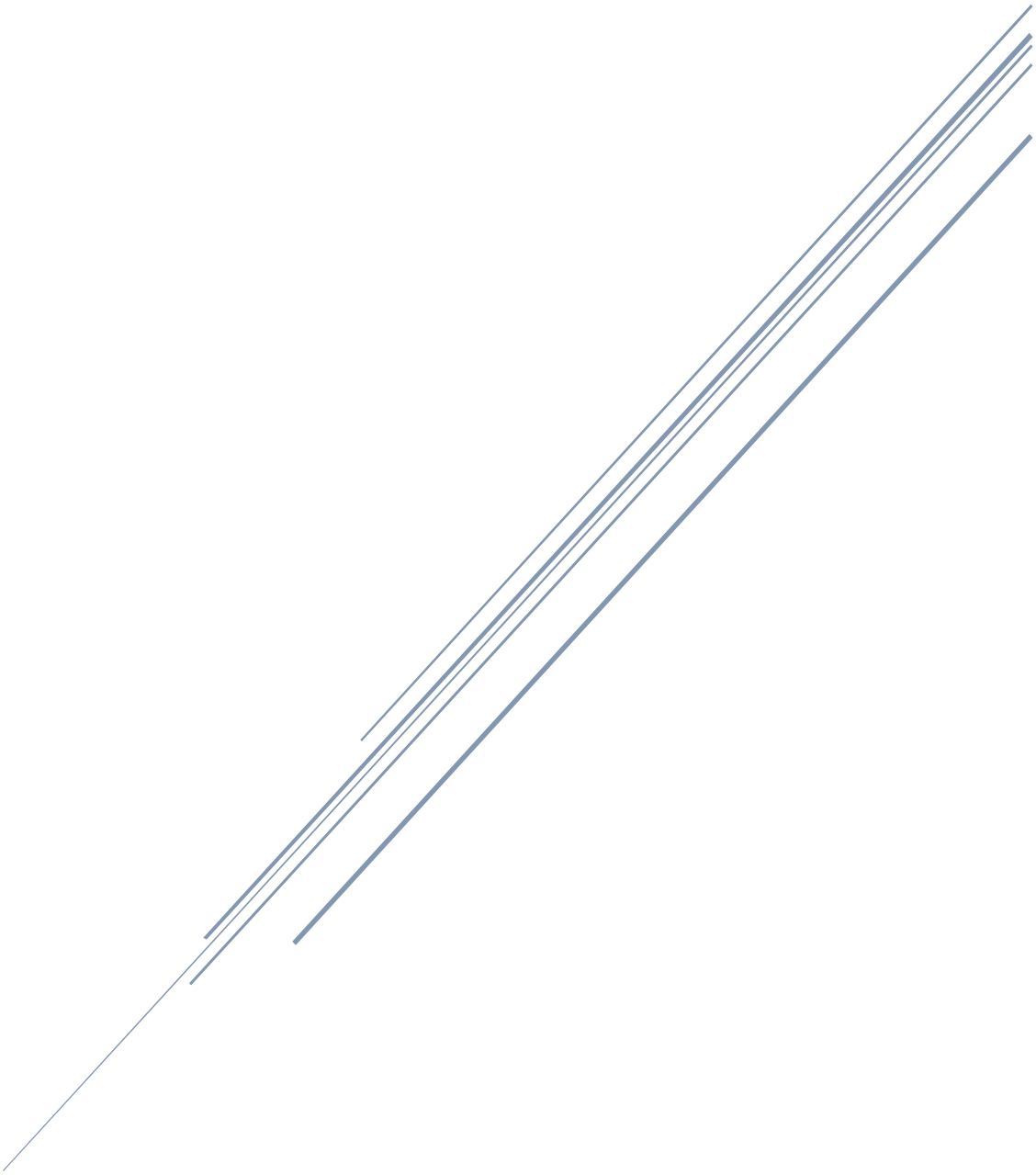
STW7071CEM INFORMATION RETRIEVAL

Assignment Report



Softwarica College
Submitted By: Manash Bhele

Contents

Abstract

With the increasing number of online technological devices and consequently, the increase in the volume of data being produced every single second, search engines are a book to humankind. An article which was published in Forbes in 2018 stated that there are 2.5 quintillion bytes of data created each day, around 5 billion searches a day with 77% of those searches conducted on Google (Marr, 2018). A proper search engine is one which provides relevant information for a user's query within an acceptable time limit. This paper provides an overview of the workings of a search engine. A vertical search engine has been created that returns papers published by Coventry University members. The pureportal website of CU was crawled and the profile information and paper specific data of the members was captured. That information was used to create means for finding the similarity between user's query and the indexed documents. Given a scientific text as an input, subject classification function has been added. For the search engine, a web interface has been added which provides a similar feel to a traditional search engine. A standalone text classification feature is also present, making use of text classification algorithms such as Random Forest, SVM and LBGM, which takes in a description of any scientific document or article and predicts if it belongs to engineering, business, or arts field.

Keywords: Crawler, Inverted index, TF-IDF, Machine learning, vertical search, cosine similarity, text classification, retrieval, web-interface

Introduction

General search engines are efficient at providing multitude of results for a query made by a user. These kinds of engines, though powerful and capable of providing a vast range of information, might not be able to provide the result that is desired by and relevant to the user's needs. With such a plethora of info a user faces, rather than being of help, it might act as an obstacle for the user. To address this problem, a specific kind of search engine called vertical search engine has been made.

A vertical search engine focuses only on a particular category or segment of online content. Some common examples of verticals include shopping, the automobile industry, research literature, homestay platforms. Unlike general search engines, vertical engines use crawlers that only focus on a specific niche or topic (Wikipedia, 2023).

This report demonstrates the working principles of information retrieval processes and the use of machine learning algorithms in text classification. This report also explores the steps, tools and techniques used for creating a search engine and its components. It also provides an insight on the workings of the process of crawling a website for gathering relevant and useful information. The techniques and their implementation are discussed and explained in the following sections.

Methodology

The primary element indispensable in building a search engine is data. Data relevant for building a vertical search engine must be selected from sources that align with the niche. For this project, the data has been collected from the pure-portal website of Coventry University. The main goal of this project is to crawl the website and gather data on the profiles of the CU academics and their respective published articles to create a vertical search engine for those research papers. The explanations below also describe the text classification system which takes keywords or text as input and identifies its relevant subjects.

The main components and workings of the developed system are described below.

1. Web Crawling using Scrapy

The crawling component was accomplished using an open-source web crawling framework called 'Scrapy.' Scrapy is a tool used to crawl websites and extract structured data from their pages.

As it is not pre-installed in systems, it was downloaded using the pip installer by executing the command 'pip install scrapy.' To make the process of crawling unbounded by hit limitations and straight forward, instead of using Google Scholar, the CU pure-portal website was used. For starting crawling in Scrapy, a 'start_urls' value should be initiated. 'start_urls,' as the name suggests, is the starting point of the crawling process. In this case, it was set as, "https://pureportal.coventry.ac.uk/en/persons."

Another important variable to set is "allowed_domains." A website might have links of various other websites present on a single page. For example, when crawling a certain news site, there might be a link re-directing to social media sites. This might lead the crawler to leave the site to be crawled and start crawling the entire internet. To stop it from doing so, the links "pureportal.coventry.ac.uk" and "pureportal.coventry.ac.uk/en/persons/" were added to allowed domains. It prevents the crawler from crawling beyond these domains.

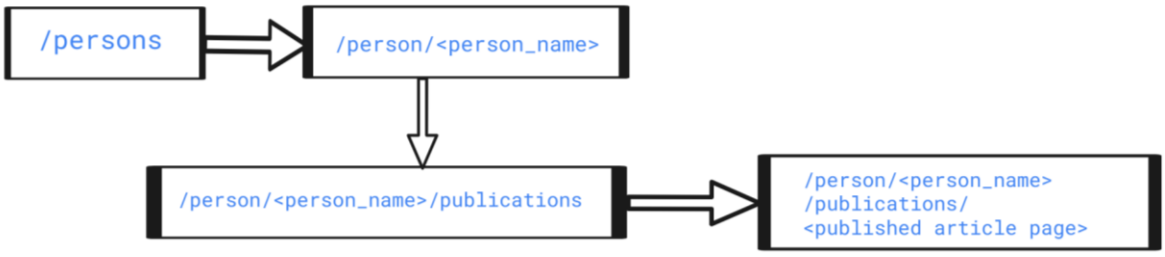


Figure 1 Phases of crawling the website

The figure above shows the flow of the crawling process. The starting page being '/persons,' first the crawler finds all the persons on the first page, using the 'response.css' method. For extracting the persons from the page, '*response.css('div.result-container')*' was used.

Then for a single person on that page, the crawler gets the link for the person's profile. Then by adding '/publications' to the person's url, the page where the individual's all publications are present is reached. It was done by using the *'response.follow(publication_url, callback = self.parse_person_pubs)'* method. The 'publication_url' referes to the person's publications page which is used as the response for the function 'parse_person_pubs' being called using callback. The 'parse_person_pubs' function gathers links for all the publications present in the page and further uses the *'follow ()'* method and follows every document's links and calls the *'get_abstract( )'* function. The *'get_abstract()'* gets the document's abstract section and lastly yields the person's details and the person's publications including its title and abstract.

The following list of steps sums up all the processes:

1. From the parse function, visit the starting url and gather attributes of all persons using the response.css method.
2. For an individuvial person, extract their publication url and call the parse_person_pubs function with the publication url as a response parameter.
3. From parse_person_pubs function, using the publication url passed from previous step, find all the published articles for that individual using the css response and extract their urls.
4. For a url, using follow method, call the get_abstract method with the published article's url as response.
5. In the get_abstract method, using the passed url yields all the information of the author and the title and abstract of the published article.

6. Go to step 2 and follow the steps for all people on that page. After that, the next_page value gets initialized and using the follow method, the parse function gets recalled and the steps from 1 to 6 gets repeated unless all the pages are crawled.

When the crawling process is executed and completed, it provides a '.csv' file as an output.

The steps above generalize the process of the crawling that was implemented in the code. In addition to crawling the website, the functionality of scheduling the crawler to crawl website automatically has also been added.

With the use of 'CrawlerProcess' method from scrapy and 'TwistedScheduler' method from 'apscheduler' was used to implement the automatic re-crawl scheduling. As the website being crawled has a low rate of being updated every day, the crawler has been scheduled to re-crawl every 14 days (about 2 weeks).

```
class UnispiderSpider(scrapy.Spider):
    name = "unispider"
    allowed_domains = [ "pureportal.coventry.ac.uk", "pureportal.coventry.ac.uk/en/persons/"]
    #so that the scarper doesn't scrape the entire internet
    start_urls = ["https://pureportal.coventry.ac.uk/en/persons"]

    def parse(self, response):
        persons = response.css('div.result-container')
        for person in persons:
            if person.css('ul li span.minor.dimmed::text').get() =! None:
                    person_url =  person.css('h3 a').attrib['href']
                    publication_url = person_url + "/publications/"
                    yield response.follow(publication_url, callback = self.parse_person_pubs)

        next_page = response.css("li.next a ::attr(href)").get()
        if  next_page is not None:
            next_page_url = "https://pureportal.coventry.ac.uk" + next_page
            yield response.follow(next_page_url, callback = self.parse)

    def parse_person_pubs(self, response):
        publications = response.css("div.result-container h3.title")
        for publication in publications:
            doc_link = publication.css("a ::attr('href')").get()

            yield response.follow(doc_link, callback=self.get_abstract,
                                  meta={'publication': publication, 'rp' : response})

    def get_abstract(self, response):
        doc_abstract = response.xpath('//*[@id="main-content"]/section[1]/div/div/div[1]/div[1]/div/text()').get()
        publication = response.meta['publication']
        rp = response.meta['rp']

        yield {
            "name": rp.css('div.header.person-details h1::text').get(),
            "job_title": rp.css('span.job-title::text').get(),
            "organization": rp.css('a.link.primary span::text').get(),
            "doc_link": publication.css("a::attr(href)").get(),
            "doc_title": publication.css("span::text").get(),
            "doc_abstract": doc_abstract,
        }
```

Figure 2 Code of the Crawler

2. Construction of the Query Processor Component

The data gathered by the crawler is the main ingredient for creating the search engine. Data crawled from websites is raw and not suitable for direct use to create any kind of models.

After the completion of the crawling process, the gathered raw csv data was imported into the program for the step of preprocessing. The following steps describe the flow of the text cleaning and prepping process.

a. Cleaning Missing and Irrelevant Values

The raw data had some missing values as well as values that would not add much towards the dataset itself. Using the 'pandas' library, the dataset was loaded into a dataframe. The null values were dropped using the 'dropna' method and other rows which did not have relevant, and enough information were also dropped.

b. Removing Stop Words and Special Characters

Stop words are highly occuring words such as *'a,' 'an,' 'the,' 'is,'* etc. Such kinds of words can be found in all kinds of texts. When creating any kind of text classification models or invreted indexes or any kind of text related tasks, they donot provide any meaningful information. That is why stop words should be removed from datasets. Also, removing stop words reduces the size of dataset to an extent which helps in faster computations. (wisdomml, 2022)

A library called 'nltk' was used for this purpose. From 'nltk,' stopwords were downloaded. Then for a document from the dataframe, the words were splitted. The words were then transformend to lower case and then checked if it was a stopword. Finally, the punctuation was also removed from the documents.

```
def clean_docs(i, doc):
    stops = stopwords.words('english')
    words = doc.split()

    final = []

    for word in words:
        word = word.lower()
        word = word.replace('-', ' ')
        if word not in stops:
            final.append(word)

    final = " ". join(final)
    final = final.translate(str.maketrans("", "", string.punctuation))
    df_final.loc[i, 'doc_content'] = final
```

Figure 3 Code Snippet of the function used to clean documents

c. Calcuating TF-IDF and Cosine Similarity

TF-IDF stands for term frequency-inverse document frequency. When creating search engines, it helps to determine the importance of string representations in a document amongst a collection of documents (Simha, 2021).

TF or Term Frequecy works by examining the occurrence rate of a specific term of interest compared to the entire document. ID or Inverse document frequency examines the prevalence or rarity of a word within a corpus of documents.

$$TF_{(w,j)} = \frac{\text{(Number of times term w appears in a document)}}{\text{(Total number of terms w in the document)}}$$

Figure 4 Formula for calculating TF

$$IDF(t) = \log\frac{1+n}{1+df(t)} + 1$$

Figure 5 Formula for calculating TDF (Sivarajah, 2020)

"Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in the same direction. It is often used to measure document similarity in text analysis." (Jiawei Han, 2012) Cosine similarity helps in determining how similar two data entities are regardless of their scale or size. When calculating cosine similarity, the data objects from a dataset are treated as vectors. The cosine similarity between two vectors can be calcuated by using the formula as shown in the figure.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2 + \cdots + a_n^2}$$

$$\|\vec{b}\| = \sqrt{b_1^2 + b_2^2 + b_3^2 + \cdots + b_n^2}$$

Figure 6 Cosine Similarity (Karbhari, 2020)

The cosine similarity calculation provides value in range from 0 to 1, which can be inversed resulting in an angular value or 'θ'. So,

- If θ = 0°, the angle between vectors a and b is zero, which means that they are similar.
- If θ = 90°, the angle between vectors a and b has increased, meaning those vectors are dissimilar.

TF-IDF and cosine similarity was implemented in the program using the 'sklearn' library. 'Scikit-learn' or 'sklearn' is a free machine learning library for Python programming language with a vast collection of machine learning tools and techniques.

After completing the process of cleaning, using 'sklearn' the 'TfidfVectorizer' and 'cosine_similarity' method was imported. An object of the 'TfidfVectorizer' was created and the documents were fitted and transformed to vectors. The output of the vectorizer was stored in a matrix, tfidf_matrix.

A query processor function was created like the text pre-processor for cleaning the query provided by the user in the web interface. The query processor was used to remove any kind of punctuation, stop words and special characters from the user's query. The user's processed and cleaned query and the tfidf matrix both were passed to the cosine_similarity function and the cosine similarity was calculated.

Then the web interface was used to present the documents that were relevant to the user's query. A simple web interface mimicing the base layout of traditional search engines was implemented using Python Flask framework and the web interface was designed using simple HTML, CSS, and bootstrap.

3. Subject Classification Using Different Machine Learning Algorithms

Text/subject classification is a technique used in machine learning that can be used to predict a category of text unseen by the model and assign pre-defined categories to it.

For this project, three different classification algorithms,

- Random Forest Classifier,
- Support Vector Machine (SVM) and
- Light Gradient Boosting Machine (LGBM) were used.

Random Forest Classifier is an ensemble classifier. It fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting (scikit-learn, n.d.). Support Vector Machine (SVM) is a machine learning algorithm used for classification, which works by finding a hyperplane in N-dimensional space that classifies the data points.

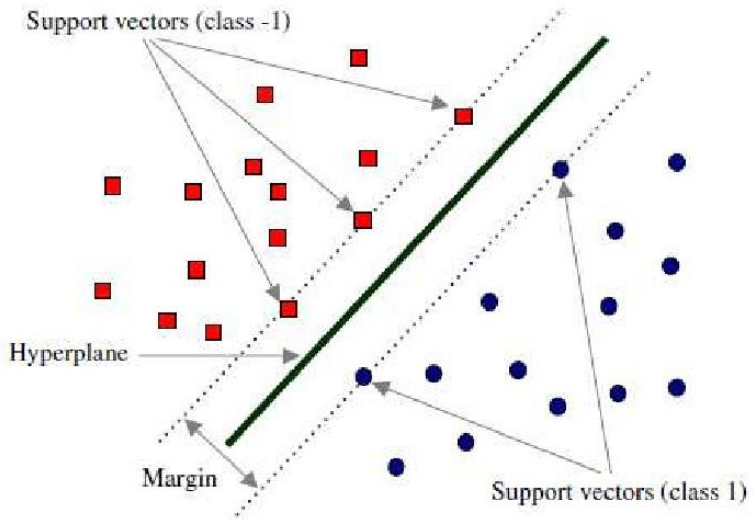


Figure 7 Support Vector Machine (Baghaee, 2020)

Light GBM is a gradient boosting framework which is based on decision trees which is very efficient, fast, provides better accuracy and can handle large-scale data. (Corporation, n.d.)

All these algorithms were implemented by importing them from the scikit learn library. First, the dataset for training the models was downloaded from the web. For that purpose, the BBC news dataset was downloaded. The dataset had two columns, one with the text and another column had the respective category for that text. The three categories present in the dataset were *engineering*, *business,* and *arts*. Like previous steps, the database text was cleaned by removing stop words, punctuation and special characters, and the text was lemmatized.

After cleaning the document texts, the text data was splitted into train and test sets using the scikit learn's train_test_split method. Then the training datas were fitted into the algorithms. Another metric called k-fold cross validation was also used to where k means the number of folds the provided data will be subjected to. It works by splitting the data into the specified number of folds, training the model on a subset of the data, and evaluating its performance on the remaining fold. This process is repeated for each fold, and the average scores for each metric are returned as the output. Scoring metrics like accuracy and weighted F1 score were also calculated.

The output and results are dicussed in the next section.

Results and Discussion

The crawler component of the project after completing the execution produces a '.csv' file with the following columns.

- Name (name)
- Job title (job_title)
- Organization (organization)
- Document URL (doc_link)
- Document Title (doc_title)
- Document Abstract (doc_abstract)

The screenshot of the file outputed by the crawler is added in the appendix. As the crawler was implemented such that it crawls the whole pureportal persons section, it can take quite a while to complete and extract all the data from the website.

The file was used as dataset for the Search engine aspect of the project. Upon executing all the processes as discussed in previous sections, the result was a web platform with a search box to type the query and a search button to execute the search function. The image below shows the resulting look of the web platform.

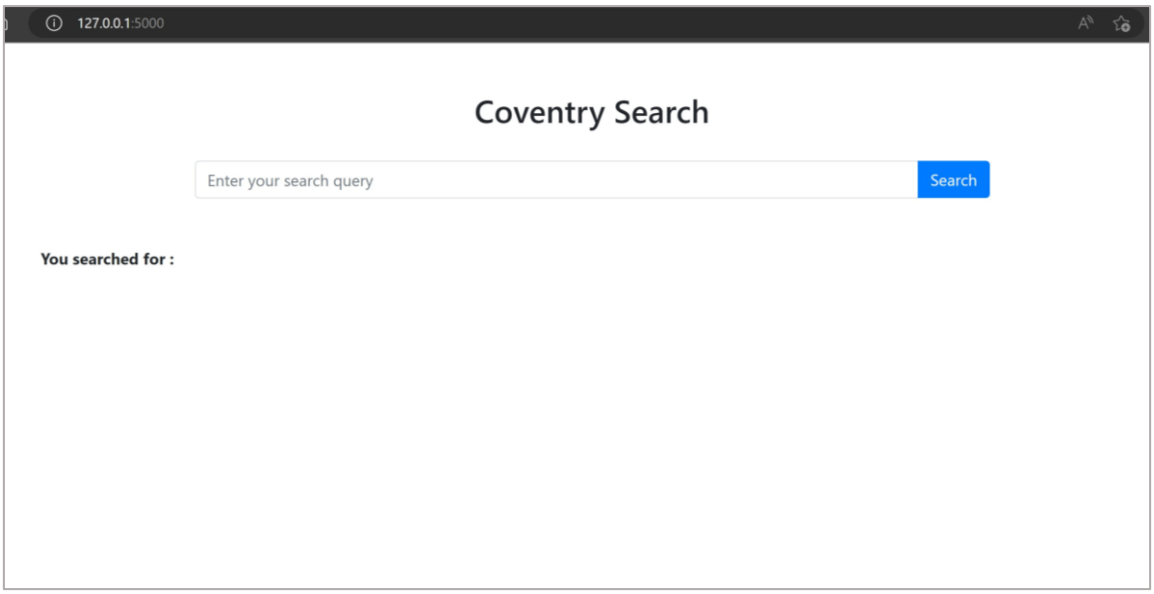


Figure 8 Search Engine UI

Upon typing the query and pressing the search button, it took around 6-8 seconds to execute and provide the output relevant to the user's query. The output contains all necessary

informations as well as the cosine similarity. The output is presented in a list like readable layout as shown in the figure below.

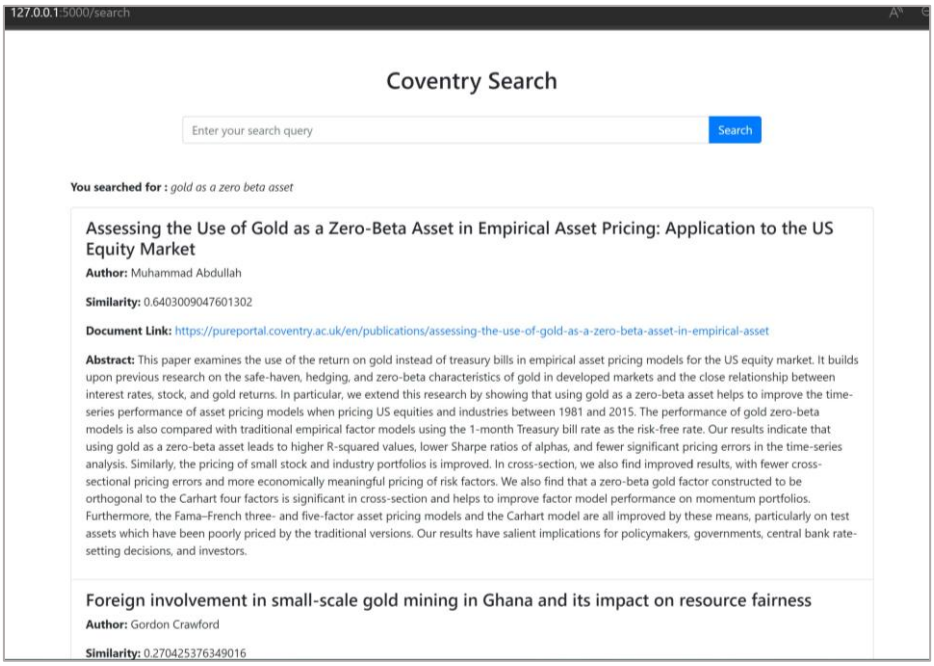


Figure 9 User Query search and results

As seen from the figure, the search engine provided the information relevant to the user's query. A cosine similarity of 0.64 can also be seen in the first result for the query which denotes a good similarity between the user's query and the result. This shows that the query processing is working properly.

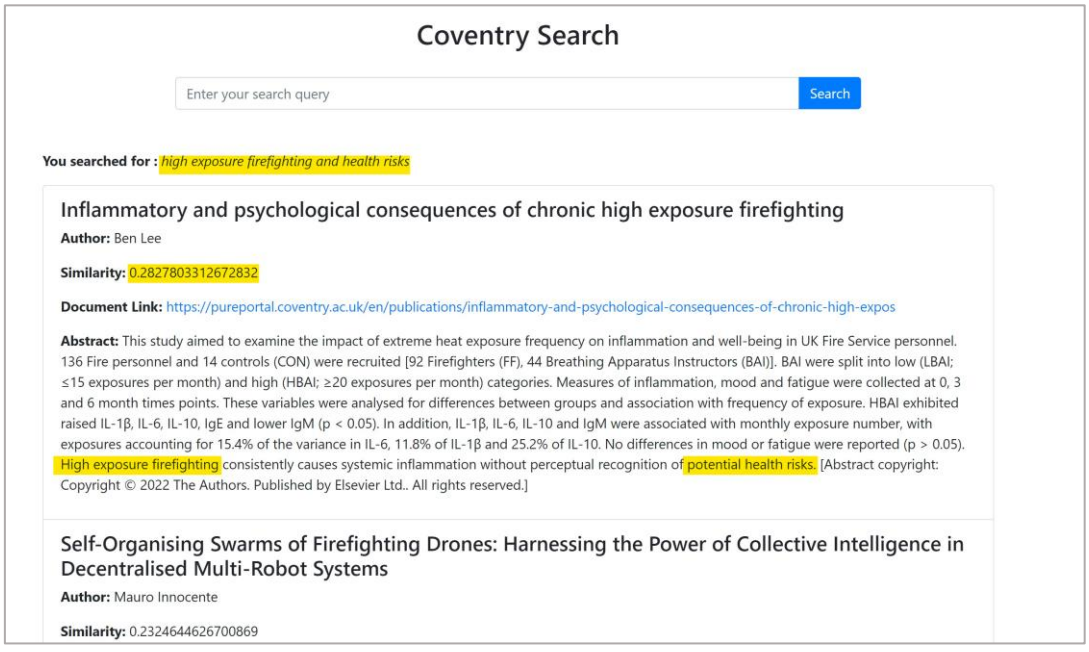


Figure 10 Searching another query with the query terms highlighted in the result

The text classification algorithms were programmed and executed separately from the search engine. As already discussed, the algorithms that were used for classification were Random Forest, SVM and Light GBM. The results of the scoring metrics for all the algorithms are as follows:

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.962428	0.964164	0.962428	0.962479
1	0.968208	0.968513	0.968208	0.968253
2	0.965217	0.965751	0.965217	0.965067

Figure 11Scoring Metrics - Random Forest Classifier

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.979769	0.980043	0.979769	0.979722
1	0.976879	0.977071	0.976879	0.976883
2	0.982609	0.983078	0.982609	0.982625

Figure 12 Scoring Metrics - Support Vector Machine

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.959538	0.960092	0.959538	0.959521
1	0.947977	0.948513	0.947977	0.948130
2	0.968116	0.969117	0.968116	0.968185

Figure 13 Scoring Metrics - Light GBM Classifier

The three tables above show the test scores of the text classifiers. The three rows of values arise from the 3-fold cross validation performed in the process. From the tables, the Support Vector Machine algorithm has the best performance of all. So, the **Support Vector Machine** classifier was chosen for predicting the categories.

The images below show the output of the classifiers when unseen data is used as input for them to classify the category of.

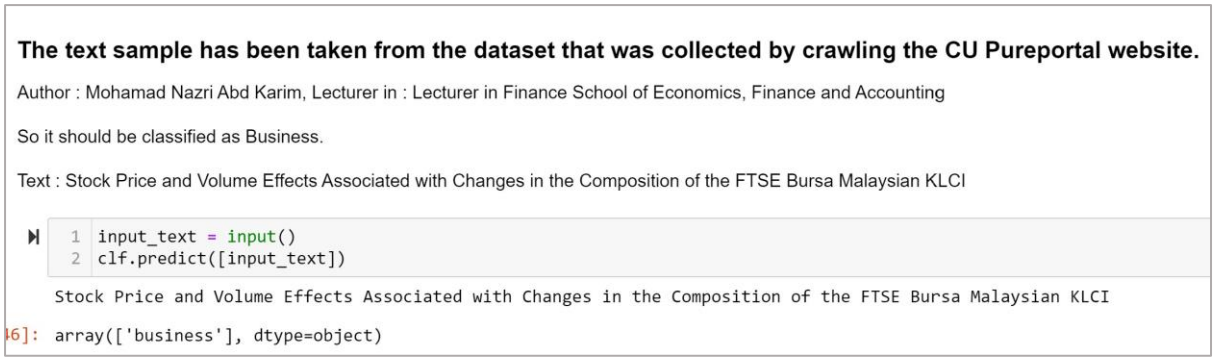


Figure 14 Text Classification

In the above figure, the sample text was taken from an article published by the CU author, Mohamad Nazri. The classifier was trained on BBC dataset, so this text data was unseen data for the model.

As the SVM classifier has an F1-score of 0.98, the model predicted the text's category successfully as "business".

Similarly, below are some more examples of the predictions made by the model on text taken from articles published by authors on Pureportal.

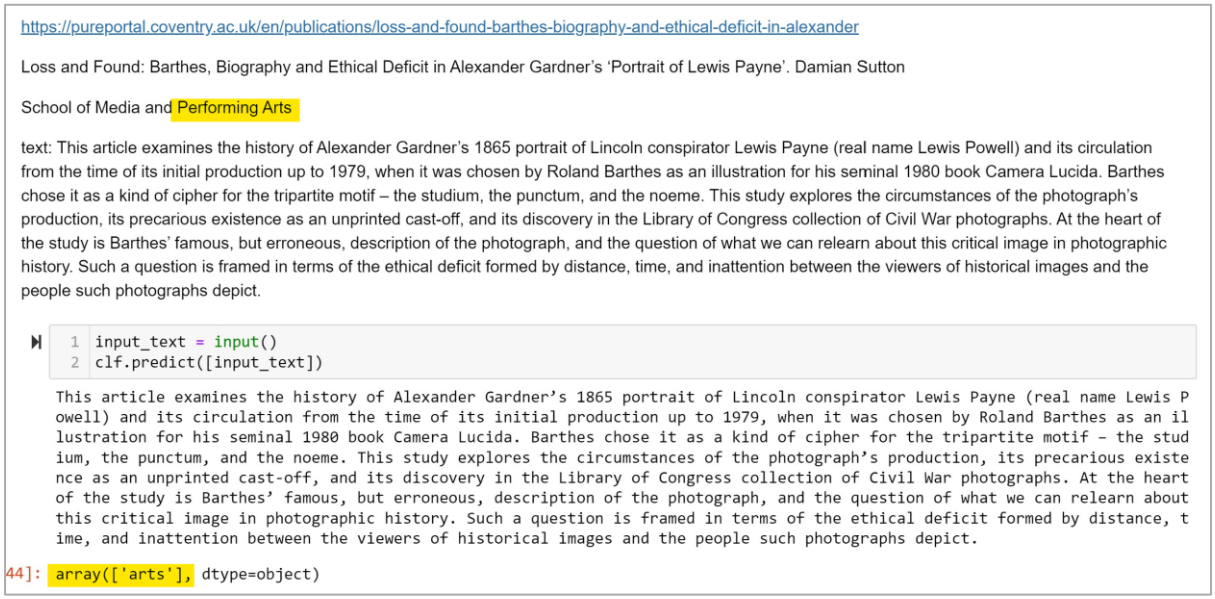


Figure 15 Arts Text classified as 'arts' by classifier

https://pureportal.coventry.ac.uk/en/publications/efficacy-of-the-4f-feedback-model-a-game-based-assessment-in-univ

Sara de Freitas, Victoria Uren, Kristian Kiili, Manuel Ninaus, Panagiotis Petridis, Petros Lameras, Ian Dunwell, Sylvester Arnab, Stephen Jarvis, Kam Star

Research Centre in Postdigital CulturesSchool of Computing, Mathematics and Data Sciences Aston UniversityCollege of Policing Tampere University of TechnologyUniversity of GrazUniversity of SuffolkIndependent Researcher

Text: Feedback is a critical aspect of optimised learning design, but there are few, if any, feedback models that map different types of feedback and how they may assist students to increase performance and enhance their learning experience. This research paper outlines a feedback model as an extension of the four-dimensional framework which includes a consideration of the type, the content, the format, and the frequency of feedback, as well as the agent which delivers it. This model is based upon an understanding of learning in the context of designing learning experiences and utilises a game-based model of learning to understand the importance of motivation and autonomy in learners to enhance and accelerate learning. The framework is developed and reflected upon by analysing two cases: a medical triage case in which the timing and frequency of feedback proved critical, and a business simulation which illuminated the need for a range of types of feedback and to be aware of the possibility of different agents (instructor peer and game) that can deliver feedback. The extended model may help game and learning designers alike to discern different types of feedback, both in games and more generally, in more explicit and nuanced ways.

```
1 input_text = input()
2 clf.predict([input_text])
```

```
Feedback is a critical aspect of optimised learning design, but there are few, if any, feedback models that map different ty
pes of feedback and how they may assist students to increase performance and enhance their learning experience. This researc
h paper outlines a feedback model as an extension of the four-dimensional framework which includes a consideration of the ty
pe, the content, the format, and the frequency of feedback, as well as the agent which delivers it. This model is based upon
an understanding of learning in the context of designing learning experiences and utilises a game-based model of learning to
understand the importance of motivation and autonomy in learners to enhance and accelerate learning. The framework is develo
ped and reflected upon by analysing two cases: a medical triage case in which the timing and frequency of feedback proved cr
itical, and a business simulation which illuminated the need for a range of types of feedback and to be aware of the possibi
lity of different agents (instructor peer and game) that can deliver feedback. The extended model may help game and learning
designers alike to discern different types of feedback, both in games and more generally, in more explicit and nuanced ways.
```

45]: array(['engineering'], dtype=object)

Figure 16 Engineering text classified as 'engineering'

References

Baghaee, H. a. (2020, September). Support Vector Machine-Based Islanding and Grid Fault Detection in Active Distribution Networks. *EEE Journal of Emerging and Selected Topics in Power Electronics*, 2385 - 2403.

Corporation, M. (n.d.). *Welcome to LightGBM's documentation!* Retrieved from LightGBM : https://lightgbm.readthedocs.io/en/v3.3.2/

Jiawei Han, M. K. (2012). 2 - Getting to Know Your Data. In M. K. Jiawei Han, *Data Mining: Concepts and Techniques* (pp. 39-82). Morgan Kaufmann. Retrieved from https://www.sciencedirect.com/topics/computer-science/cosine-similarity#:~:text=Cosine%20similarity%20measures%20the%20similarity,document%20similarity%20in%20text%20analysis.

Karbhari, V. (2020, January 29). *What is a cosine similarity matrix?* Retrieved from Medium: https://medium.com/acing-ai/what-is-cosine-similarity-matrix-f0819e674ad1

Marr, B. (2018, May 21). *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. Retrieved 6 2, 2023, from https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=6b9302dd60ba

scikit-learn. (n.d.). *sklearn.ensemble.RandomForestClassifier*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Simha, A. (2021, October 7). *Understanding TF-IDF for Machine Learning*. Retrieved from Capital One: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

Sivarajah, S. (2020, June 13). *"Sklearn's TF-IDF" vs "Standard TF-IDF"*. Retrieved from Medium: https://towardsdatascience.com/how-sklearns-tf-idf-is-different-from-the-standard-tf-idf-275fa582e73d

Wikipedia. (2023, May 22). *Vertical Search*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Vertical_search

wisdomml. (2022, August 5). *What are Stop words in NLP and Why we should remove them?* Retrieved from Wisdom ML: https://wisdomml.in/what-are-stopwords-in-nlp-and-why-we-should-remove-them/

Appendices

Github link: https://github.com/manashb21/IRassignment (The video showing the commands and program being executed is also on the same github respository.)

Screenshots of code for scarpy crawler:

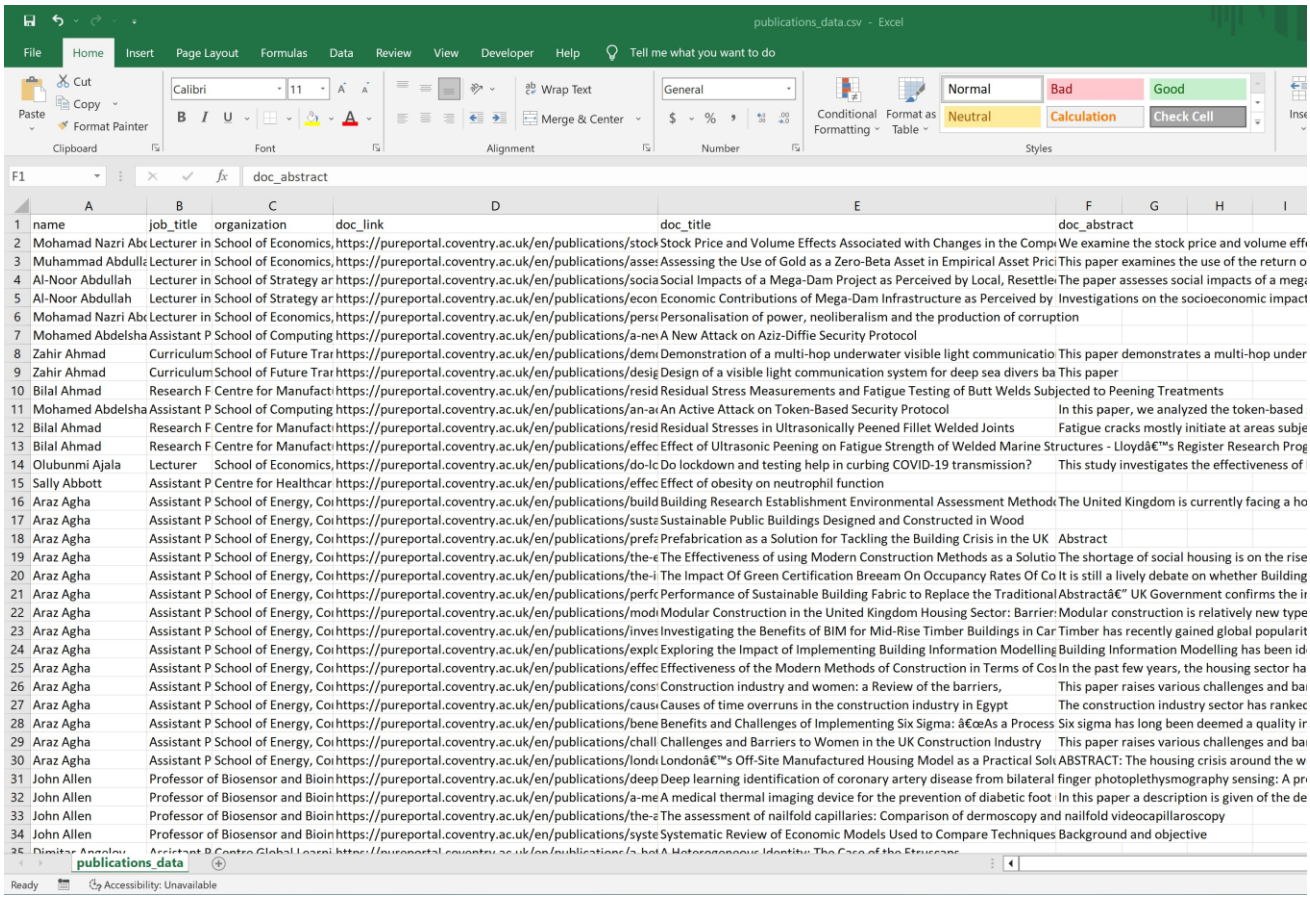
crawler > crawler > spiders > unispider.py > ...

```python
import scrapy
from scrapy.crawler import CrawlerProcess
from apscheduler.schedulers.twisted import TwistedScheduler
from datetime import datetime, timedelta

class UnispiderSpider(scrapy.Spider):
    name = "unispider"
    allowed_domains = [ "pureportal.coventry.ac.uk", "pureportal.coventry.ac.uk/en/persons/"]
    #so that the scarper doesn't scrape the entire internet
    start_urls = ["https://pureportal.coventry.ac.uk/en/persons"]

    def parse(self, response):
        persons = response.css('div.result-container')
        for person in persons:
            if person.css('ul li span.minor.dimmed::text').get() != None:
                    person_url =  person.css('h3 a').attrib['href']
                    publication_url = person_url + "/publications/"
                    yield response.follow(publication_url, callback = self.parse_person_pubs)

        next_page = response.css("li.next a ::attr(href)").get()
        if  next_page is not None:
            next_page_url = "https://pureportal.coventry.ac.uk" + next_page
            yield response.follow(next_page_url, callback = self.parse)

    def parse_person_pubs(self, response):
        publications = response.css("div.result-container h3.title")
        for publication in publications:
            doc_link = publication.css("a ::attr('href')").get()

            yield response.follow(doc_link, callback=self.get_abstract,
                                  meta={'publication': publication, 'rp' : response})

    def get_abstract(self, response):
        doc_abstract = response.xpath('//*[@id="main-content"]/section[1]/div/div/div[1]/div[1]/div/text()').get()
        publication = response.meta['publication']
        rp = response.meta['rp']

        yield {
            "name": rp.css('div.header.person-details h1::text').get(),
            "job_title": rp.css('span.job-title::text').get(),
            "organization": rp.css('a.link.primary span::text').get(),
            "doc_link": publication.css("a::attr(href)").get(),
            "doc_title": publication.css("span::text").get(),
            "doc_abstract": doc_abstract,
        }
```

Scheduler Implementation

```python
# Create a scheduler
scheduler = TwistedScheduler()

# Schedule the spider to run every 14 days
scheduler.add_job(
    lambda: CrawlerProcess({'USER_AGENT':
                            'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36'})
            .crawl(UnispiderSpider),
    'interval',
    days=14,
    start_date=datetime.now() + timedelta(days=1)  # Start the first run tomorrow
)

# Start the scheduler
scheduler.start()
```

Output CSV file from the Crawler:



	A	B	C	D	E	F
1	name	job_title	organization	doc_link	doc_title	doc_abstract
2	Mohamad Nazri Abd	Lecturer in	School of Economics,	https://pureportal.coventry.ac.uk/en/publications/stock	Stock Price and Volume Effects Associated with Changes in the Compe	We examine the stock price and volume effe
3	Muhammad Abdulla	Lecturer in	School of Economics,	https://pureportal.coventry.ac.uk/en/publications/asse	Assessing the Use of Gold as a Zero-Beta Asset in Empirical Asset Prici	This paper examines the use of the return o
4	Al-Noor Abdullah	Lecturer in	School of Strategy ar	https://pureportal.coventry.ac.uk/en/publications/socia	Social Impacts of a Mega-Dam Project as Perceived by Local, Resettle	The paper assesses social impacts of a mega
5	Al-Noor Abdullah	Lecturer in	School of Strategy ar	https://pureportal.coventry.ac.uk/en/publications/econ	Economic Contributions of Mega-Dam Infrastructure as Perceived by	Investigations on the socioeconomic impact
6	Mohamad Nazri Abd	Lecturer in	School of Economics,	https://pureportal.coventry.ac.uk/en/publications/pers	Personalisation of power, neoliberalism and the production of corruption	
7	Mohamed Abdelsha	Assistant P	School of Computing	https://pureportal.coventry.ac.uk/en/publications/a-ne	A New Attack on Aziz-Diffie Security Protocol	
8	Zahir Ahmad	Curriculum	School of Future Trar	https://pureportal.coventry.ac.uk/en/publications/dem	Demonstration of a multi-hop underwater visible light communicatio	This paper demonstrates a multi-hop under
9	Zahir Ahmad	Curriculum	School of Future Trar	https://pureportal.coventry.ac.uk/en/publications/desig	Design of a visible light communication system for deep sea divers ba	This paper
10	Bilal Ahmad	Research F	Centre for Manufact	https://pureportal.coventry.ac.uk/en/publications/resid	Residual Stress Measurements and Fatigue Testing of Butt Welds Subjected to Peening Treatments	
11	Mohamed Abdelsha	Assistant P	School of Computing	https://pureportal.coventry.ac.uk/en/publications/an-a	An Active Attack on Token-Based Security Protocol	In this paper, we analyzed the token-based
12	Bilal Ahmad	Research F	Centre for Manufact	https://pureportal.coventry.ac.uk/en/publications/resid	Residual Stresses in Ultrasonically Peened Fillet Welded Joints	Fatigue cracks mostly initiate at areas subje
13	Bilal Ahmad	Research F	Centre for Manufact	https://pureportal.coventry.ac.uk/en/publications/effec	Effect of Ultrasonic Peening on Fatigue Strength of Welded Marine Structures - Lloydâ€™s Register Research Prog	
14	Olubunmi Ajala	Lecturer	School of Economics,	https://pureportal.coventry.ac.uk/en/publications/do-lc	Do lockdown and testing help in curbing COVID-19 transmission?	This study investigates the effectiveness of
15	Sally Abbott	Assistant P	Centre for Healthcar	https://pureportal.coventry.ac.uk/en/publications/effec	Effect of obesity on neutrophil function	
16	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/build	Building Research Establishment Environmental Assessment Method	The United Kingdom is currently facing a ho
17	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/susta	Sustainable Public Buildings Designed and Constructed in Wood	
18	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/prefa	Prefabrication as a Solution for Tackling the Building Crisis in the UK	Abstract
19	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/the-e	The Effectiveness of using Modern Construction Methods as a Solutio	The shortage of social housing is on the rise
20	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/the-i	The Impact Of Green Certification Breeam On Occupancy Rates Of Co	It is still a lively debate on whether Building
21	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/perfo	Performance of Sustainable Building Fabric to Replace the Traditional	Abstractâ€" UK Government confirms the ir
22	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/mod	Modular Construction in the United Kingdom Housing Sector: Barriers	Modular construction is relatively new type
23	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/inves	Investigating the Benefits of BIM for Mid-Rise Timber Buildings in Car	Timber has recently gained global popularit
24	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/explo	Exploring the Impact of Implementing Building Information Modelling	Building Information Modelling has been id
25	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/effec	Effectiveness of the Modern Methods of Construction in Terms of Cos	In the past few years, the housing sector ha
26	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/cons	Construction industry and women: a Review of the barriers,	This paper raises various challenges and ba
27	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/caus	Causes of time overruns in the construction industry in Egypt	The construction industry sector has ranked
28	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/bene	Benefits and Challenges of Implementing Six Sigma: â€œAs a Process	Six sigma has long been deemed a quality ir
29	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/chall	Challenges and Barriers to Women in the UK Construction Industry	This paper raises various challenges and ba
30	Araz Agha	Assistant P	School of Energy, Co	https://pureportal.coventry.ac.uk/en/publications/lond	Londonâ€™s Off-Site Manufactured Housing Model as a Practical Sol	ABSTRACT: The housing crisis around the w
31	John Allen	Professor of	Biosensor and Bioin	https://pureportal.coventry.ac.uk/en/publications/deep	Deep learning identification of coronary artery disease from bilateral finger photoplethysmography sensing: A pr	
32	John Allen	Professor of	Biosensor and Bioin	https://pureportal.coventry.ac.uk/en/publications/a-me	A medical thermal imaging device for the prevention of diabetic foot	In this paper a description is given of the de
33	John Allen	Professor of	Biosensor and Bioin	https://pureportal.coventry.ac.uk/en/publications/the-a	The assessment of nailfold capillaries: Comparison of dermoscopy and nailfold videocapillaroscopy	
34	John Allen	Professor of	Biosensor and Bioin	https://pureportal.coventry.ac.uk/en/publications/syste	Systematic Review of Economic Models Used to Compare Techniques	Background and objective

Python Flask Web app:

app.py

```python
from flask import Flask, render_template, request
import logic_search as ls

app = Flask(__name__)

@app.route('/')


def index():
    return render_template('index.html')

@app.route('/search', methods=['POST'])
def search():
    query = request.form['query']
    final_list = ls.exec_logic(query)
    # Process the search query and perform search operations here
    return render_template('index.html', query=query, final_list = final_list)

if __name__ == '__main__':
    app.run(debug=True)
```

Search Engine Code:

app.py | logic_search.py

logic_search.py > exec_logic

```python
import pandas as pd
import string
import math
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def exec_logic(gui_query):

    pd.set_option('display.max_colwidth', 0)
    df = pd.read_csv("publications_data.csv")

    df['doc_content'] = df['name']+ ' ' + df['doc_title'] +" "+ df['doc_abstract']
    df = df.dropna()
    one_word_titles = df[df['doc_title'].str.count('\s') == 0]['doc_title']
    df = df[~df['doc_title'].isin(one_word_titles)]
    df = df.reset_index()
    df = df.drop(columns = 'index')

    df_final = df.drop(columns = ['doc_content'])

    def clean_docs(i, doc):
        stops = stopwords.words('english')
        words = doc.split()

        final = []

        for word in words:
            word = word.lower()
            word = word.replace('-', ' ')
            if word not in stops:
                final.append(word)

        final = " ". join(final)
        final = final.translate(str.maketrans("", "", string.punctuation))
        df_final.loc[i, 'doc_content'] = final
```

```
    for i in range(0, len(df)):
        data = df.loc[i, 'doc_content']
        clean_docs(i, data)

    documents = list(df_final['doc_content'])

    vectorizer = TfidfVectorizer(lowercase = True)

    tfidf_matrix = vectorizer.fit_transform(documents)
    # print(tfidf_matrix.toarray())

    def query_processor(query):
        stops = stopwords.words('english')
        words = query.split()

        final = []

        for word in words:
            word = word.lower()
            word = word.replace('-', ' ')
            if word not in stops:
                final.append(word)

        final = " ". join(final)
        final = final.translate(str.maketrans("", "", string.punctuation))
        return final


    query = gui_query
    processed_query = query_processor(query)
    #Transform the query using the fitted vectorizer
    query_vector = vectorizer.transform([processed_query])

    #Compute the cosine similarity
    cosine_similarities = cosine_similarity(query_vector, tfidf_matrix)

    #Find the most similar document, argsort() causes the max value's index to be at the last
    most_similar_doc_index = cosine_similarities.argsort()[0]
    #converting most_similar_doc_index to a list so that the index values can be reversed (in descending order)
    similar_index_list = list(most_similar_doc_index)
    similar_index_list.reverse()
```

```
    similarities = []
    similarities_percentage = []
    most_similar_doc = []

    for index in similar_index_list:
        if cosine_similarities[0][index] > 0:
            similarities.append(cosine_similarities[0][index])
            most_similar_doc.append(documents[index])

    for s in similarities:
        similarities_percentage.append(((s + 1) / 2 * 100))

    # print("Most similar document: \n", most_similar_doc[0])

    final_result = []

    for i in range(0, len(most_similar_doc)):
        result = df_final.loc[similar_index_list[i]]

        final_result.append({"Similarity": similarities[i],
                             "Author": result['name'],
                             "Document Link": result['doc_link'],
                             "Title" : result['doc_title'],
                             "Abstract" : result['doc_abstract']
                             })

    return final_result
```

Text Classifier Code:

```
import os
import string

# Data Handling and Processing
import pandas as pd
import numpy as np
import re
from scipy import interp


# Visualuzation
import matplotlib.pyplot as plt
%matplotlib inline
#to render high res images
%config InlineBackend.figure_format='retina'

# NLP Packages
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
import nltk
from joblib import dump, load


# ML Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier

# Scikit Learn packages
from sklearn.base import clone
from sklearn.preprocessing import label_binarize, LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import KFold, cross_validate, cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

```

Load Datasets

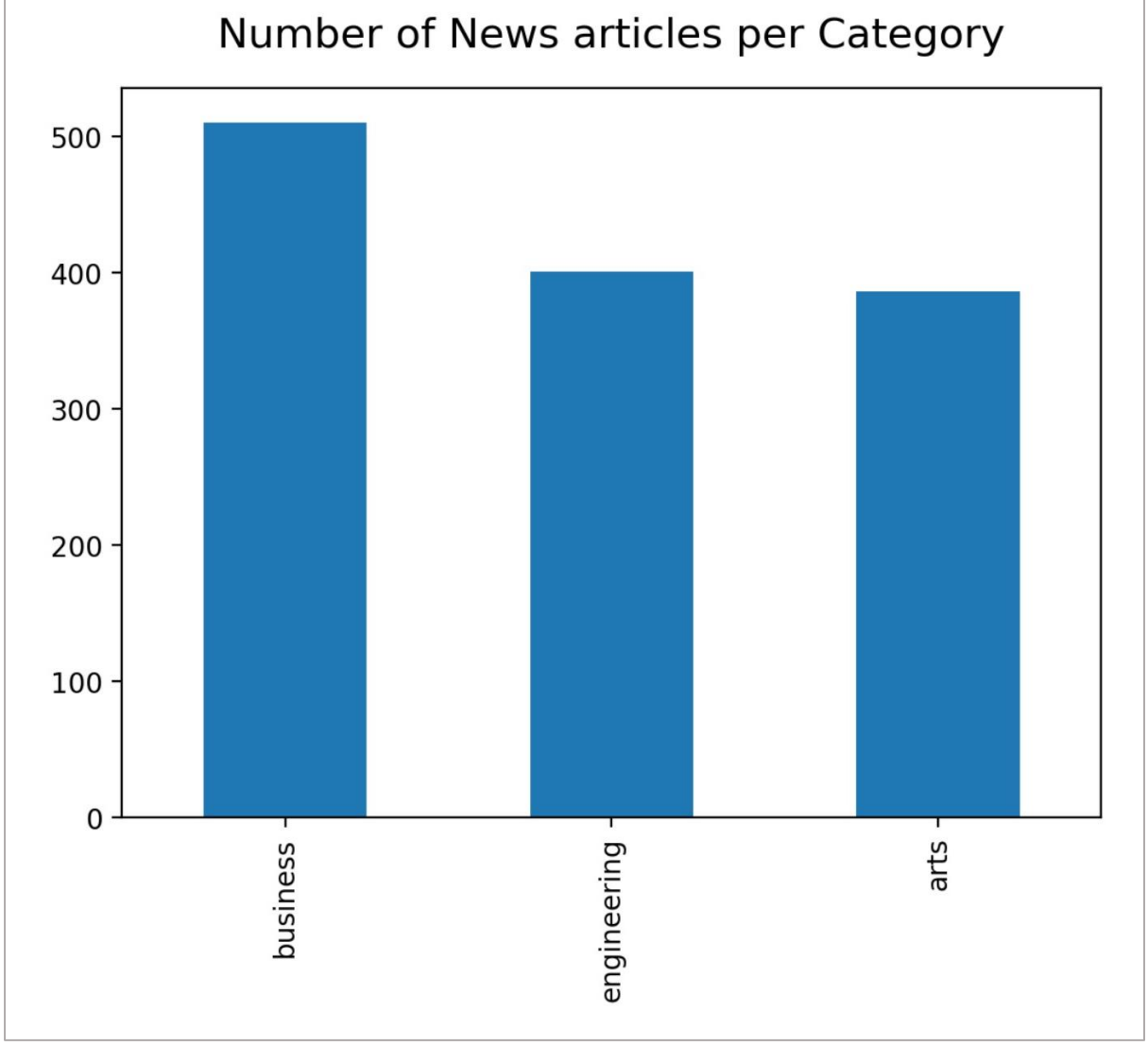
```
df = pd.read_csv('bbc-text.csv')
print(df.shape)

```

```
(2225, 2)
```

```
1 df.head()
```

	category	text
0	engineering	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	arts	ocean s twelve raids box office ocean s twelve...
3	arts	last star wars not for children the sixth an...
4	arts	berlin cheers for anti-nazi film a german movi...

```
1 df['category'].value_counts().plot(kind='bar')
2 plt.title('Number of News articles per Category', size=15, pad=15);
```



text processing step

```
def text_preprocess(df):
    # Remove special characters
    df['p_text'] = df['text'].replace('\n',' ')
    df['p_text'] = df['p_text'].replace('\r',' ')

    # Remove punctuation signs and lowercase all
    df['p_text'] = df['text'].str.lower()
    df['p_text'] = df['text'].str.translate(str.maketrans('', '', string.punctuation))


    # Remove stop words
    stop_words = stopwords.words("english")
    lemmatizer = WordNetLemmatizer()

    def fwpt(each):
        tag = pos_tag([each])[0][1][0].upper()
        hash_tag = {"N": wordnet.NOUN,"R": wordnet.ADV, "V": wordnet.VERB,"J": wordnet.ADJ}
        return hash_tag.get(tag, wordnet.NOUN)


    def lematize(text):
        tokens = nltk.word_tokenize(text)
        ax = ""
        for each in tokens:
            if each not in stop_words:
                ax += lemmatizer.lemmatize(each, fwpt(each)) + " "
        return ax

    df['p_text'] = df['p_text'].apply(lematize)
```

```
text_preprocess(df)
```

train test split

```
X_train, X_test, y_train, y_test = train_test_split(df['p_text'],
                                                    df['category'],
                                                    test_size=0.2,
                                                    random_state=9)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1037,) (260,) (1037,) (260,)
```

Using tfidfvectorizer sklearn

```
vector = TfidfVectorizer(stop_words='english',
                         ngram_range = (1,2),
                         min_df = 3,
                         max_df = 1.,
                         max_features = 10000)
```

```
#creating fit model function

def fit_model(model, model_name):
    line = Pipeline([('vectorize', vector), (model_name, model)])

    output = cross_validate(line,
                            X_train,
                            y_train,
                            cv = KFold(shuffle = True,
                                       n_splits = 3,
                                       random_state = 9),
                            scoring = ('accuracy', 'f1_weighted','precision_weighted','recall_weighted'),
                            return_train_score=True)
    return output
```

```
rf_clf = fit_model(RandomForestClassifier(), 'Randf')
svm_clf = fit_model(SVC(), 'SVM')
lgbm_clf = fit_model(LGBMClassifier(), 'LGBM')

rf = pd.DataFrame.from_dict(rf_clf)
sv = pd.DataFrame.from_dict(svm_clf)
lg = pd.DataFrame.from_dict(lgbm_clf)
```

```
l1 = [rf, sv, lg]
l2 =["Randf", "SVM", "LGBM"]

for each, tag in zip(l1, l2):
    each['model'] = [tag, tag, tag]

joined_output = pd.concat([rf, sv, lg])
```

```
1 rf_clf
```

```
{'fit_time': array([1.57403708, 1.64364266, 1.85054135]),
 'score_time': array([0.16299629, 0.15400004, 0.213027  ]),
 'test_accuracy': array([0.95953757, 0.96820809, 0.95942029]),
 'train_accuracy': array([1., 1., 1.]),
 'test_f1_weighted': array([0.9594442 , 0.96831353, 0.95948247]),
 'train_f1_weighted': array([1., 1., 1.]),
 'test_precision_weighted': array([0.96080775, 0.9687316 , 0.96195102]),
 'train_precision_weighted': array([1., 1., 1.]),
 'test_recall_weighted': array([0.95953757, 0.96820809, 0.95942029]),
 'train_recall_weighted': array([1., 1., 1.])}
```

```
1 svm_clf
```

```
{'fit_time': array([1.57499242, 1.65300179, 1.69954896]),
 'score_time': array([0.41600108, 0.52000117, 0.50400043]),
 'test_accuracy': array([0.97976879, 0.97687861, 0.9826087 ]),
 'train_accuracy': array([1., 1., 1.]),
 'test_f1_weighted': array([0.97972236, 0.9768829 , 0.98262483]),
 'train_f1_weighted': array([1., 1., 1.]),
 'test_precision_weighted': array([0.98004298, 0.97707112, 0.98307751]),
 'train_precision_weighted': array([1., 1., 1.]),
 'test_recall_weighted': array([0.97976879, 0.97687861, 0.9826087 ]),
 'train_recall_weighted': array([1., 1., 1.])}
```

```
1 lgbm_clf
```

```
{'fit_time': array([2.47426343, 2.18765259, 2.14998913]),
 'score_time': array([0.19399786, 0.19899988, 0.24099994]),
 'test_accuracy': array([0.95953757, 0.94797688, 0.96811594]),
 'train_accuracy': array([1., 1., 1.]),
 'test_f1_weighted': array([0.95952149, 0.94812999, 0.9681848 ]),
 'train_f1_weighted': array([1., 1., 1.]),
 'test_precision_weighted': array([0.96009208, 0.94851315, 0.96911688]),
 'train_precision_weighted': array([1., 1., 1.]),
 'test_recall_weighted': array([0.95953757, 0.94797688, 0.96811594]),
 'train_recall_weighted': array([1., 1., 1.])}
```

```
relevant_measures = list(['test_accuracy','test_precision_weighted', 'test_recall_weighted', 'test_f1_weighted'])

random_forest_metrics = joined_output.loc[joined_output.model == 'Randf'][relevant_measures]
svm_metrics = joined_output.loc[joined_output.model == 'SVM'][relevant_measures]
lgbm_metrics = joined_output.loc[joined_output.model == 'LGBM'][relevant_measures]
```

Random Forest metrics

```
random_forest_metrics
```

2]:

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.959538	0.960808	0.959538	0.959444
1	0.968208	0.968732	0.968208	0.968314
2	0.959420	0.961951	0.959420	0.959482

SVM Metrics

```
svm_metrics
```

3]:

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.979769	0.980043	0.979769	0.979722
1	0.976879	0.977071	0.976879	0.976883
2	0.982609	0.983078	0.982609	0.982625

LGBM Metrics

```
lgbm_metrics
```

4]:

	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.959538	0.960092	0.959538	0.959521
1	0.947977	0.948513	0.947977	0.948130
2	0.968116	0.969117	0.968116	0.968185

Selection of Model : model selected -> SVM

```
# Join training and test datasets
X = pd.concat([X_train,
               X_test])
y = pd.concat([y_train,
               y_test])
```

```
def model_fit(clf, x, y):
    best_clf = clf
    pipeline = Pipeline([('vectorize', vector), ('model', best_clf)])
    return pipeline.fit(x, y)
```

```
# Create model
clf = model_fit(SVC(), X, y)
```

```
clf.classes_
```

7]: array(['arts', 'business', 'engineering'], dtype=object)

The text sample has been taken from the dataset that was collected by crawling the CU Pureportal website.

Author : Mohamad Nazri Abd Karim, Lecturer in : Lecturer in Finance School of Economics, Finance and Accounting

So it should be classified as Business.

Text : Stock Price and Volume Effects Associated with Changes in the Composition of the FTSE Bursa Malaysian KLCI

```
input_text = input()
clf.predict([input_text])
```

```
Stock Price and Volume Effects Associated with Changes in the Composition of the FTSE Bursa Malaysian KLCI
```

6]: array(['business'], dtype=object)