

(7071CEM)

Coursework

Information Retrieval

MODULE LEADER: Seyed Mousavi

Student Name: Sahil Kalpesh Kumbhar

SID: 13140139

Table of Contents

Task 1 – Search Engine	3
1. Crawler	3
2. Indexer	5
3. Query Processor	6
Task 2 – Document Clustering	9
APPENDIX	12
Part-1 Code for Search engine	12
Part-2 Code for Document clustering	22
References	26

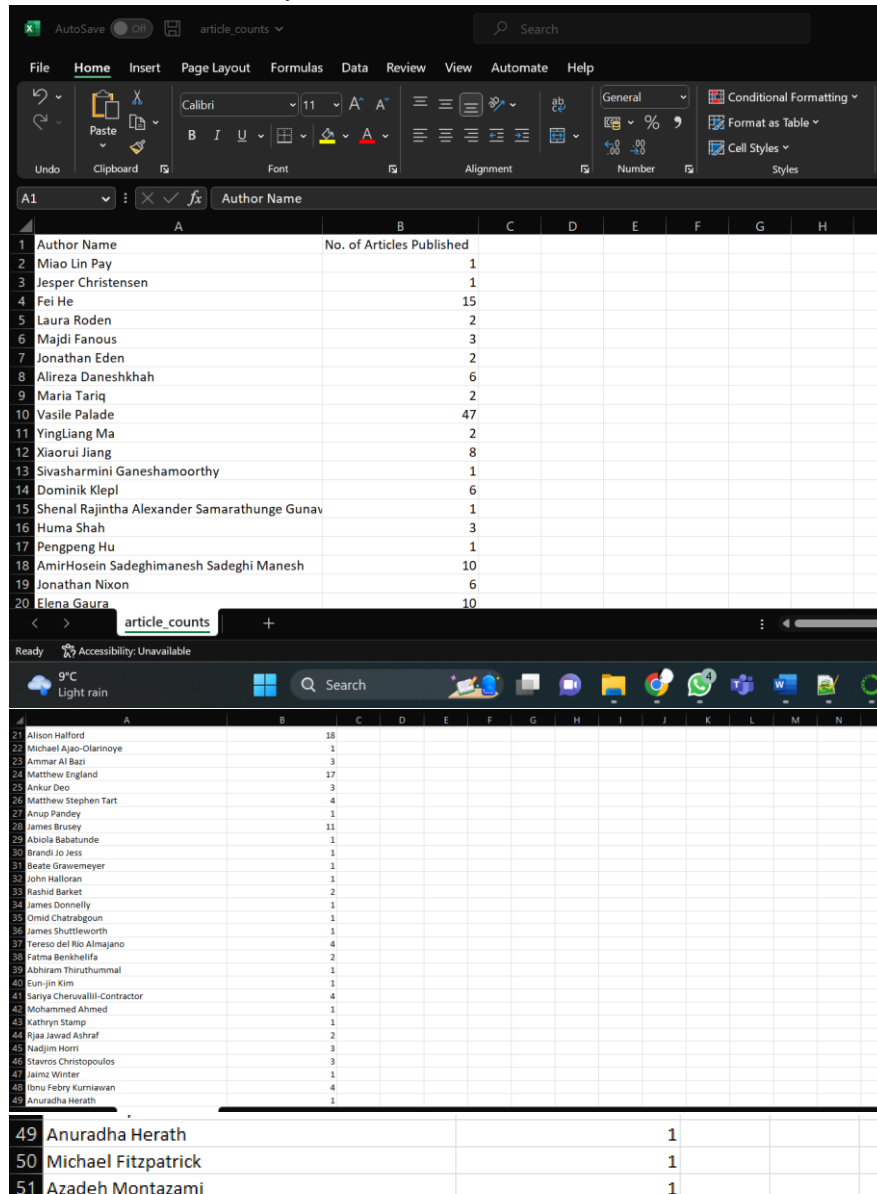
Task 1 – Search Engine

1. crawler

1.1 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.



Author Name	No. of Articles Published
Miao Lin Pay	1
Jesper Christensen	1
Fei He	15
Laura Roden	2
Majdi Fanous	3
Jonathan Eden	2
Alireza Daneshkhan	6
Maria Tariq	2
Vasile Palade	47
YingLiang Ma	2
Xiaorui Jiang	8
Sivasharmini Ganesamoorthy	1
Dominik Klepl	6
Shenal Rajintha Alexander Samarathunge Gunav	1
Huma Shah	3
Pengpeng Hu	1
AmirHosein Sadeghimanesh Sadeghi Manesh	10
Jonathan Nixon	6
Elena Gaura	10
Alison Halford	18
Michael Ajao-Olarinoye	1
Ammar Al Bazi	3
Matthew England	17
Ankur Des	3
Matthew Stephen Tart	4
Anup Pandey	1
James Brusey	11
Abiola Babatunde	1
Brandi Jo Jess	1
Seate Grauemeyer	1
John Halloran	1
Rashid Barket	2
James Donnelly	1
Omid Chatrabgoun	1
James Shuttleworth	1
Tereso del Rio Almajano	4
Fatma Benkhalifa	2
Abhiram Thiruthummal	1
Eun-jin Kim	1
Seriyu Cheruvallil-Contractor	4
Mohammed Ahmed	1
Kathryn Stamp	1
Rjaa Jawad Ashraf	2
Nadim Horri	3
Stavros Christopoulos	3
Jaime Winter	1
Ibnu Febry Kurniawan	4
Anuradha Herath	1
Anuradha Herath	1
Michael Fitzpatrick	1
Azadeh Montazami	1

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.

```
os.path.isfile('output.csv')
output=pd.read_csv("output.csv")
output.head(2)
```

	Name of Publication	Publication Link	Publication Date	List of Authors	Author Pureportal Profile Link
0	An Extended Plant Circadian Clock Model for Ch...	https://pureportal.coventry.ac.uk/en/publicati...	9 Jan 2023	['Pay, M. L.', 'Christensen, J.', 'He, F.', 'R...	['https://pureportal.coventry.ac.uk/en/persons...
1	Challenges and prospects of climate change imp...	https://pureportal.coventry.ac.uk/en/publicati...	25 Feb 2023	['Fanous, M.', 'Eden, J.', 'Daneshkhah, A.']	['https://pureportal.coventry.ac.uk/en/persons...

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.

```
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()
```

Select the following options

1. To Run Crawler
2. Schedule the crawler to run every Friday
3. Exit

1

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

fetching https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/?page=1

fetching https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/?page=2

fetching https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/?page=3

fetching https://pureportal.coventry.ac.uk/en/organisations/research-centre-for-computational-science-and-mathematical-modell/publications/?page=4

Crawler running at 2023-03-30 03:04:34

crawling completed

2 files generated

File path: C:\Users\Sahil\Documents\Information retrieval\output.csv

check output File path: C:\Users\Sahil\Documents\Information retrieval\article_counts.csv

Select the following options

1. To Run Crawler
2. Schedule the crawler to run every Friday
3. Exit

3

2. Indexer

2.1 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.

2.2 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.

```
# 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)
```

{'extend': [0], 'plant': [0, 30], 'circadian': [0], 'clock': [0], 'model': [0, 1, 6, 30, 31, 38, 71, 78, 80, 97]}

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"

```

cated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
warnings.warn(msg, category=FutureWarning)

[14]: 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)

Enter Publication Name: machine learning
['machin', 'learn']
```

```

Entered Query: mchine learn
showing results:

Publication Name: Machine Learning for Computer Algebra
Publication Link: https://pureportal.coventry.ac.uk/en/publications/machine-learning-for-computer-algebra
Publication Date: 2022
Authors: ['Barket, R.', 'del Río, T.', 'England, M.']
Author Portal Links: ['https://pureportal.coventry.ac.uk/en/persons/rashid-barket', 'https://pureportal.coventry.ac.uk/en/persons/tereso-del-r%C3%ADo-almajano
o', 'https://pureportal.coventry.ac.uk/en/persons/matthew-england']
*****
Publication Name: SC-Square: Future Progress with Machine Learning
Publication Link: https://pureportal.coventry.ac.uk/en/publications/sc-square-future-progress-with-machine-learning
Publication Date: 14 Nov 2022
Authors: ['England, M.']
Author Portal Links: ['https://pureportal.coventry.ac.uk/en/persons/matthew-england']
*****
Publication Name: Using Machine Learning in SC2
Publication Link: https://pureportal.coventry.ac.uk/en/publications/using-machine-learning-in-sc2
Publication Date: 23 Aug 2022
Authors: ['del Río, T.']
Author Portal Links: ['https://pureportal.coventry.ac.uk/en/persons/tereso-del-r%C3%ADo-almajano']
*****
Publication Name: A machine learning based software pipeline to pick the variable ordering for algorithms with polynomial inputs
Publication Link: https://pureportal.coventry.ac.uk/en/publications/a-machine-learning-based-software-pipeline-to-pick-the-variable-o
Publication Date: 8 Jul 2020
Authors: ['England, M.']
Author Portal Links: ['https://pureportal.coventry.ac.uk/en/persons/matthew-england']
*****
```

input query: "extnd plant"

```
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)
```

Enter Publication Name: extnd plant
['extend', 'plant']

Entered Query: extnd plant
showing results:

Publication Name: An Extended Plant Circadian Clock Model for Characterising Flowering Time under Different Light Quality Conditions
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/an-extended-plant-circadian-clock-model-for-characterising-flower>
Publication Date: 9 Jan 2023
Authors: ['Pay, M. L.', 'Christensen, J.', 'He, F.', 'Roden, L.']
Author Portal Links: ['<https://pureportal.coventry.ac.uk/en/persons/miao-lin-pay>', '<https://pureportal.coventry.ac.uk/en/persons/jesper-christensen>', '<https://pureportal.coventry.ac.uk/en/persons/fei-he>', '<https://pureportal.coventry.ac.uk/en/persons/laura-roden>']

Publication Name: Data-driven dynamical modelling of a pathogen-infected plant gene regulatory network: A comparative analysis
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/data-driven-dynamical-modelling-of-a-pathogen-infected-plant-gene>
Publication Date: Sep 2022
Authors: ['He, F.']
Author Portal Links: ['<https://pureportal.coventry.ac.uk/en/persons/fei-he>']

Publication Name: Challenges and prospects of climate change impact assessment on mangrove environments through mathematical models
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/challenges-and-prospects-of-climate-change-impact-assessment-on-m>
Publication Date: 25 Feb 2023
Authors: ['Fanous, M.', 'Eden, J.', 'Daneshkhah, A.']
Author Portal Links: ['<https://pureportal.coventry.ac.uk/en/persons/majdi-fanous>', '<https://pureportal.coventry.ac.uk/en/persons/jonathan-eden>', '<https://pureportal.coventry.ac.uk/en/persons/alireza-daneshkhah>']

Publication Name: Diabetic Retinopathy Detection Using Transfer and Reinforcement Learning with Effective Image Preprocessing and Data Augmentation Techniques
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/diabetic-retinopathy-detection-using-transfer-and-reinforcement-l>
Publication Date: 7 Feb 2023
Authors: ['Tariq, M.', 'Palade, V.', 'Ma, Y.']
Author Portal Links: ['<https://pureportal.coventry.ac.uk/en/persons/maria-tariq>', '<https://pureportal.coventry.ac.uk/en/persons/vasile-palade>', '<https://pureportal.coventry.ac.uk/en/persons/yingliang-ma>']

input query: "dibetic retinpathy"

```
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)
```

Enter Publication Name: dibetic retinpathy
['diabet', 'retinpathi']

Entered Query: diabetic retinopathy
showing results:

Publication Name: Diabetic Retinopathy Detection Using Transfer and Reinforcement Learning with Effective Image Preprocessing and Data Augmentation Techniques
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/diabetic-retinopathy-detection-using-transfer-and-reinforcement-1>
Publication Date: 7 Feb 2023
Authors: ['Tariq, M.', 'Palade, V.', 'Ma, Y.']
Author Portal Links: [<https://pureportal.coventry.ac.uk/en/persons/maria-tariq>, '<https://pureportal.coventry.ac.uk/en/persons/vasile-palade>', '<https://pureportal.coventry.ac.uk/en/persons/yingliang-ma>']

Publication Name: Transfer Learning based Classification of Diabetic Retinopathy on the Kaggle EyePACS dataset
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/transfer-learning-based-classification-of-diabetic-retinopathy-on>
Publication Date: 15 Nov 2022
Authors: ['Tariq, M.', 'Palade, V.', 'Ma, Y.']
Author Portal Links: [<https://pureportal.coventry.ac.uk/en/persons/maria-tariq>, '<https://pureportal.coventry.ac.uk/en/persons/vasile-palade>', '<https://pureportal.coventry.ac.uk/en/persons/yingliang-ma>']

Publication Name: An Extended Plant Circadian Clock Model for Characterising Flowering Time under Different Light Quality Conditions
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/an-extended-plant-circadian-clock-model-for-characterising-flower>
Publication Date: 9 Jan 2023
Authors: ['Pay, M. L.', 'Christensen, J.', 'He, F.', 'Rodén, L.']
Author Portal Links: [<https://pureportal.coventry.ac.uk/en/persons/miao-lin-pay>, '<https://pureportal.coventry.ac.uk/en/persons/jesper-christensen>', '<https://pureportal.coventry.ac.uk/en/persons/fei-he>', '<https://pureportal.coventry.ac.uk/en/persons/laura-roden>']

Publication Name: Challenges and prospects of climate change impact assessment on mangrove environments through mathematical models
Publication Link: <https://pureportal.coventry.ac.uk/en/publications/challenges-and-prospects-of-climate-change-impact-assessment-on-m>
Publication Date: 25 Feb 2023
Authors: ['Fanous, M.', 'Eden, J.', 'Daneshkhah, A.']
Author Portal Links: [<https://pureportal.coventry.ac.uk/en/persons/majdi-fanous>, '<https://pureportal.coventry.ac.uk/en/persons/jonathan-eden>', '<https://pureportal.coventry.ac.uk/en/persons/alireza-daneshkhah>']

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

1. 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.

2. 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:

```
# Define the URLs for the news feeds
urls = {
    "sports": ["https://feeds.bbc1.co.uk/sport/rss.xml", "https://www.espn.com/espn/rss/news"],
    "technology": ["https://feeds.bbc1.co.uk/news/technology/rss.xml", "https://www.wired.com/feed/rss"],
    "climate": ["https://feeds.bbc1.co.uk/news/science-environment-56837908/rss.xml", "https://theconversation.com/uk/topics/climate-change-27"]
}
```

"sports": ["https://feeds.bbc1.co.uk/sport/rss.xml",
"https://www.espn.com/espn/rss/news"],

"technology": ["https://feeds.bbc1.co.uk/news/technology/rss.xml",
"https://www.wired.com/feed/rss"],

"climate": ["https://feeds.bbc1.co.uk/news/science-environment-56837908/rss.xml",
"https://theconversation.com/uk/topics/climate-change-27"]

And the total number of documents collected are

```
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:]]
```

Total documents collected: 193

3. 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

```
# 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=3)
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))
```

TF-IDF Performance Metrics:
Silhouette Score: 0.010008362285493377

4. 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.

5. 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.

```
[16]: 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()
```

```
Select the following options
1. To Enter Query
2. Exit
1
Enter Query: ronaldo is a football player
Result: Sports
Select the following options
1. To Enter Query
2. Exit
2
```

```
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()
```

```
Select the following options
1. To Enter Query
2. Exit
1
Enter Query: Amazon forest catches fire
Result: Climate
Select the following options
1. To Enter Query
2. Exit
2
```

```
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()
```

```
Select the following options
1. To Enter Query
2. Exit
apple launched a new product
Enter correct option

Select the following options
1. To Enter Query
2. Exit
1
Enter Query: apple launched a new product
Result: Climate
```

```

Select the following options
1. To Enter Query
2. Exit
1
Enter Query: pollution levels are rising
Result: Sports
Select the following options
1. To Enter Query
2. Exit
1
Enter Query: climate
Result: Technology
Select the following options
1. To Enter Query
2. Exit
2

```

6. *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.

7. *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)

```

```

tfidf = TfidfVectorizer(use_idf=True, smooth_idf=True, norm=None, binary=False)
tfidf_matrix = tfidf.fit_transform(processed_pub_names)
terms = tfidf.get_feature_names()
dense_matrix = tfidf_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.bbc.co.uk/sport/rss.xml", "https://www.espn.com/espn/rss/news"],

    "technology": ["https://feeds.bbc.co.uk/news/technology/rss.xml",

"https://www.wired.com/feed/rss"],

    "climate": ["https://feeds.bbc.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/>