**(7071CEM)**

**Information Retrieval**

MODULE LEADER: Seyed Mousavi

Student Name: Siddhesh Ranjeet Desai

SID: 13008482

COURSEWORK

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

Table of Contents

[Task 1: Search Engine 3](#_Toc131119847)

[Crawler 3](#_Toc131119848)

[Number of staff whose publications are crawled (approximately) and the minimum number of publications per staff 3](#_Toc131119849)

[Information collected about each publication 5](#_Toc131119850)

[Which pre-processing tasks are performed before passing data to Indexer 6](#_Toc131119851)

[When the crawler operates 7](#_Toc131119852)

[A brief explanation of how it works 8](#_Toc131119853)

[Indexer 9](#_Toc131119854)

[Whether you implemented the index or used Elastic Search 9](#_Toc131119855)

[Data structure used 9](#_Toc131119856)

[Whether it is incremental, i.e. it grows and gets updated over time, or it is constructed from scratch every time your crawler is run 9](#_Toc131119857)

[Show some part of its content 9](#_Toc131119858)

[A brief explanation of how it works 10](#_Toc131119859)

[Query Processor 11](#_Toc131119860)

[Which pre-processing tasks are applied to a given query 11](#_Toc131119861)

[Do you only support Boolean queries (using AND, OR, NOT, etc.) or accept keywords like Google does (without any need for AND, OR, NOT etc.) 11](#_Toc131119862)

[If Elastic Search is used, how you convert a user query to an appropriate query for Elastic Search 11](#_Toc131119863)

[If Elastic Search is NOT used, whether or not you perform ranked retrieval; if yes, specify whether or not you used vector space and the method used to calculate the ranks 11](#_Toc131119864)

[Demonstration of the running system 12](#_Toc131119865)

[A brief explanation of how it works 13](#_Toc131119866)

[Part 2 – Document Clustering 14](#_Toc131119867)

[How and how many input documents are collected 14](#_Toc131119868)

[Which document clustering method (e.g. K-means with appropriate K value) has been used and how its performance is measured 14](#_Toc131119869)

[Which type of clustering is used (hierarchical/flat and hard/soft) 14](#_Toc131119870)

[Demonstration 15](#_Toc131119871)

[A brief explanation of how it works 17](#_Toc131119872)

[References 18](#_Toc131119873)

[Appendix 19](#_Toc131119874)

[Task 1: Search Engine 19](#_Toc131119875)

[Crawler.py 19](#_Toc131119876)

[InvertedIndex.py 23](#_Toc131119877)

[QuerySearch.py 25](#_Toc131119878)

[Task 2: Document Clustering 28](#_Toc131119879)

[DocumentClustering.py 28](#_Toc131119880)

All the tasks are solved using the Python programming language.

# Task 1: Search Engine

## Crawler

### Number of staff whose publications are crawled (approximately) and the minimum number of publications per staff

The total number of staff members who are part of the Research Centre for Computational Science and Mathematical Modelling is 47. Below are the details of each staff member and the total number of publications published by them on the Research Centre for Computational Science and Mathematical Modelling publication page and overall as well.

|  |  |  |
| --- | --- | --- |
| Staff Member | Crawled Research papers count | Total published Research papers |
| Michael Ajao-Olarinoye | 1 | 1 |
| Opeoluwa Akinseloyin | 0 | 0 |
| Emran Ali | 0 | 1 |
| Rjaa Jawad Ashraf | 2 | 2 |
| Abiola Babatunde | 1 | 1 |
| Daniel Babalola Bammeke | 0 | 0 |
| Rashid Barket | 2 | 2 |
| Fatma Benkhelifa | 2 | 36 |
| James Brusey | 11 | 112 |
| Heng Jie Choong | 0 | 0 |
| Tereso del Río Almajano | 4 | 4 |
| Ankur Deo | 3 | 3 |
| James Donnelly | 1 | 1 |
| Matthew England | 17 | 72 |
| Majdi Fanous | 3 | 3 |
| Sivasharmini Ganeshamoorthy | 1 | 1 |
| Susanta Kumar Ghosh | 0 | 0 |
| Stephan Goerttler | 0 | 0 |
| Shenal Rajintha Alexander Samarathunge Gunawardena | 1 | 1 |
| Alison Halford | 18 | 20 |
| Fei He | 15 | 40 |
| Yulian Ivan Mykhailo Honchar | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| Staff Member | Crawled Research papers count | Total published Research papers |
| Pengpeng Hu | 1 | 28 |
| Charlie Ingram | 0 | 1 |
| Brandi Jo Jess | 1 | 1 |
| Xiaorui Jiang | 8 | 19 |
| Sokipriala Solomon Jonah | 0 | 2 |
| Amirkazem Kayhani | 0 | 0 |
| Dominik Klepl | 6 | 6 |
| Ibnu Febry Kurniawan | 4 | 4 |
| Giacomo Demetrio Masone | 0 | 0 |
| Ruchita Mehta | 0 | 0 |
| Mohamed Ahsan Mohamed Mafaz | 0 | 0 |
| Shruti Nair | 0 | 0 |
| Vasile Palade | 47 | 159 |
| Rishav Raj | 0 | 0 |
| Luigi Russi | 0 | 3 |
| AmirHosein Sadeghimanesh Sadeghi Manesh | 5 | 13 |
| Sara Sardari | 0 | 4 |
| Huma Shah | 3 | 25 |
| Colin Stephen | 0 | 4 |
| Saurav Sthapit | 0 | 8 |
| Maria Tariq | 2 | 3 |
| Matthew Stephen Tart | 4 | 5 |
| Abhiram Thiruthummal | 1 | 1 |
| Julia Wieczorek | 0 | 0 |
| Jaimz Winter | 1 | 1 |

### Information collected about each publication

For each article published below information was retrieved

* Title of the research paper
* Link to the research paper
* Published date
* Authors of the research paper
* Profile links to the author profiles

Graphical user interface, text

Description automatically generated

Screenshot 1: Output of a single article

Providing only one output example to avoid lengthy document.

To print details of all the articles uncomment the below section from the code in the Crawler.py file which is provided in Appendix.

Text

Description automatically generated

Code Block 1: To print details of the articles that are crawled

### Which pre-processing tasks are performed before passing data to Indexer

There are a bunch of pre-processed tasks that are performed on the research paper titles before sending it to the Indexer as below

* Removing the non-ASCII characters
* Removing any mentions in the title text
* Converting all the letters into lower cases
* Removing any punctuation marks
* Removing stopwords
* Creating word tokens
* Stemming those tokens

All the pre-processing data code is available in the InvertedIndex.py file under the “## Pre-processing data” section.

Text

Description automatically generated

Code Block 2: Pre-processing data

Graphical user interface, text, application

Description automatically generated

Code Block 3: Tokenization and Stemming

Output after pre-processing:

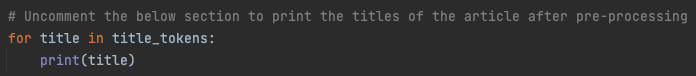
Below are a few examples of how pre-processed data looks

Text

Description automatically generated

Screenshot 2: Pre-processed titles

To print all the titles after pre-processing uncomment the below section in the InvertedIndex.py file.



Code Block 4: To print details of titles of the article after pre-processing

### When the crawler operates

The crawler is programmed to operate in two ways as below

1. Manually  
     
   The crawler can be triggered manually whenever needed to crawl the website and fetch the latest details published on the page.
2. Scheduled run  
     
   The crawler can also be scheduled to be running at a given point in time. As of now since we were supposed to run it once a week I have scheduled it to run every Sunday.

Here is how the options are displayed after running the program

Text

Description automatically generated

Screenshot 3: Crawler operating options

### A brief explanation of how it works

* The crawler can be triggered manually or scheduled to run on a Sunday. The user is given the option to choose.
* If the user schedules the crawler, it calculates the time left until Sunday and triggers the scheduler accordingly.
* The crawler performs the same operation whether triggered manually or scheduled, but the scheduler updates the Inverted Index with the latest data crawled.
* The website is checked for its robots.txt before crawling to ensure safety. The check\_allowed\_robots() method returns true if it's safe to crawl.
* The crawler function is called with the URLs for the publication page and the profile page of CSM group members, and both pages are crawled for necessary information.
* Any available next-page links are saved and crawled to fetch all the details.
* The retrieved data is added to a list using a dictionary containing key-value pairs.
* Finally, the list is written in a CSV file.

## Indexer

### Whether you implemented the index or used Elastic Search

I have chosen to use Index.

### Data structure used

As a part of this implementation, I have chosen an Inverted Index.

### Whether it is incremental, i.e. it grows and gets updated over time, or it is constructed from scratch every time your crawler is run

The inverted index is programmed to run every time a scheduled crawler runs. The inverted index is formed by reading the data present in the CSV file every time it runs. So the contents of the inverted index are updated every time it runs.

### Show some part of its content

Graphical user interface, text

Description automatically generated

Screenshot 4: Contents of Inverted Index

### A brief explanation of how it works

The inverted index code could be found in the InvertedIndex.py file in the Appendix section.

* The inverted\_index function is triggered whenever the scheduler crawler function is executed. A manual trigger is also present in the code if needed.
* The first step of the process is to read the data present in the CSV file. Once the data is fetched a bunch of pre-processing operations are performed on the data.
* Pre-processing activities include removing non-ASCII characters, punctuations, stopwords and mentions. Tokenizing and stemming the input data before creating an actual index from it.
* Once the pre-processing is complete an inverted index is created using the tokenized words.

## Query Processor

### Which pre-processing tasks are applied to a given query

The user input first goes through a spell checker function to check for any misspelt words provided as input. These need to be fixed as they might hamper the performance of the query. Other than that most of the pre-processing tasks are similar to that were applied to the data while creating an Inverted Index like

* Removing stopwords
* Tokenizing words
* Stemming words

### Do you only support Boolean queries (using AND, OR, NOT, etc.) or accept keywords like Google does (without any need for AND, OR, NOT etc.)

As a part of this implementation, I have designed a query processor which only accepts keywords like Google.

### If Elastic Search is used, how you convert a user query to an appropriate query for Elastic Search

Elastic Search is not used as part of this implementation.

### If Elastic Search is NOT used, whether or not you perform ranked retrieval; if yes, specify whether or not you used vector space and the method used to calculate the ranks

The retrieval method used is Boolean retrieval, where documents either match or do not match the query. There is no use of ranking or vector space models for retrieval. The code only matches the query terms to the documents in the inverted index and returns the documents that match the query.

### Demonstration of the running system

Text

Description automatically generated

Screenshot 5: Search result for query “Machine Learning”

In the above screenshot, we can see that the search query is “Machine Learning”. Both the tokens in the search query are correctly spelt and the query processor is able to return 14 documents related to it.

Text

Description automatically generated

Screenshot 6: Search result for the query "nural ntwrk"

In the above screenshot, we can see that the search query is “nural ntwrk”. Both the tokens here are misspelt but the query processor takes care of it and is able to modify the search tokens to “neural network” and display relevant articles for it. This shows the robustness of the system to handle any kind of input.

### A brief explanation of how it works

* The code allows users to search for articles published under the “Research Centre for Computational Science and Mathematical Modelling” by entering a search query.
* With other essential libraries, I have also imported the InvertedIndex module to make use of the function inverted\_index() which will help us in creating an inverted index of the data stored in a CSV file.
* A search string is taken as input from the user. Then it is first checked for any misspelt words.
* Once the spell check is completed further the query is pre-processed by tokenizing and stemming.
* After that, the token words from the query are searched in the document index.
* Once the index is found the relevant documents are fetched and returned.

# Part 2 – Document Clustering

## How and how many input documents are collected

For this application, the documents were collected mainly from 2 main news sources CNN and BBC. There are 5 RSS feeds which were used as input for this application. The feeds belong to categories like Sports, Technology, Entertainment, Politics, and Business.

Text

Description automatically generated

Screenshot 7: New articles details

## Which document clustering method (e.g. K-means with appropriate K value) has been used and how its performance is measured

K-means is used as a clustering method and the number of clusters is 5 as there are 5 types of news categories. The performance of the model is calculated using methods like Inertia and Silhouette.

Text

Description automatically generated

Screenshot 8: Performance metrics for K-means

## Which type of clustering is used (hierarchical/flat and hard/soft)

The type of clustering used in this application is flat and hard clustering.

## Demonstration

Text

Description automatically generated

Screenshot 9: News article 1

In the above result, a news article related to the “Politics” category was provided as input and the program was able to predict the appropriate cluster. Also if you notice there were no spelling mistakes in the input.

Text

Description automatically generated

Screenshot 10: News article 2

Same as the previous example but the new category was “Technology”.

In the below example, news article 3 had misspelt words and regardless of the wrong spelling, the program was able to detect the right cluster for the new news article.

Text

Description automatically generated

Screenshot 11: News article 3

## A brief explanation of how it works

This code performs the following tasks:

* Imports necessary libraries.
* Defines categories for each RSS feed.
* Defines empty lists for each category.
* Defines a function "get\_text" to fetch the text from RSS feeds and stores the text in the corresponding category's list.
* Defines a function "tokenize\_and\_stem" to tokenize and stem the text using the Porter Stemmer algorithm.
* Defines a list of RSS feeds and then calls the "get\_text" function to fetch the text from the RSS feeds and store it in the corresponding category's list.
* Combines text from all feeds into one list and prints the total number of articles fetched from each category and total articles fetched.
* Defines the number of clusters (k) equal to the number of categories.
* Creates a bag of words using CountVectorizer, which converts a collection of text documents to a matrix of token counts.
* Clusters documents using the KMeans algorithm, which is an unsupervised machine learning technique used to cluster data points into a specified number of groups.
* Labels the clusters and prints the number of documents and the documents in each cluster.
* Defines a loop to allow the user to input a news headline and find which cluster it belongs to.
* Spell-checks the search string using the SpellChecker library and assigns the corrected search string to the query variable.
* Transforms the query into a vector using the CountVectorizer, predicts which cluster it belongs to using the KMeans algorithm, and prints the cluster to which the new document belongs.

Overall, this code performs text clustering of RSS feeds using the KMeans algorithm and allows users to find which cluster a new document belongs to based on a news headline input.

# References

1. CNN.com - RSS channel - Sport - Feedburner. (n.d.). Retrieved March 31, 2023, from http://rss.cnn.com/rss/edition\_sport.rss *Function reference* (no date) *• ggplot2*. Available at: https://ggplot2.tidyverse.org/reference/ (Accessed: February 17, 2023).
2. McCallum, S. (2023, March 29). UK rules out new AI regulator. BBC News. Retrieved March 31, 2023, from https://www.bbc.co.uk/news/technology-65102210
3. Youngs, I. (2023, March 29). Paul O'Grady: How lily savage defied police who raided a pub with rubber gloves. BBC News. Retrieved March 31, 2023, from https://www.bbc.co.uk/news/entertainment-arts-65115059
4. BBC News - UK politics. (n.d.). Retrieved March 31, 2023, from https://feeds.bbci.co.uk/news/politics/rss.xml
5. *World Business News - CNNMoney.com - Feedburner. (n.d.). Retrieved March 31, 2023, from http://rss.cnn.com/rss/money\_news\_international.rss*

# Appendix

## Task 1: Search Engine

### Crawler.py

## Importing all the necessary libraries and components

**import** **csv**

**import** **ssl**

**import** **requests**

**import** **sched**, **time**, **datetime**

# Adding this to skip the SSL certificate check

**try**:

\_create\_unverified\_https\_context = ssl.\_create\_unverified\_context

**except** **AttributeError**:

**pass**

**else**:

ssl.\_create\_default\_https\_context = \_create\_unverified\_https\_context

**from** **bs4** **import** BeautifulSoup

**from** **urllib.parse** **import** urljoin

**from** **requests.exceptions** **import** HTTPError

**from** **urllib.robotparser** **import** RobotFileParser

**from** **InvertedIndex** **import** inverted\_index

## Checks whether a given URL is allowed to be crawled by a web crawler, based on the rules specified in the robots.txt file for the domain.

**def** **check\_allowed\_robots**(url) -> bool:

robot\_parser = RobotFileParser()

robot\_parser.set\_url(urljoin(url, "/robots.txt"))

**try**:

robot\_parser.read()

**print**("Provided URL can be crawled as it adheres to the rules mentioned in robots.txt.")

**return** robot\_parser.can\_fetch("\*", url)

**except** HTTPError:

**return** False

## Crawls the given link and stores records in a csv file

**def** **crawler**(pub\_url, prof\_url):

# Initializing variables

publication\_results = []

profile\_results = []

**print**("Crawling Started.")

**print**("Publications link which will be crawled -> ", pub\_url)

**print**("Profiles link which will be crawled -> ", prof\_url)

**if** check\_allowed\_robots(website\_url):

profiles = requests.get(prof\_url)

profiles\_soup = BeautifulSoup(profiles.content, 'html.parser')

profiles\_lists = profiles\_soup.find\_all('div', class\_="result-container")

# Extracting all the profile links of the members of the profiles\_url passed

**print**("Fetching all the profile details.")

**for** lists **in** profiles\_lists:

**for** profile **in** lists.find\_all("a", class\_="link person"):

profile\_results.append(profile.get('href'))

**print**("All the profile details are now retrieved.")

**while** True:

**if** check\_allowed\_robots(website\_url):

publications = requests.get(pub\_url)

publications\_soup = BeautifulSoup(publications.content, 'html.parser')

publications\_lists = publications\_soup.find\_all('div', class\_="result-container")

# Extracting all the information from articles fetched from the publications page

**print**("Fetching all the article details.")

**for** paper **in** publications\_lists:

author\_names = []

author\_profile\_links = []

dictionary = {}

title = paper.find("h3", class\_="title")

paper\_link = paper.find("a", class\_="link")

published\_date = paper.find("span", class\_="date")

#################################################################################################################################################################################

# I attempted to extract the publisher information, but since the property class/tag is not consistent, I am disregarding this information.

#################################################################################################################################################################################

# publisher = paper.find("a", class\_="link", attrs={"rel": ["Publisher", "Journal"]}).find("span").text

# print(publisher)

#################################################################################################################################################################################

# I made an attempt to retrieve the names of authors who did not have a profile, but I was unable to do so because their names were only declared within <span> tags without any specific class. Since pulling all <span> tags returns unnecessary values, I have decided to skip the process of fetching the names of authors without a profile.

#################################################################################################################################################################################

# authors\_span = paper.find("div", class\_="rendering rendering\_researchoutput rendering\_researchoutput\_portal-short rendering\_contributiontobookanthology rendering\_portal-short rendering\_contributiontobookanthology\_portal-short").find\_all("span")

# for author in authors\_span:

# if author.has\_attr('class'):

# continue

# author\_names.append(author.string.strip())

# print(author.string.strip())

**for** author **in** paper.find\_all("a", class\_="link person"):

author\_names.append(author.string)

author\_profile\_links.append(author.get('href'))

#################################################################################################################################################################################

# Uncomment the below section to print details of individual articles.

#################################################################################################################################################################################

# print("--------------------------------------------------------------------------------")

# print("Title of the research paper: ", title.text)

# print("Link to the research paper: ", paper\_link.get('href'))

# print("Published date: ", published\_date.text)

# print("Authors of the research paper: ", ', '.join(author\_names))

# print("Profile links to the author profiles: ", ', '.join(author\_profile\_links))

# Refining the article selection by filtering for pieces authored by members of the Research Centre for Computational Science and Mathematical Modelling (CSM) at Coventry University.

**for** link **in** author\_profile\_links:

**if** link **in** profile\_results:

# Storing all the retrieved information regarding each publication in a dictionary

dictionary['Title of the Research Paper'] = title.text

dictionary['Link to the Research Paper'] = paper\_link.get('href')

dictionary['Published Date'] = published\_date.text

dictionary['Authors'] = author\_names

dictionary['Pureportal Profile Link'] = author\_profile\_links

# Adding all rows to a list

publication\_results.append(dictionary)

**break**

next\_link = publications\_soup.find("a", class\_="nextLink")

**if** **not** next\_link:

**break**

**print**("There are additional articles to browse through, crawling next page contents.")

pub\_url = "https://pureportal.coventry.ac.uk" + next\_link["href"]

**print**("Crawling Completed.")

**print**("Creating a CSV file to store all the records.")

**with** open('records.csv', 'w', newline='', encoding="utf-8") **as** csv\_file:

field\_names = ['Title of the Research Paper', 'Link to the Research Paper', 'Published Date', 'Authors',

'Pureportal Profile Link']

csv\_writer = csv.DictWriter(csv\_file, fieldnames=field\_names)

csv\_writer.writeheader()

**for** record **in** publication\_results:

csv\_writer.writerow(record)

csv\_file.close()

**print**("A file named records.csv has been generated.")

## Calling the Crawler function

publications\_url = 'https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/'

profiles\_url = 'https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/persons/'

website\_url = 'https://pureportal.coventry.ac.uk'

# Create a scheduler object

scheduler = sched.scheduler(time.time, time.sleep)

**while** True:

**print**(

"----------------------------------------------------------------------------------------------------------------------------------------------")

user\_input = input(

"Below are your options to run the crawler. **\n** a. Run the crawler manually.(This will crawl the data now and create a csv file) **\n** b. Start the scheduler.(This will run every Sunday and will update the csv file with new entries. No manual input needed.)**\n\n** Make a your choice a or b and provide the same as input.**\n**").lower()

**if** user\_input == 'a':

**print**("You selected to run the crawler manually.")

crawler(publications\_url, profiles\_url)

**break**

**elif** user\_input == 'b':

**print**("You selected to schedule the crawler run.")

**while** True:

# Calculate the delay until the next Sunday

now = datetime.datetime.now()

delay = (**7** - now.weekday()) % **7** # delay until next Sunday

# Schedule the function to be called on the next Sunday

scheduler.enter(delay \* **24** \* **60** \* **60**, **1**, crawler, (publications\_url, profiles\_url))

# Uncomment the below code for testing the scheduler. This executes after 20 seconds

# scheduler.enter(20, 1, crawler, (publications\_url, profiles\_url))

# Starting the scheduler

scheduler.run()

# Updating the Inverted Index with new data

inverted\_index()

# Wait for the next event to be scheduled

time.sleep(**60**) # specified in seconds

**else**:

**print**("You gave an Invalid Input. **\n**Please choose either (a) or (b).**\n**")

### InvertedIndex.py

## Importing all the necessary libraries and components

**import** **ssl**

**import** **os**

**import** **re**

**import** **nltk**

**import** **pandas** **as** **pd**

**from** **nltk** **import** word\_tokenize

**from** **nltk.stem.porter** **import** PorterStemmer

# Adding this to skip the SSL certificate check to download stopwords from the nltk package

**try**:

\_create\_unverified\_https\_context = ssl.\_create\_unverified\_context

**except** **AttributeError**:

**pass**

**else**:

ssl.\_create\_default\_https\_context = \_create\_unverified\_https\_context

nltk.download('punkt')

nltk.download('stopwords')

**from** **nltk.corpus** **import** stopwords

## Function to create an inverted index by fetching the data from records.csv

**def** **inverted\_index**():

**print**("**\n**Starting to create an Inverted Index.")

## Reading the data from csv file and creating a dataframe

**if** os.path.isfile('records.csv'):

dataframe = pd.read\_csv("records.csv")

## Exploring the dataframe

**print**(

"----------------------------------------------------------------------------------------------------------------------------------------------")

**print**("No of research papers found: ", len(dataframe))

**print**(

"----------------------------------------------------------------------------------------------------------------------------------------------")

**print**(dataframe.count())

**print**(

"----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------")

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.width', None)

**print**(dataframe.head())

**print**(

"----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------")

## Pre-processing data

clean\_titles = []

stemmer = PorterStemmer()

title\_tokens = []

**for** title **in** dataframe['Title of the Research Paper']:

# Removing non-ASCII characters

title = re.sub(r'[^\x00-\x7F]+', '', title)

# Removing mentions (starting with '@')

title = re.sub(r'@\w+', '', title)

# Converting to lowercase

title = title.lower()

# Removing punctuation marks

title = re.sub(r'[^\w\s]', '', title)

# Removing stop words

stop\_words = set(stopwords.words('english'))

title = ' '.join(word **for** word **in** title.split() **if** word **not** **in** stop\_words)

clean\_titles.append(title)

**for** title **in** clean\_titles:

# Creating word tokens

tokenized\_title = word\_tokenize(title)

# Stemming the word tokens

stemmed\_title = [stemmer.stem(word) **for** word **in** tokenized\_title **if** word.isalpha()]

processed\_title = ' '.join(stemmed\_title)

title\_tokens.append(processed\_title)

# Uncomment the below section to print the titles of the article after pre-processing

# for title in title\_tokens:

# print(title)

## Creating an Inverted Index

# Creating an empty dictionary to hold the inverted index

inverted\_index = {}

# iterate through each title and tokenize it

**for** i, title **in** enumerate(title\_tokens):

tokens = title.split()

# iterate through each token and update the inverted index

**for** token **in** tokens:

**if** token **in** inverted\_index:

inverted\_index[token].append(i)

**else**:

inverted\_index[token] = [i]

# Uncomment the below section to print the inverted index

**for** token **in** inverted\_index:

**print**(token + ":", inverted\_index[token])

**print**("Inverted Index is created.**\n**")

**return** inverted\_index

**else**:

**print**(

"The records.csv file is not present at the moment. Run the crawler to generate the file and then execute this program.")

# Uncomment the below section to call the inverted index manually

inverted\_index()

### QuerySearch.py

**import** **pandas** **as** **pd**

**from** **nltk.tokenize** **import** word\_tokenize

**from** **nltk.corpus** **import** stopwords

**from** **spellchecker** **import** SpellChecker

**from** **nltk.stem.porter** **import** PorterStemmer

**from** **InvertedIndex** **import** inverted\_index

# Taking the user input

**while** True:

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

user\_input = input("**\n**Please select one the options from below. **\n** a. Search for article published under **\"**Research Centre for Computational Science and Mathematical Modelling**\"** **\n** b. Exit**\n**")

**if** user\_input == 'a':

# Variable initializations

porter\_stemmer = PorterStemmer()

stop\_words = set(stopwords.words('english'))

matched\_words = []

corrected\_search\_string = []

# Creating instance of SpellChecker class

spell\_checker = SpellChecker()

# Getting the inverted index and reading the csv file in dataframe

document\_index = inverted\_index()

dataframe = pd.read\_csv("records.csv")

search\_string = input("Enter the article name you are looking for: ")

# Spell-checking the search string

corrected\_search\_string = ''

**for** word **in** search\_string.split():

corrected\_search\_string += spell\_checker.correction(word) + ' '

search\_string = corrected\_search\_string.strip()

**print**("Query after spell check: ", search\_string)

## Pre-processing the input data

# Tokenize the search string

tokenized\_words = word\_tokenize(search\_string.lower())

# Stem and remove stop words and non-alphabetic words

stemmed\_words = [porter\_stemmer.stem(word) **for** word **in** tokenized\_words **if** word **not** **in** stop\_words **and** word.isalpha()]

# Join the stemmed words and split into a list

processed\_query = ' '.join(stemmed\_words).split()

# Print the pre-processed query

**print**("Processed Query tokens:", processed\_query)

# Searching for the query in the document index

**for** word **in** processed\_query:

**if** word **in** document\_index:

matched\_words.append(word)

# Finding the documents that match the query

matched\_docs = set()

**for** word **in** matched\_words:

**for** doc\_id **in** document\_index[word]:

matched\_docs.add(doc\_id)

# Displaying the matched documents

**if** len(matched\_docs) > **0**:

**print**("Found", len(matched\_docs), "documents matching the search query:")

**for** doc\_id **in** matched\_docs:

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

**print**("Title:", dataframe.iloc[doc\_id]['Title of the Research Paper'])

**print**("Link:", dataframe.iloc[doc\_id]['Link to the Research Paper'])

**print**("Published Date:", dataframe.iloc[doc\_id]['Published Date'])

**print**("Authors:", dataframe.iloc[doc\_id]['Authors'])

**print**("Authors Profile Link:", dataframe.iloc[doc\_id]['Pureportal Profile Link'])

**else**:

**print**("No documents found matching the search query.")

**elif** user\_input == 'b':

**break**

**else**:

**print**("You gave an Invalid Input. **\n**Please choose either (a) or (b).**\n**")

## Task 2: Document Clustering

### DocumentClustering.py

**import** **feedparser**

**import** **nltk**

**from** **spellchecker** **import** SpellChecker

**from** **nltk.stem.porter** **import** PorterStemmer

**from** **sklearn.cluster** **import** KMeans

**from** **sklearn.metrics** **import** silhouette\_score

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

# Define categories for each feed

categories = {

'http://rss.cnn.com/rss/edition\_sport.rss': 'Sports',

'http://feeds.bbci.co.uk/news/technology/rss.xml': 'Technology',

'http://feeds.bbci.co.uk/news/entertainment\_and\_arts/rss.xml': 'Entertainment',

'http://feeds.bbci.co.uk/news/politics/rss.xml': 'Politics',

'http://rss.cnn.com/rss/money\_news\_international.rss': 'Business'

}

# Define lists for each category of documents

sports\_list = []

technology\_list = []

entertainment\_list = []

politics\_list = []

business\_list = []

# Define function to get text from RSS feeds

**def** **get\_text**(feed\_url, category):

feed\_parser = [feedparser.parse(feed\_url)]

**for** feeder **in** feed\_parser:

**for** entry **in** feeder.entries:

**if** category == 'Sports':

sports\_list.append(entry.title)

**elif** category == 'Technology':

technology\_list.append(entry.title)

**elif** category == 'Entertainment':

entertainment\_list.append(entry.title)

**elif** category == 'Politics':

politics\_list.append(entry.title)

**elif** category == 'Business':

business\_list.append(entry.title)

# Define a function to tokenize and stem the text

**def** **tokenize\_and\_stem**(text):

tokens = nltk.word\_tokenize(text)

stemmer = PorterStemmer()

stems = [stemmer.stem(t) **for** t **in** tokens]

**return** stems

# Define RSS feeds

feed\_links = list(categories.keys())

# Get text from RSS feeds

**print**("**\n**Starting to fetch news articles from various rss feeds.")

**for** feed **in** feed\_links:

get\_text(feed, categories[feed])

**print**("News articles fetched.")

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

**print**("Total news articles fetched from Sports category:", len(sports\_list))

**print**("Total news articles fetched from Technology category:", len(technology\_list))

**print**("Total news articles fetched from Entertainment category:", len(entertainment\_list))

**print**("Total news articles fetched from Politics category:", len(politics\_list))

**print**("Total news articles fetched from Business category:", len(business\_list))

# Combine text from all feeds into one list

text\_list = sports\_list + technology\_list + entertainment\_list + politics\_list + business\_list

**print**("**\n**Total news articles fetched: ", len(text\_list))

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

# Define number of clusters

k = len(categories)

# Create bag of words

vectorizer = CountVectorizer(stop\_words='english')

X = vectorizer.fit\_transform(text\_list)

# Cluster documents using K-means

km = KMeans(n\_clusters=k, init='k-means++', max\_iter=**100**, n\_init=**1**)

km.fit(X)

# Label the clusters

labels = km.labels\_

cluster\_labels = {}

**for** i, category **in** enumerate(categories.values()):

cluster\_labels[i] = category

clusters = {}

**for** i **in** range(len(labels)):

**if** labels[i] **in** clusters:

clusters[labels[i]].append(text\_list[i])

**else**:

clusters[labels[i]] = [text\_list[i]]

**for** i **in** range(k):

cluster\_docs = clusters[i]

label = cluster\_labels.get(i, f"Cluster {i+1}")

**print**(f'{label}: {len(cluster\_docs)} documents')

**for** doc **in** cluster\_docs:

**print**(doc)

**print**()

# Use the model to predict a new document

**while** True:

# Taking input from user

user\_input = input("**\n**Please select one the options from below. **\n**a. Search for a cluster using news headline **\n**b. Exit**\n**")

**if** user\_input == 'a':

query = input("Enter a news headline to find it's cluster.**\n**")

# Creating instance of SpellChecker class

spell\_checker = SpellChecker()

# Spell-checking the search string

corrected\_search\_string = ''

**for** word **in** query.split():

correction = spell\_checker.correction(word)

**if** correction **is** **not** None:

corrected\_search\_string += correction + ' '

query = corrected\_search\_string.strip()

**print**("Query after spell check: ", query)

vectorized\_query = vectorizer.transform([query])

prediction = km.predict(vectorized\_query)[**0**]

**print**("Inertia: ", km.inertia\_)

silhouette\_avg = silhouette\_score(X, labels)

**print**("Silhouette Score: ", silhouette\_avg)

**print**(f'The new document belongs to cluster {cluster\_labels[prediction]} ({cluster\_labels[prediction]})')

**elif** user\_input == 'b':

**break**

**else**:

**print**("You gave an Invalid Input. **\n**Please choose either (a) or (b).**\n**")