(7071CEM)

Coursework

Information Retrieval

MODULE LEADER: Seyed Mousavi

Student Name: Sahil Kalpesh Kumbhar

SID: 13140139

Table of Contents

[Task 1 – Search Engine 3](#_Toc131162628)

[1. Crawler 3](#_Toc131162629)

[2. Indexer 6](#_Toc131162634)

[3. Query Processor 7](#_Toc131162635)

[Task 2 – Document Clustering 10](#_Toc131162636)

[APPENDIX 13](#_Toc131162637)

[Part-1 Code for Search engine 13](#_Toc131162638)

[Part-2 Code for Document clustering 23](#_Toc131162639)

[References 27](#_Toc131162640)

# Task 1 – Search Engine

### Crawler

#### Overview

The crawler is designed to extract information about publications from a website. The crawler retrieves data from the website <https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/> and extracts publications published by members of CSM only.

A screenshot of a computer

Description automatically generated

Graphical user interface, application

Description automatically generated



#### 1.2 Publication Information Collected

The crawler extracts various information about each publication, including the title, link, publication date, names of authors, and Pureportal profile link of each author.

**Table

Description automatically generated**

#### 1.3 Pre-processing Tasks

Before passing data to the Indexer, the data undergoes pre-processing. The preprocessing tasks carried out are removing non-ASCII characters, mentions, punctuation marks, and stop words, as well as stemming words.

#### 1.4 Crawler Operation

The crawler operates manually, requiring the user to execute the script within a Python environment. It begins by fetching the seed URL provided in the code and searching for all available publications. It then extracts the relevant information about each publication, moves on to the next page, and keeps track of visited URLs to avoid revisiting the same URLs again. Finally, the crawled data is stored in a list of dictionaries, with each dictionary representing a publication.

**Graphical user interface, text, application, email

Description automatically generated**

### Indexer

#### Overview

This user manual provides an overview of the index used in a search engine. The index is responsible for storing and organizing the content of documents so they can be searched efficiently. It stores information about each document, such as the words and phrases it contains and their location. The index uses an inverted data structure to allow for fast retrieval of documents containing a particular word or phrase. In addition, the index stores information such as the number of times each word or phrase appears in the document, which is used to rank search results. Understanding the index is crucial for building or optimizing search engines.

#### Implementation Method

The inverted index has been implemented using Python's standard libraries and functions,

rather than Elastic Search.

#### 2.3 Data Structure

The inverted index is implemented using a dictionary, where the keys are words and the values are lists of the positions where the word occurs in the list of publication names. This is an example of an inverted index based on a term-document matrix.

#### 2.4. Incremental or Non-incremental:

The index is non-incremental, meaning it is constructed from scratch every time the crawler is run.

#### 2.5 Content of the Index:

The index contains a dictionary with each word in the publication names as a key, and a list of positions where the word occurs in the publication names as a value.

Graphical user interface, text, application

Description automatically generated

#### 2.6 How It Works

The inverted index is created by first preprocessing the publication names to remove stop words, punctuation, and other irrelevant characters. Then, each publication name is tokenized, and the position of each token is added to the index. The index is then used to efficiently search for documents that contain specific words or combinations of words.

# 3. Query Processor

#### 3.1 Overview

The Query Processor is a tool that takes a user's input query and retrieves relevant results from a database or search engine. It performs pre-processing tasks to clean and tokenize the query, and may use ranking algorithms to determine the relevance of each result.

#### 3.2 Preprocessing Tasks

The pre-processing tasks applied to a given query are:

* Removing non-ASCII characters
* Removing mentions (starting with '@')
* Converting to lowercase
* Removing punctuation marks
* Removing stop words
* Stemming words

In addition to the preprocessing tasks, the user's input query is checked for misspelled words and corrected using the autocorrect module. The Speller class is imported and used to create an instance, which is then used to correct any misspelled words in the input query. The corrected query is then passed through the rest of the preprocessing pipeline.

#### 3.3 Query Types Supported by the System

The system does not support Boolean queries. It accepts keyword queries similar to Google, without the need for Boolean operators.

#### 3.4 Query Conversion for Elastic Search (Not Applicable)

Elasticsearch is not used in this system, so there is no need to convert a user query to an appropriate Elasticsearch query.

#### 3.5 Demonstration of the Running System

To run the system, follow these steps:

1. Open the Python script file in a Python environment such as Jupyter Notebook or PyCharm.
2. Run the script.
3. Enter the publication name in the input prompt.

The system will process the input query and return the top 6 related research papers.

Sample input queries to test the system include "natural language processing," "machine learning," "deep learning," "computer vision," and "neural networks."

Screenshots of the Running System:

input query: "machine learn"

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text

Description automatically generated

input query: "extnd plant"

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated with low confidence

input query: " dibetic retinpathy”

Graphical user interface, application

Description automatically generated

Text

Description automatically generated with low confidence

#### 3.6 Explanation of System Functionality

The system applies pre-processing tasks to clean and tokenize publication names before creating an inverted index of the tokens to speed up the search process. It uses the TF-IDF vectorizer to create a matrix of publication names and their corresponding TF-IDF scores. When a user query is entered, it is processed in the same way as publication names, and match scores are calculated using the cosine similarity measure. The system returns the top 6 publications with the highest match scores. The system also performs spelling correction using the Autocorrect library to improve the accuracy of the search results.

# Task 2 – Document Clustering

#### Overview

Document clustering is the process of grouping similar documents together based on their content. It is a popular technique used in information retrieval, natural language processing, and data mining. The goal of document clustering is to discover the underlying structure of a collection of documents and to group them into meaningful clusters. This can be useful for tasks such as document organization, information retrieval, and sentiment analysis. In this user manual, we describe a program that uses the K-Means clustering method to cluster news articles based on their content and category.

#### Input Documents Collection:

This program collects news articles from RSS feeds of different news websites related to the categories of sports, technology, and climate. The program extracts the article titles and stores them in a list along with their respective category.

The URLs from which news articles are collected are:



"sports": ["https://feeds.bbci.co.uk/sport/rss.xml", "https://www.espn.com/espn/rss/news"],

"technology": ["https://feeds.bbci.co.uk/news/technology/rss.xml", "https://www.wired.com/feed/rss"],

"climate": ["https://feeds.bbci.co.uk/news/science-environment-56837908/rss.xml", "https://theconversation.com/uk/topics/climate-change-27"]

And the total number of documents collected are  
A picture containing shape

Description automatically generated

#### Clustering Method Used and Performance Measurement:

The program uses the K-Means clustering method with a user-defined number of clusters (default value = 3) for clustering the preprocessed news articles. The performance of the clustering algorithm is measured using the Silhouette Score. The Silhouette Score measures the similarity of the data points within a cluster compared to other clusters. A score closer to 1 indicates better cluster separation.

The Silhouette Score for test data is shown in the following image Graphical user interface

Description automatically generated with medium confidence

#### Type of Clustering Used

The program uses flat clustering, which assigns each data point to exactly one cluster. Also, the program uses hard clustering, which means that each data point belongs to one and only one cluster.

#### Accuracy and Robustness Demonstration

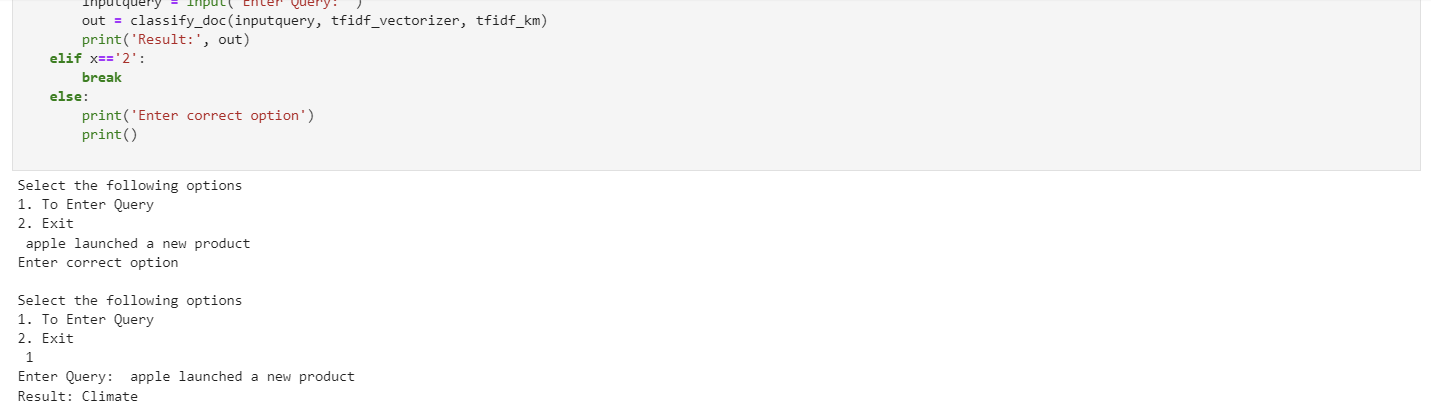
The program has been tested on various news articles related to the three categories. The program's accuracy and robustness can be seen by the Silhouette Score of the clustering method, which indicates how well the clustering separates the data points. Users can also test the program's accuracy by entering their query and checking the program's output.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



Background pattern

Description automatically generated with low confidence

#### Brief Explanation of How It Works

The program first preprocesses the news articles by tokenizing them, removing stop words and non-alphabetic tokens, and lemmatizing them. Then, the program vectorizes the preprocessed articles using the TF-IDF vectorization method. After that, the program trains the K-Means clustering model on the vectorized articles. The user can enter their query, and the program will preprocess the query, convert it into a numerical vector using the vectorizer, and predict the cluster of the document using the trained K-Means model. Finally, the program returns the predicted category of the document.

#### Optional Points

The results appended in the Accuracy and Robustness Demonstration have some of the results correctly predicted and one of the image shows wrong prediction, this is because the data is constantly changing and also the length of training dataset is much smaller. The accuracy can be increased by appending more and more data to the dataset.

# APPENDIX

### Part-1 Code for Search engine

import requests

from bs4 import BeautifulSoup

import time

from collections import defaultdict

import re

import csv

from urllib.parse import urljoin

import os

import string

import pandas as pd

from urllib.parse import urljoin

from urllib.robotparser import RobotFileParser

import schedule

from sklearn.feature\_extraction.text import TfidfVectorizer

import itertools

from spellchecker import SpellChecker

import nltk

from nltk.stem.porter import PorterStemmer

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

# Downloading necessary resources for pre-processing

nltk.download('stopwords')

nltk.download('punkt')

res=requests.get('https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/')

print(res)

soup=BeautifulSoup(res.text,'html.parser')

#print(soup.prettify)

# Fetch the robots.txt file

rp = RobotFileParser()

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

rp.set\_url(urljoin(root\_url, "/robots.txt"))

rp.read()

def is\_allowed(url):

return rp.can\_fetch("\*", url)

def crawl\_persons():

csm\_member\_links=set()

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

profile=requests.get(url)

if is\_allowed(url):

profile\_soup=BeautifulSoup(profile.text,'html.parser')

tiles=profile\_soup.select('.result-container')

for profiles in tiles:

profile\_links=profiles.find("a", class\_="link person").get('href')

csm\_member\_links.add(profile\_links)

return csm\_member\_links

def crawl\_author\_pub\_and\_count(pub\_author\_url\_count):

author\_pub\_and\_count = dict()

for key in list(pub\_author\_url\_count.keys()):

authors\_portal\_page = requests.get('https://pureportal.coventry.ac.uk/en/persons/'+key)

author\_soup = BeautifulSoup(authors\_portal\_page.text, "html.parser")

author\_name=author\_soup.find('h1').text

author\_pub\_and\_count[author\_name] = pub\_author\_url\_count[key]

#print(author\_pub\_and\_count)

with open('article\_counts.csv', 'w', newline='') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(['Author Name', 'No. of Articles Published'])

for author, count in author\_pub\_and\_count.items():

writer.writerow([author, count])

def crawler(root, page\_count):

search\_list=[]

visited\_urls = set()

queue = [root] #this is the queue which initially contains the given seed URL

count = 0

csm\_member\_links=crawl\_persons()

while(queue!=[] and count < page\_count):

url = queue.pop(0)

if is\_allowed(url):

if url in visited\_urls:

continue

visited\_urls.add(url)

print("fetching " + url)

count +=1

page = requests.get(url)

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

tiles=soup.select('.result-container')

c=1

for subsections in tiles:

list\_of\_records={}

members = subsections.find\_all("a", class\_="link person")

is\_member\_csm=False

for member in members:

#print(member.get('href'))

if member.get('href') in csm\_member\_links:

#print('----------')

#print(member.get('href'))

is\_member\_csm=True

break

#print(is\_member\_csm)

#count+=1

#member\_link=members.find('a',class\_="link person").get('href')

#print(member\_link)

if is\_member\_csm:

p\_title=subsections.find('a',{'class':'link'}).get\_text()

p\_link=subsections.find('a',{'class':'link'}).get('href')

p\_date=subsections.find('span',{'class':'date'}).get\_text()

p\_auth\_names=[]

p\_auth\_portals=[]

for auth\_info in subsections.findAll('a',{'class':'link person'}):

p\_auth\_name=auth\_info.get\_text()

p\_auth\_portal\_link=auth\_info.get('href')

p\_auth\_names.append(p\_auth\_name)

p\_auth\_portals.append(p\_auth\_portal\_link)

##for author and their publishing count

p\_auth\_portal\_link=p\_auth\_portal\_link.split('/')[-1]

if p\_auth\_portal\_link in pub\_author\_url\_count:

pub\_author\_url\_count[p\_auth\_portal\_link] += 1

else:

pub\_author\_url\_count[p\_auth\_portal\_link] = 1

list\_of\_records['Name of Publication']=p\_title

list\_of\_records['Publication Link']=p\_link

list\_of\_records['Publication Date']=p\_date

list\_of\_records['List of Authors']=p\_auth\_names

list\_of\_records['Author Pureportal Profile Link']=p\_auth\_portals

search\_list.append(list\_of\_records)

c=c+1

else:

print(f'{url} is not allowed to be crawled')

#print(author\_pub\_and\_count)

for nextpage in soup.findAll('a',attrs={"class": "step"}):

new\_url = nextpage.get('href')

if(new\_url != None and new\_url != '/'):

new\_url = urljoin(url, new\_url)

#print("new\_url is : ", new\_url) #uncomment the print statement to see the urls

queue.append(new\_url)

# Sleep for a few seconds to avoid hitting the server too fast

time.sleep(rp.crawl\_delay('\*'))

headers=['Name of Publication','Publication Link','Publication Date','List of Authors','Author Pureportal Profile Link']

with open('output.csv', mode='w', newline='', encoding='utf-8') as output\_file:

writer = csv.DictWriter(output\_file, fieldnames=headers)

writer.writeheader()

for record in search\_list:

writer.writerow(record)

#print(pub\_author\_url\_count)

crawl\_author\_pub\_and\_count(pub\_author\_url\_count)

def run\_crawler():

crawl\_persons()

print("Crawler running at", time.strftime("%Y-%m-%d %H:%M:%S"))

crawler('https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/',10)

#if it is running on friday

while True:

schedule.run\_pending()

time.sleep(60) # Wait for 1 minute

while True:

x = input('Select the following options\n1. To Run Crawler \n2. Schedule the crawler to run every Friday\n3. Exit\n')

if x == '1':

pub\_author\_url\_count=dict()

crawl\_persons()

crawler('https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/', 10)

print("Crawler running at", time.strftime("%Y-%m-%d %H:%M:%S"))

print('crawling completed')

#crawl\_author\_pub\_and\_count(auth\_dict)

print()

print('2 files generated')

current\_dir = os.path.abspath(os.curdir)

file\_path = os.path.join(current\_dir, 'output.csv')

print("CSM website crawled output File path:", file\_path)

current\_dir = os.path.abspath(os.curdir)

file\_path = os.path.join(current\_dir, 'article\_counts.csv')

print("check Authors and their publication counts File path:", file\_path)

print()

elif x == '2':

schedule.every().friday.at('01:00').do(run\_crawler)

print('crawler scheduled to run every friday at 01:00 AM')

print()

elif x=='3':

break

else:

print('Choose the correct option')

print()

os.path.isfile('output.csv')

output=pd.read\_csv("output.csv")

output.head(2)

processed\_pub\_names = []

def preprocess\_text(pub\_name):

# Removing non-ASCII characters

pub\_name = pub\_name.encode('ascii', 'ignore').decode()

# Removing mentions (starting with '@')

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

# Converting to lowercase

pub\_name = pub\_name.lower()

# Removing punctuation marks

pub\_name = pub\_name.translate(str.maketrans('', '', string.punctuation))

# Removing stop words

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

pub\_name = ' '.join(word for word in pub\_name.split() if word not in stop\_words)

# Stemming words

stemmer = PorterStemmer()

pub\_name = ' '.join(stemmer.stem(word) for word in pub\_name.split())

return pub\_name

# apply the pre-processing function to each publication name and store the results in the processed\_pub\_names list

for pub\_name in output['Name of Publication']:

processed\_pub\_name = preprocess\_text(pub\_name)

processed\_pub\_names.append(processed\_pub\_name)

# create an empty dictionary to hold the inverted index

inverted\_index = {}

# iterate through each publication name and tokenize it

for i, pub\_name in enumerate(processed\_pub\_names):

tokens = pub\_name.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]

# print top 5 entries in the inverted index

top\_five = dict(itertools.islice(inverted\_index.items(), 5))

print(top\_five)

tfid = TfidfVectorizer(use\_idf=True, smooth\_idf=True, norm=None, binary=False)

tfid\_matrix = tfid.fit\_transform(processed\_pub\_names)

terms = tfid.get\_feature\_names()

dense\_matrix = tfid\_matrix.toarray()

dense\_list = dense\_matrix.tolist()

# Create a dataframe with the tf-idf values

tf\_idf\_df = pd.DataFrame(dense\_list, columns=terms)

from autocorrect import Speller

spell = Speller(lang='en')

inputquery = input('Enter Publication Name: ')

enteredquery=inputquery

corrected\_query=[]

for word in inputquery.split(' '):

corrected\_word = spell(word)

corrected\_query.append(corrected\_word)

inputquery=' '.join(corrected\_query)

clean\_inputquery = preprocess\_text(inputquery).split()

print(clean\_inputquery)

# Calculate match scores

match\_scores = [sum(tf\_idf\_df.iloc[i][j] for j in clean\_inputquery if j in tf\_idf\_df.columns) for i in range(len(processed\_pub\_names))]

# Check if there are any matches

if all(score == 0 for score in match\_scores):

print("No related research paper found.")

else:

# Sort results by match score and select top 6

top\_results = sorted(enumerate(match\_scores), key=lambda x: x[1], reverse=True)[:6]

# Print results

print('Entered Query:',enteredquery)

print('showing results for',' '.join(clean\_inputquery))

print()

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

for i, score in top\_results:

row = output.iloc[i]

publication\_name = row['Name of Publication'].strip()

publication\_link = row['Publication Link'].strip()

publication\_date = row['Publication Date'].strip()

authors = row['List of Authors'].strip()

authors\_portal\_links = row['Author Pureportal Profile Link'].strip()

print('Publication Name:', publication\_name)

print('Publication Link:', publication\_link)

print('Publication Date:', publication\_date)

print('Authors:', authors)

print('Author Portal Links:', authors\_portal\_links)

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

### Part-2 Code for Document clustering

import feedparser

import random

import nltk

import re

import string

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

# Define the URLs for the news feeds

urls = {

"sports": ["https://feeds.bbci.co.uk/sport/rss.xml", "https://www.espn.com/espn/rss/news"],

"technology": ["https://feeds.bbci.co.uk/news/technology/rss.xml", "https://www.wired.com/feed/rss"],

"climate": ["https://feeds.bbci.co.uk/news/science-environment-56837908/rss.xml", "https://theconversation.com/uk/topics/climate-change-27"]

}

# Define the number of clusters

num\_clusters = 3

# Define the preprocessing function

def preprocess\_text(text):

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

lemmatizer = WordNetLemmatizer()

tokens = word\_tokenize(text.lower())

filtered\_tokens = [lemmatizer.lemmatize(token) for token in tokens if token.lower() not in stop\_words and token.isalpha()]

return ' '.join(filtered\_tokens)

# Fetch and preprocess the news articles

news\_articles = []

for category, feed\_urls in urls.items():

for url in feed\_urls:

feed = feedparser.parse(url)

for entry in feed.entries:

news\_articles.append((category, preprocess\_text(entry.title)))

print('Total documents collected: ',len(news\_articles))

# Shuffle the news articles

random.shuffle(news\_articles)

# Split the news articles into training and testing sets

train\_size = int(0.8 \* len(news\_articles))

train\_data = [article[1] for article in news\_articles[:train\_size]]

test\_data = [article[1] for article in news\_articles[train\_size:]]

# Vectorize the news articles using TF-IDF

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_X = tfidf\_vectorizer.fit\_transform(train\_data)

# Train a KMeans model on the training data

tfidf\_km = KMeans(n\_clusters=num\_clusters)

tfidf\_km.fit(tfidf\_X)

# Test the model on the testing data

test\_vectors = tfidf\_vectorizer.transform(test\_data)

predicted\_labels = tfidf\_km.predict(test\_vectors)

# Print the performance metrics

print("TF-IDF Performance Metrics:")

print("Silhouette Score:", silhouette\_score(test\_vectors, predicted\_labels))

def classify\_doc(doc, vectorizer, km):

# Preprocess the document

preprocessed\_doc = preprocess\_text(doc)

# Convert the document into a numerical vector using the vectorizer

doc\_vec = vectorizer.transform([preprocessed\_doc])

# Predict the cluster of the document using the KMeans model

cluster = km.predict(doc\_vec)[0]

# Return the predicted label of the document

if cluster == 0:

return "Sports"

elif cluster == 1:

return "Technology"

else:

return "Climate"

while True:

x = input('Select the following options\n1. To Enter Query \n2. Exit\n')

if x=='1':

inputquery=input('Enter Query: ')

out=classify\_doc(inputquery, tfidf\_vectorizer, tfidf\_km)

print('result: ',out)

elif x=='2':

break

else:

print('enter correct option')

print()

# References

<https://files.coventry.aula.education/85a3ae7e148d70bd0132ff5ede9abab4crawler_sample_code_to_start_with.txt>

JCharisTech. (2019). *Data Science Tools - Spell Checker and Auto Correction with Python[2019]*

<https://www.youtube.com/watch?v=rjXeG0aT-7w&t=275s>

GeeksforGeeks. (2020). *Create Inverted Index for File using Python.*

<https://www.geeksforgeeks.org/create-inverted-index-for-file-using-python/>