7kej3pRQWV7afcV?usp=sharing>