INFORMATION RETRIEVAL

7071CEM

COURSEWORK

1. INTRODUCTION

Search Engine

Every single day, every one of us uses a search engine. You've looked through enormous volumes of unstructured data today, whether it was to discover a gadget on Currys, a movie on Netflix, branded watches on Amazon, or anything else on the internet through Google. A search engine is a software program that performs web searches. Crawling, indexing and ranking are the three fundamental processes of a search engine.

The objective of the task is to develop a search engine that, like Google Scholar, exclusively papers and books produced at Coventry University.

Document Clustering

Clustering is the process of assembling related things. Items that are similar to one another are contained in each group, also known as a cluster. Clustering algorithms are unsupervised learning algorithms (Sanjaya's Blog, 2019). There are numerous clustering algorithms, such as Spectral Clustering, Hierarchical Clustering, DBSCAN, and KMeans. Here I used the KMeans algorithm to cluster the data.

2. METHODOLOGY

Task 1: Search Engine

Crawler

Web crawling plays a crucial role in search engines. Every well-known search engine has a web crawler, and the big ones have numerous crawlers, each with a particular focus. For instance, Google's primary spider is Googlebot. But there are also a number of other Google bots, like AdsBot, Googlebot News, Googlebot Images, and Googlebot Videos. Bing offers additional specialised bots as well as a conventional web crawler called Bingbot (Shirey, n.d.).

With a list of URLs to visit, a web crawler begins its job. The crawler locates links in the HTML for each URL, filters those links according to specified criteria, and then adds the new links to a queue.

https://pureportal.coventry.ac.uk/en/organisations/school-of-economics-finance-and-accounting is the website that our search engine crawled. More than 600 publishing pages from this website have been accessed by our web crawler. For each publication, it extracts available data (such as authors, publication year, and title) and the links to both the publication page and the author's profile (also called "pureportal" profile) page.

For crawling the data here I used a python library called Beautiful Soup. It can extract data from XML and HTML files.

After crawling the relevant information I applied data preprocessing to crawled data before passing it to the indexer. All crawled data is converted to lowercase because there might be a

chance to treat a word which is beginning with a capital letter different from the same word which appears later without any capital letter. This could cause the accuracy to drop. I tested all regular expressions in the crawled data using the python re module. Using the NLTK stopwords I removed all stopwords and tokenized the crawled data. Also, using SnowballStemmer I stemmed the entire data. In stemming, word suffixes are removed or substituted to determine a word's root form.

Indexer

An indexing process enables search engines to respond incredibly quickly to queries by organizing information prior to a search. The search engines would have a very difficult time finding useful material by going through individual pages and searching for keywords. As an alternative, search engines employ an inverted index, also referred to as a reverse index.

I chose not to use Elastic Search for my search engine and instead used Python's built-in dictionary data structure to construct the inverted index.

Inverted indexes enable quick full-text searches. It consists of a database with one entry for each word found in all the processed documents and a set of key pairs that include the document id and the frequency of the term in the document.

Fig (1.1) Inverted Index

Scheduler

A scheduling mechanism is required by the majority of applications created today. Here with the help of the python schedule, I scheduled a crawler that automatically runs the crawler every Monday at 03:00 hours because of the web page's low pace of information changes. Additionally, we can execute our Python crawler software manually.

Fig (1.2) Scheduler for crawler

Query processor

In this proposed system it is possible to execute both a single-text search and a multiple-text search. In this system, the entire query is changed to lowercase to prevent issues with case sensitivity. I then eliminated all stopwords from the search query, along with any trailing spaces, special letters, and symbols. My search engine does not support boolean queries, but it does accept keywords the way Google Scholar does.

Using a Bootstrap, CSS & HTML frontend, I retrieved the user's search query with the help of Flask and preprocessed it, such as deleting stopwords, trailing spaces, special characters, and symbols, as well as changing the text's case.

```
def filterArticle(searchQuery):
    result = []
    relevance = dict()
    searchQuery = [
        word
        for word in word_tokenize(
            re.sub("\W+"," ", searchQuery.lower().strip())
        )
        if not word in set(stopwords.words("english"))
    ]

    for word in searchQuery:
        for pubId in searchIndex.get(word, list()):
            relevance[pubId] = relevance.get(pubId,0) + 1

    relevance = sorted(relevance.items(), key = lambda rank: rank[1], reverse=True)
    result = [data[pubId] for pubId, _ in relevance]

    return result
```

Fig (1.3) searchQuery preprocessing

Then I determine which information is pertinent to the search query by looking at the rank of the publication data. I display the results of the search with the most relevance at the top and the publications with the lowest relevance at the bottom.

Task 2: Document clustering

One of the most popular topics in data science is clustering. Basically, clusters are collections of related objects. The method used to divide the objects into these groups is called clustering. In a cluster, objects should be as similar as possible. There should be as little similarity between objects in different clusters as possible.

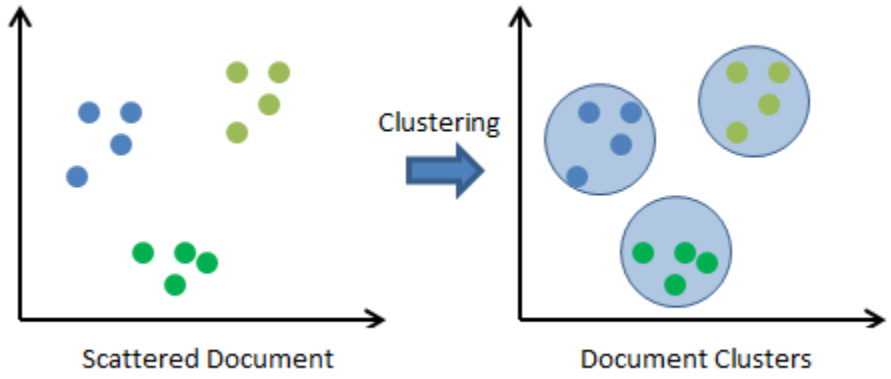

Fig (1.4) Clustering Source: Kunwar 2013

Dataset

For this system, I have selected a dataset from Kaggle that is divided into 5 categories: sports, business, technology, entertainment, and politics. I have merged another dataset containing the category health. I've applied a few of the standard preprocessing methods, including term filtering, stemming, and tokenizing.

Document Clustering

To create the document clustering system for this coursework, I used the K-means clustering technique. K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid (Pulkit Sharma, 2019).

```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
import joblib
import numpy as np
import os
import pandas as pd
```

Fig (1.5) Importing necessary library for clustering

I use the Python Pandas module to import the dataset, which is a CSV file containing all of our documents. I have two datasets one containing categories sports, politics, entertainment, business, and technology and dataset2 containing category health so I merged both datasets using Pandas module merge. Using beautifulsoup, I crawled the BBC news website to find health data. In total, I have 1523 datasets.

```
dataset = pd.read_csv(os.path.join("BBC News2.csv"))
dataset2 = pd.read_csv(os.path.join("HealthBBC.csv"))
output_dataset = pd.merge(dataset, dataset2,
                          how='outer')
print("Total data in dataset: ",len(output_dataset))
```

Fig (1.6) Merging two datasets

I removed all stopwords from the dataset. I employed the TF-IDF vectoriser to fit the algorithm for prediction. K-means will be used to cluster the documents using this document-term matrix.

```
vector = TfidfVectorizer(sublinear_tf=True,min_df=5, ngram_range=(1,2), stop_words="english")
features = vector.fit_transform(output_dataset.Text).toarray()
```

Fig (1.7) converting document as vector

Next, we cluster our documents using the K-means clustering algorithm that was imported from the Sklearn package.

```
model = KMeans(n_clusters=6).fit(features)
output_dataset["Cluster"] = model.labels_
```

Fig (1.8) K-mean clustering algorithm

```
while contRunning == "y":
  docu = input("Please enter the document:\n").lower()
  docu = np.array([docu])
  docu = tf.transform(docu)
  cluster = model.predict(docu)
  print(f"Predicted cluster: {Category[cluster[0]]}")

  contRunning = input("Do you want to continue Y/N ? ").lower()
```

Fig (1.9) Predicting new document

```
(myenv) Alans-MBP:Task_2 aleenaalby$ python3.10 clustering.py
Total data in dataset:  1523
Saving model at: Trained_model.pkl
Saving vectorizer at: vector.pkl
(myenv) Alans-MBP:Task_2 aleenaalby$
```

Fig (1.10) Output of clustering.py

3. RESULTS

Task 1: Search Engine

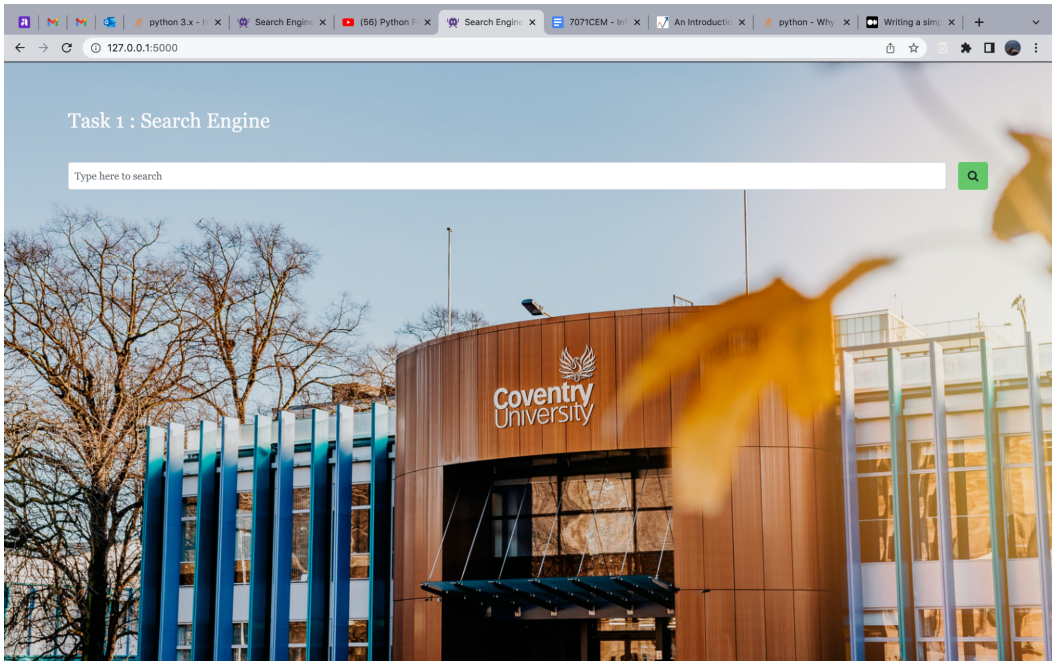

Fig (3.1) Front-end of search engine [Background image-source flickr.com]

The above figure is the front end of the search engine, where users can type their queries. The front-end was developed using HTML, CSS and bootstrap. Python-based Flask, which is implemented on Werkzeug and Jinja2, is used here to create this web application.

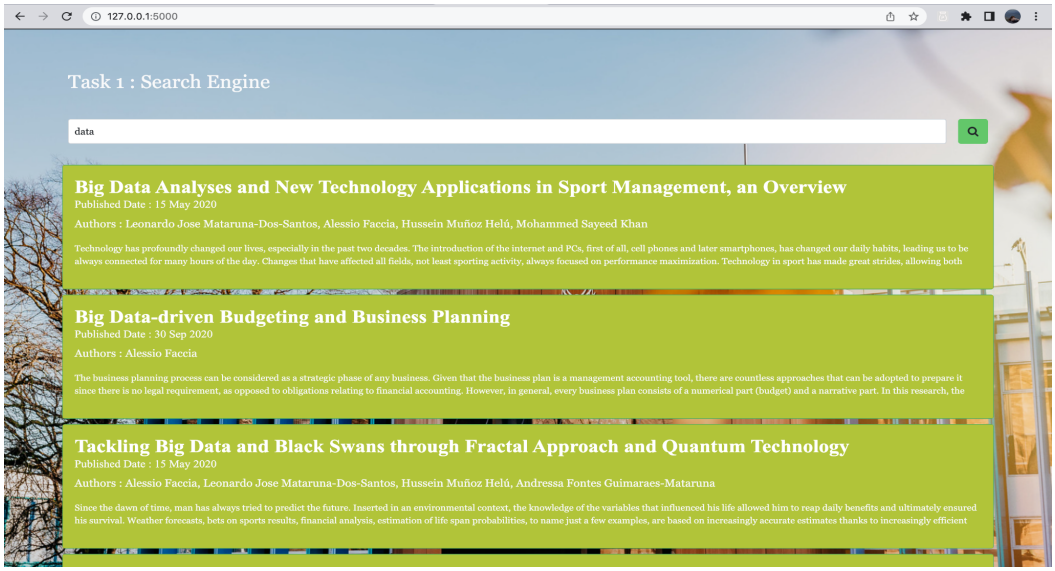

Fig (3.2) Result of searching one keyword

Figure 3.2 shows the result of searching a single keyword and it displayed publications containing that term. The result contains the publication name and link to it, published date, name of authors and abstract of publications with the scroll down option.

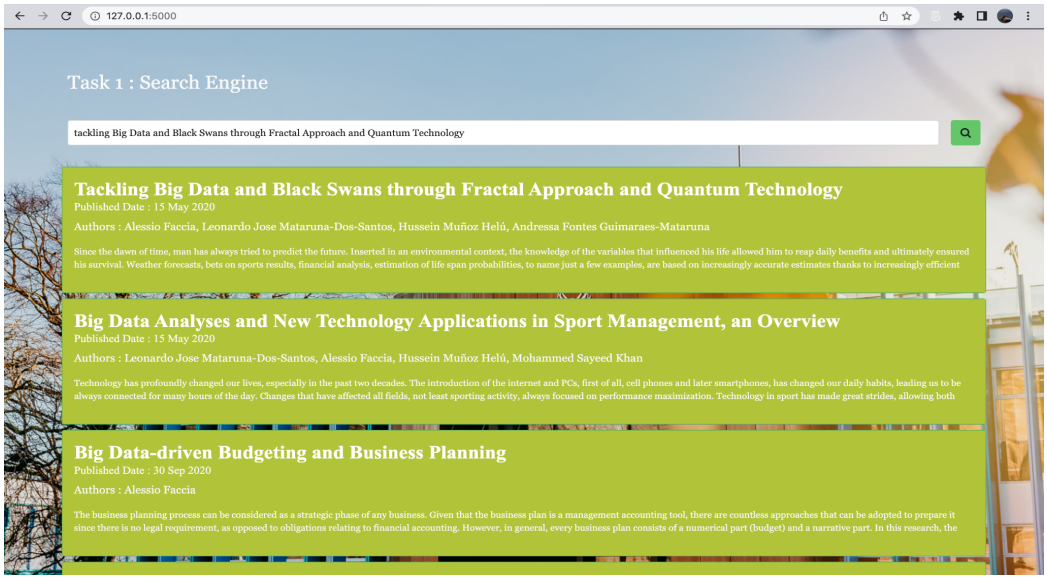

Fig (3.3) Result of searching multiple keywords with stopwords

In figure 3.3 we can see multiple keywords are searched with stop words. It displayed the outcomes of articles that contain those search phrases. The first result is the same as the search term. Then the publication data with more relevance to the search term is displayed.

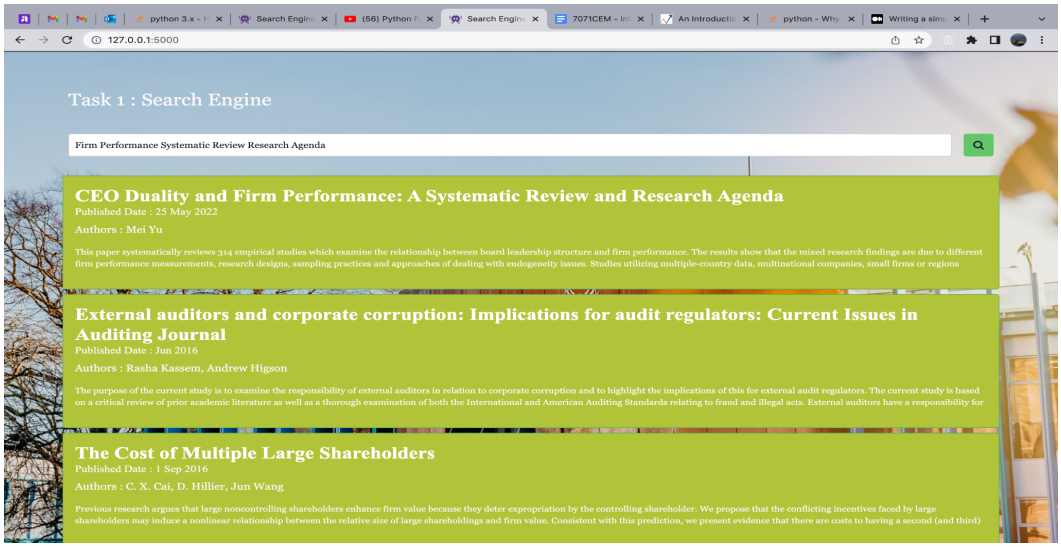

Fig (3.4) Result of searching multiple keywords without stopwords

In figure 3.4 we can see multiple keywords are searched without stop words. It displayed the outcomes of articles that contain those search phrases.

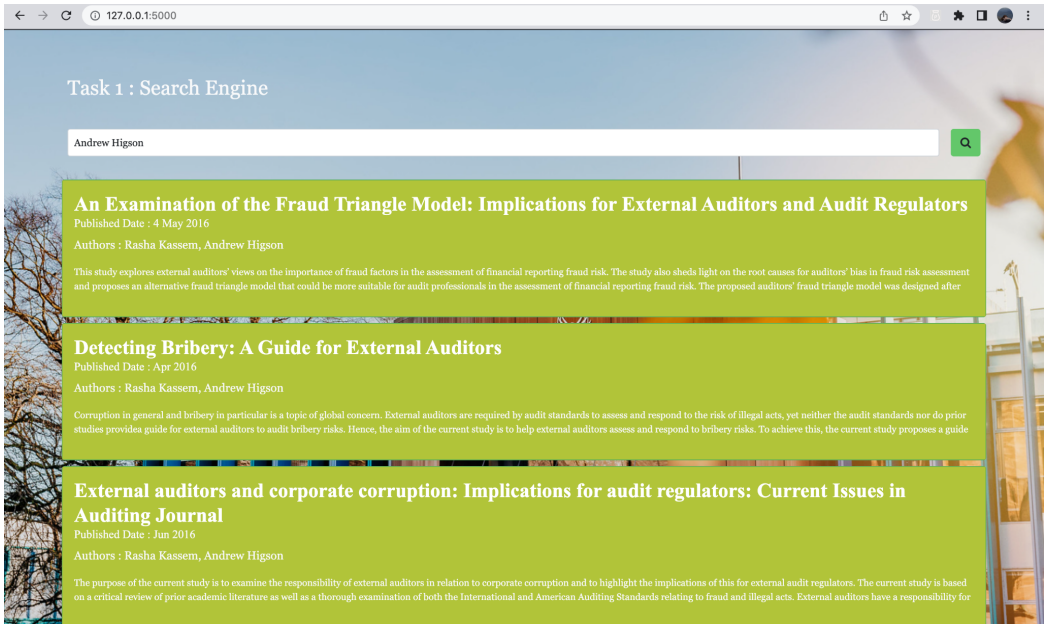

Fig (3.5) Result of searching with author-name

The above figure shows the result of searching the author name ‘Andrew Higson’. It displayed all publications which include keywords ‘Andrew’ and ‘Higson’.

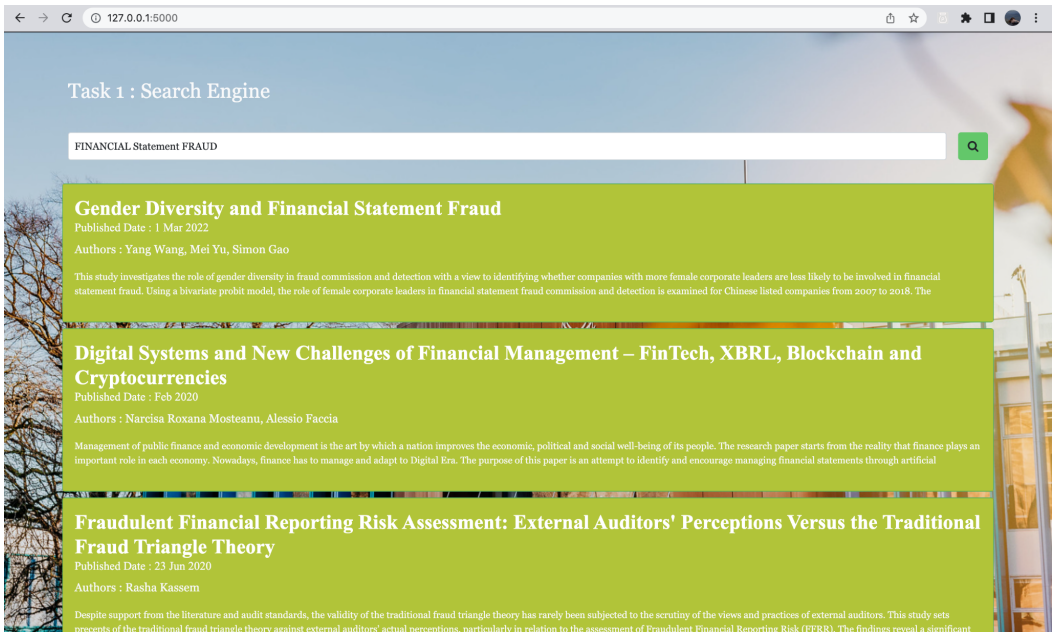

Fig (3.6) Result of searching keywords both with uppercase and lowercase

In Figure 3.6, several keywords, both capital and lowercase, are entered into the search engine, and the results of publications containing those phrases are displayed without regard to the case.

Task 2: Document clustering

As part of this project, I input a variety of documents with simple and complex sentences, with and without stop words, as well as some challenging queries in order to test our method's accuracy and resiliency.

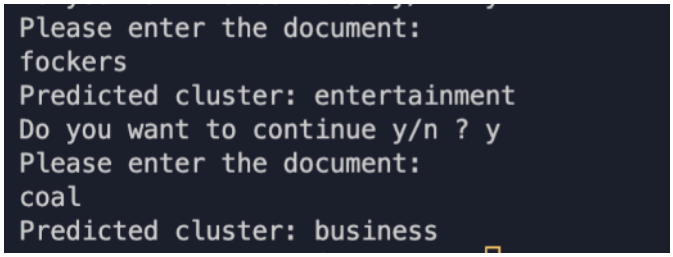

Fig (3.7) With short word

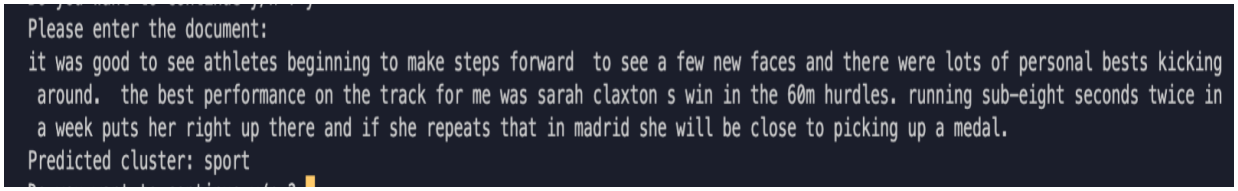

Fig (3.8) With lengthy sentence

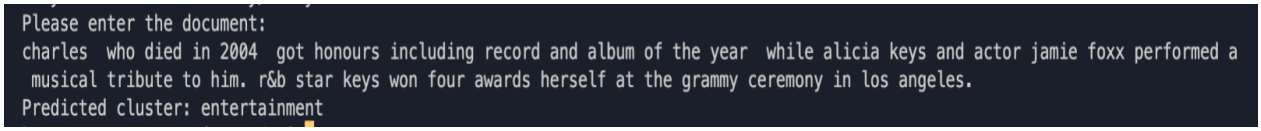

Fig (3.9) lengthy sentence with stop words

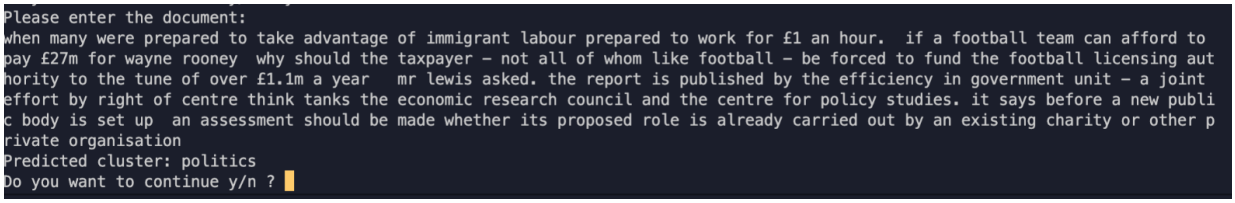

Fig (3.10) Document clustered into Politics

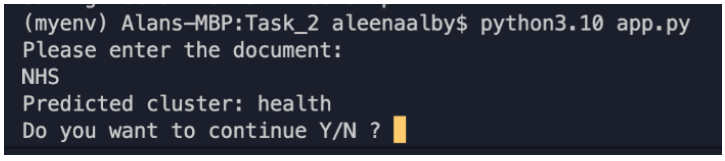

Fig (3.11) Document clustered into health

4. CONCLUSION

Task 1: Search Engine

A fully functional search engine system that only returns papers and books written by members of the School of Economics, Finance, and Accounting was successfully developed by using python. The system has the ability to breadth-first search (BFS) all available publications by crawling the pertinent web pages. We further tested the reliability of our search engine by feeding it a variety of search terms. Our system then presented all the pertinent publication data, including each article's title, authors, publication date, and abstract description.

Task 2: Document clustering

I have created a successful document clustering system that can assign any given document to the appropriate cluster. Our technology is capable of accurately predicting which document belongs to which cluster. By passing many kinds of documents with short and long sentences, as well as with and without stop words, I have demonstrated its robustness and accuracy and predicted the appropriate cluster for those papers.

5. REFERENCE

Bajo, A. (2020, December 11). *Web crawling with Python*. ScrapingBee. Retrieved July 21, 2022, from https://www.scrapingbee.com/blog/crawling-python/

Coventry University | Coventry City Council. (2019, February 28). Flickr. Retrieved July 24, 2022, from https://www.flickr.com/photos/coventrycc/46512799284

Esteban. (2018, October 22). Writing a simple Inverted Index in Python | by Esteban. Medium. Retrieved July 24, 2022, from https://medium.com/@fro_g/writing-a-simple-inverted-index-in-python-3c8bcb52169a

Koch, K. (2020, March 25). A Friendly Introduction to Text Clustering | by Korbinian Koch. Towards Data Science. Retrieved August 1, 2022, from https://towardsdatascience.com/a-friendly-introduction-to-text-clustering-fa996bcefd04

Python | Schedule Library. (2022, April 11). GeeksforGeeks. Retrieved July 24, 2022, from https://www.geeksforgeeks.org/python-schedule-library/

Shirey, T. (n.d.). Web Crawler 101: What Is a Web Crawler? (And How It Works). WebFX. Retrieved July 21, 2022, from https://www.webfx.com/blog/internet/what-is-a-web-crawler/

Sharma, P. (2022). K Means Clustering | K Means Clustering Algorithm in Python. Retrieved 1 August 2022, from https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

(n.d.). Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation. Retrieved July 21, 2022, from https://beautiful-soup-4.readthedocs.io/en/latest/

Subedi, S. (2022). NLP with Python: Text Clustering. Retrieved 2 August 2022, from https://sanjayasubedi.com.np/nlp/nlp-with-python-document-clustering/

6. APPENDIX

Task 1: Search Engine

crawler.py

```
import requests
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from time import sleep
import pandas as pd
import re
import os
import uuid
import json
from nltk.stem import SnowballStemmer

def run_crawler():
   df = pd.DataFrame({'title': [''],
'author_name':[''],'title_link':[''],'date':['']})
   page_nums = soup.find("nav", class_="pages").find_all("li")
   length_of_pages = len(page_nums)
   final_data = dict()
   page_nums = 0
   while page_nums != length_of_pages + 1:
        URL =
"https://pureportal.coventry.ac.uk/en/organisations/school-of-economics-finance-and-ac
counting/publications/?page={}".format(page_nums)
       #  print(URL)
        page = requests.get(URL)
        soup = BeautifulSoup(page.content, "html.parser")
        results = soup.find("ul", class_="list-results")
        job_elements = results.find_all("div", class_="result-container")

        for job_element in job_elements:
           # print(job_element, end="\n" *2)
           title_name = job_element.find("h3", class_="title" ).find("a", href =
True).find("span").text
           title_link = job_element.find("h3", class_="title" ).find("a", href =
True).get("href")
           date_element = job_element.find("span", class_="date" ).text

           URL_title = title_link
```

```
            page_title = requests.get(URL_title)
            soup_1 = BeautifulSoup(page_title.content, "html.parser")
            author_name = soup_1.find("p", class_="relations persons").text
            data_desc = soup_1.find("div", class_="textblock").text if
soup_1.find("div",class_="textblock") else ""

            print(data_desc)

            filtered_data = title_name + ' ' + author_name + ' '+ data_desc
            filtered_title = re.sub("\W+", " ", filtered_data ).lower()
            print(filtered_title , "filtered_title ***********")

            filtered_title = [ word for word in word_tokenize( filtered_title ) if not
word in set(stopwords.words("english"))]
            snowball = SnowballStemmer(language = 'english')

            filtered_title_ =[]
            for word in filtered_title:
                x = snowball.stem(word)
                filtered_title_ .append(x)
            print( filtered_title_ , "filtered_title$$$$$")


            uuidValue = str(uuid.uuid4())
            data = {'title': title_name,
'author_name':author_name,'title_link':title_link,'date':date_element,'data_desc':data
_desc,'filtered_title':filtered_title_}


            final_data[uuidValue] = data
            # print("**********",title_name)
            # print("author_name",author_name)
            # print("**********",title_link)
            # print("#####",date_element)
            # break
        page_nums = page_nums + 1
       #  break

   writeDataToDisk(final_data)


def writeDataToDisk(data):
```

```
    with open(os.path.join("data.json"), "w") as write_file:
        json.dump(data, write_file)

if __name__ == "__main__":
    run_crawler()
```

Indexer.py

```
from genericpath import isfile
import os
import json

def InvertedIndex():
    publications = None
    if os.path.isfile(os.path.join("data.json")):
        with open(os.path.join("data.json"),"r") as jsonFile:
            publications = json.load(jsonFile)
    else:
        return publications

    invertedIndex = dict()
    for id_, article in publications.items():
        for key in article["filtered_title"]:
            if key not in invertedIndex:
                invertedIndex[key] = dict()
            invertedIndex[key][id_] = True
    return invertedIndex
```

Schedular.py

```
import schedule
import time
import datetime
import crawler

def crawler_job():
    date = datetime.datetime.now()
    print(date)
                    print(f"Running       at:       {date.day}/{date.month}/{date.year}
({date.hour}:{date.minute})\n")
    crawler.run_crawler()
```

```
schedule.every().monday.at("03:00").do(crawler_job)
# schedule.every(10).seconds.do(crawler_job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

app.py

```
from flask import Flask, render_template, request
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from indexer import InvertedIndex
import re
import os
import json

app = Flask(__name__)
searchIndex = None
data = None

#IndexPage
@app.route('/', methods=["POST", "GET"])
def index():
    searchQuery = ""
    article = None
    if request.method == "POST":
        searchQuery = request.form['searchWord']
        article = filterArticle(searchQuery)
    return render_template('index.html', articles_ = article, searchWord = searchQuery
)

def loadData():
    with open(os.path.join("data.json"), "r") as read_file:
        data = json.load(read_file)
    invertedIndex = InvertedIndex()
    return data, invertedIndex


def filterArticle(searchQuery):
    result = []
    relevance = dict()
```

```
    searchQuery = [
        word
        for word in word_tokenize(
            re.sub("\W+"," ", searchQuery.lower().strip())
        )
        if not word in set(stopwords.words("english"))
    ]

    for word in searchQuery:
        for pubId in searchIndex.get(word, list()):
            relevance[pubId] = relevance.get(pubId,0) + 1

    relevance = sorted(relevance.items(), key = lambda rank: rank[1], reverse=True)
    result = [data[pubId] for pubId, _ in relevance]

    return result

if __name__ == '__main__':
    data, searchIndex = loadData()
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Search Engine</title>
    <link rel="icon" type="image/x-icon" href="/static/img.png" />
                                                <link                rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.c
ss">
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
      rel="stylesheet"

integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
      crossorigin="anonymous"
    />
    <style>
```

```
    .bg-back{
      /* background-color: #a598ee; */
      background-image: url(/static/Screenshot\ 2022-07-24\ at\ 5.25.53\ PM.png);
      background-size: 100%;
      /* background-repeat: no-repeat; */
      font-family: Georgia;
    }

    .btn1{
      background-color: #63c76a;
    }
    .article_class{
      background-color: #63c76a;
    }
    .data-description{
      line-height: 1.5em;
      height: 3em;
      overflow: scroll;
    }
  </style>
</head>
<body class="bg-back" >
  <div class="container text-light">
    <h3 class="mt-5 p-3">Task 1 : Search Engine</h3>
    <form class="d-flex mt-2 p-3" action="/" method="POST">
      <input
        class="form-control form-control-sm me-3"
        name="searchWord"
        type="search"
        value="{{ searchWord }}"
        placeholder="Type here to search"
        aria-label="Search"
      />
      <button class="btn btn1" type="submit"><i class="fa fa-search"></i></button>
    </form>

    <div class="mt-3 mb-3"></div>
    {% if articles_ %} {% for article in articles_ %}
    <div class="card m-2 article_class">
      <div class="card-body bg-warning" style="--bs-bg-opacity: 0.5">
        <h5
          class="card-title fw-bolder fs-3"
```

```
          style="font-family: 'Times New Roman', Times, serif"
        >
          <a
            href="{{ article.title_link }}"
            class="text-white"
            style="text-decoration: none"
          >
            {{ article.title }}
          </a>
          <p class="h6"> Published Date : {{ article.date }}</p>
        </h5>
        <p>
          Authors : {{ article.author_name}}
        </p>
        <p class="data-description" style="font-size: small">
          {{ article.data_desc }}
        </p>
      </div>
    </div>

    {% endfor %} {% endif %}
  </div>

  <script

src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"
 https://cdnjs.cloudflare.com/ajax/libs/fontawesome-iconpicker/3.1.0/js/fontawesome-i
  ></script>
 </body>
</html>
```

Task 2: Document clustering

Clustering.py

```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import os
import pandas as pd

np.random.seed(0)

dataset = pd.read_csv(os.path.join("BBC News2.csv"))
dataset2 = pd.read_csv(os.path.join("HealthBBC.csv"))
output_dataset = pd.merge(dataset, dataset2,
                  how='outer')
print("Total data in dataset: ",len(output_dataset))

vector      =      TfidfVectorizer(sublinear_tf=True,min_df=5,      ngram_range=(1,2),
stop_words="english")
features = vector.fit_transform(output_dataset.Text).toarray()

model = KMeans(n_clusters=6).fit(features)
output_dataset["Cluster"] = model.labels_

Category = {}

for i in output_dataset.groupby("Category"):
 Category[i[1]["Cluster"].value_counts().idxmax()] = i[0].strip()


mName = os.path.join("Trained_model.pkl")
print(f"Saving model at: {mName}")
joblib.dump(model, mName)

vName = os.path.join("vector.pkl")
print(f"Saving vectorizer at: {vName}")
joblib.dump(vector, vName)

joblib.dump(Category, os.path.join("Category.pkl"))
```

crawling2.py

```
import requests
import pandas as pd
from bs4 import BeautifulSoup

response = requests.get('https://www.bbc.co.uk/news/health/rss.xml?edition=uk%27#')
web = BeautifulSoup(response.text, 'html.parser')
df = pd.DataFrame({'Text': [''],'Category':['']})
headlines_ = web.find_all('description')
i = 0
for headline in headlines_:
   txt = headline.get_text()
   df = df.append({'Text': txt,'Category':"health" },ignore_index=True)
df.to_csv("health23.csv", index=False)
```

app.py

```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import os
import pandas as pd

mName = os.path.join("Trained_model.pkl")
vName = os.path.join("vector.pkl")
CategoryName = os.path.join("Category.pkl")

model = joblib.load(mName)
tf = joblib.load(vName)
Category = joblib.load(CategoryName)

contRunning = "y"

while contRunning == "y":
 docu = input("Please enter the document:\n").lower()
 docu = np.array([docu])
 docu = tf.transform(docu)
 cluster = model.predict(docu)
 print(f"Predicted cluster: {Category[cluster[0]]}")

 contRunning = input("Do you want to continue Y/N ? ").lower()
```