

**Contents**

[Part.1 Vertical search engine 2](#_Toc131159637)

[1. Crawler 2](#_Toc131159638)

[2. Indexer 6](#_Toc131159639)

[3. Query processor 9](#_Toc131159640)

[Part.1 Document clustering 12](#_Toc131159641)

[Dataset used 12](#_Toc131159642)

[Method / algorithm 12](#_Toc131159643)

[Type of clustering 13](#_Toc131159644)

[Clustering screenshots 13](#_Toc131159645)

[Working 18](#_Toc131159646)

[Appendix 19](#_Toc131159647)

## Part.1 Vertical search engine

A vertical search engine focuses on a specific domain or topic, such as news, images, videos, products, or employment. Vertical search engines, as opposed to general search engines such as Google or Bing, only index content pertinent to their specific niche or domain. Vertical search engines use specialised algorithms and ranking factors to deliver more accurate and relevant results to users conducting searches within a specific domain. A job search engine may enable users to filter job listings by location, salary, or job title, whereas a product search engine may enable users to arrange products by price, brand, or customer ratings. Indeed for job listings, Yelp for local business evaluations, and Kayak for travel-related information are well-known vertical search engines.

## Crawler

In a vertical search engine, a crawler is a software programme used to scan the web and acquire information pertinent to the specific domain or topic covered by the search engine. The crawler follows relevant connections to websites, indexes their content, and stores it in the database of the search engine. For instance, a job search engine's crawler would collect information such as job titles, descriptions, locations, and salaries by crawling job posting websites. This information is then organised and made accessible to users conducting employment searches on the search engine. Crawlers utilised by vertical search engines may have features and functions distinct from those utilised by general search engines. For instance, they may be programmed to concentrate on particular domains or types of content, or to extract information beyond text, such as images, videos, and user reviews.

Critical to the quality of a vertical search engine's search results is the performance of its crawler. A well-designed crawler can help ensure that the database of a search engine is comprehensive, up-to-date, and relevant to the requirements of its users.

* 1. A total number of 47 profiles are crawled from Research Centre for Computational Science and Mathematical Modelling department.



Graphical user interface, application

Description automatically generated

Figure 1 : number of profiles crawled

Graphical user interface, text, application

Description automatically generated

Figure 2 : Total number of publications retrieved

Text

Description automatically generated

Figure 3 :sample publication links

* 1. Information collected about each publication

Author’s name, title, paperlink are collected from each paper’s metadata.

Graphical user interface, application

Description automatically generated with medium confidence

Figure 4 : information collected for each paper

* 1. Which pre-processing tasks are performed before passing data to Indexer/Elastic Search

I have performed some pre-processing tasks before passing it to the indexer.

Preprocessing tasks includes removing unwanted columns from data farme , stop word removal and lemmatization.

A picture containing company name

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Figure 5 :preprocessing tasks done before the indexer

* 1. When the crawler operates, such as when it is scheduled or manually run

This labour requires manual, step-by-step execution. As with python IDE, the crawler must execute each cell manually when using Google Colab and which can store output csv file into drvie.

* 1. Explanation

The system I've devised specifically for this program will "crawl" the relevant websites in order to collect the necessary data in order to collect information on all of the required articles. Among the details we compile for each book are its authors, publication year, title, and connections to the publication's website and the author's profile page. (also known as the "pureportal" profile page). On this task I need to crawl publications of only Research Centre for Computational Science and Mathematical Modelling — Coventry University. So the source url directs to this page.

It takes some time to update this information, so I have scheduled a time for my crawler to look for new information. However, it would be preferable if it could do so automatically on its own. It is anticipated that when the programme is executed, the index will be updated so that it reflects the most recent information.

I have used python ide UI for this vertical search engine. In this section, the user has the option to input queries or keywords to search for particular types of content. My system will then organise the findings into a list, similar to how Google Scholar does, and rank the items in the list according to their relevance. In contrast, when you conduct a search, only articles written by members of the computing and mathematical engineering department at Coventry University will be displayed. I was first given a data frame so that I could investigate the results, and then the information was saved as a links.csv file. Iam saving this file to my drive.

Company name

Description automatically generated with medium confidence

Figure 6 : saving crawled dataframe into a excel file

## Indexer

* 1. I have implemented index search method.

Indexing is the process of adding web pages and other categories of content to a vertical search engine's database so that they can be searched and retrieved by users. During indexing, a web crawler or spider programme follows web links to discover new pages, which are then added to the search engine's index. The indexing process of a vertical search engine is more concentrated on a specific topic or industry. The web crawler may be programmed to seek out content that is pertinent to a particular topic or industry and to disregard irrelevant content. A job search engine, for instance, may only index pages containing job listings, while disregarding pages containing recipes or travel information.

Once a page has been indexed, it is searchable and retrievable by users who are looking for relevant content. The search engine's algorithms will analyse the page's content, including the page's title, meta tags, and other factors, to determine the page's relevance to the user's search query. Because the database of a vertical search engine is more limited in scope than that of a traditional search engine, indexing is crucial. This necessitates more precise indexing to ensure that users discover the most relevant content in response to their search queries.

2.2 In this assignment, I have utilised an inverted index data structure.

Vertical search engines heavily rely on inverted indexing as a crucial technique for quickly retrieving relevant information from a large content database. In the process of creating an index using inverted indexing, each word or term within the content is mapped to the specific web pages or documents on which it appears.

Graphical user interface

Description automatically generated with medium confidence

Figure 7 : importing libraries and wordlemmatizer

2.3 Inverted index implementation is not incremental. It is generated whenever we execute crawler

2.4 screenshots

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Figure 8 : index process functions

Graphical user interface, text, application

Description automatically generated

Figure 9 : constructed dictionary

2.5 Working

I will perform a lemmatization test on the entire first row to get a sense of how everything fits together. Every punctuation mark has been removed from the text, and it has been formatted to use only uppercase characters throughout. Using the POS Tag lemmatizer, it is possible to lemmatize morphological structures. In this section, we will endeavour to develop a function that, using the lemmatization technique, can determine the grammatical function of a given word. Following this, I will straighten up the text by removing all periods and other punctuation marks. By utilising the function, each DataFrame document will be converted to all capital letters. To vectorize the data, it is necessary to first merge all of the columns from the cleansed data, then incorporate the improved DataFrame into the existing one, and finally return the data to its original state. After that, the procedure for designating labels to the dictionary followed. Create a posting list by dividing the items of the dictionary into more manageable dictionaries and including each id in its corresponding segment. Now that everything has been completed, the indexer has access to every piece of data.

## Query processor

3.1 Query preprocessing



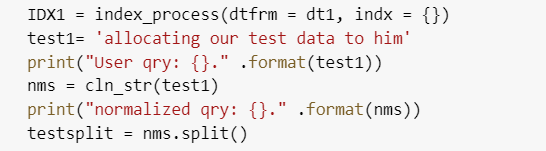


Figure 10 : preprocessing before query processor

3.2 This vertical search engine supports keywords including support Boolean queries and without Boolean operators also.

3.3 I used vector space . Vectorization is a crucial technique utilised by a multitude of search engines, including vertical search engines. Vectorization is the process of converting text data (such as user queries or documents) into numerical vectors that can be used for machine learning and other computational operations in the context of search engines. Vectorization can be used to represent various aspects of the content being searched in a vertical search engine. For instance, vectorization can be used to represent job titles, job descriptions, company names, and other pertinent information in a job search engine. This enables the search engine to match user queries with relevant results more effectively. Utilizing techniques such as term frequency-inverse document frequency (TF-IDF) or word embeddings is a common approach to vectorization. TF-IDF assigns weights to individual words based on their frequency of occurrence in a document and their prevalence across the corpus as a whole. In contrast, word embeddings represent words as dense numerical vectors that convey their semantic meaning.

3.4 Working of query processor

Graphical user interface, text, application

Description automatically generated

Figure 11 : input screen

Graphical user interface

Description automatically generated

Figure 12 : number of matching documents

Graphical user interface, text, application, Word

Description automatically generated

Figure 13 : Most relevant document

3.5 Working

As part of this assignment, I developed a vertical search engine employing the indexer technique. When we input a phrase or keyword into the search bar, the search engine will remove stop words and other extraneous text before passing the phrase or keyword to the indexer. The query processor is also used to retrieve the data from the examined data.

## Part.1 Document clustering

## Dataset used

I have clustered the dataset using fetch\_20newsgroups.

The fetch\_20newsgroups dataset is a well-known dataset used in text classification projects. It can be accessed through the sklearn Python framework, which is part of the Scikit-learn suite. It contains a compilation of approximately 20,000 newsgroup postings on twenty distinct topics, including politics, athletics, religion, science, and technology, among many others. The dataset is divided into two subsets, with one comprising approximately 11,000 posts for training and the other containing approximately 9,000 posts for testing. Each message is recorded as a string of text, and its metadata includes information about the newsgroup category, the author's identity and email address, and the date the message was posted.

Used categories are hockey for sports, hardware for tech, space for climate.

## Method / algorithm

Clustering is performed using naive Bayes.

Clustering and Naive Bayes are two distinct machine learning methods used for distinct purposes. The probabilistic classification algorithm Naive Bayes is founded on Bayes' theorem. It implies that the presence of one trait within a class is independent of the presence of other traits. This is why it is considered to be a "naive" assumption. Spam filtering and sentiment analysis are common Naive Bayes text classification applications. It computes the likelihood of a document belonging to a specific class based on the frequency of its characteristics.

The Naive Bayes algorithm is implemented by initially training a model on a labelled dataset with known features and classes. Using this model, the algorithm categorises newly generated documents into one of the predefined classes. The model calculates the probability that a document belongs to each class based on the occurrence of each class's characteristics and then assigns the document to the class with the highest probability.

Clustering, in contrast, is an unsupervised learning method. It involves clustering comparable data points without knowing their class labels beforehand. Typical applications of clustering include customer segmentation and image segmentation. The goal of clustering algorithms is to decrease the distance between data points within the same cluster while increasing the distance between data points in distinct clusters.

## Type of clustering

Hard clustering is used for performing naïve bayes algorithm.

## Clustering screenshots

Text

Description automatically generated

Figure 14 importing libraries

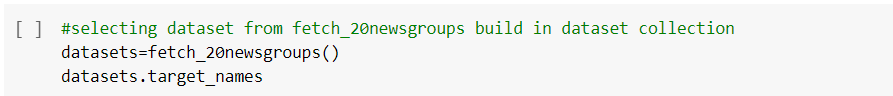


Figure 15 dataset fetching

Text

Description automatically generated

Figure 16 data sample

Graphical user interface, text

Description automatically generated

Figure 17 :categories and training data

Graphical user interface, text, application

Description automatically generated

Figure 18 : vectorizer for pre-processing

Chart, waterfall chart

Description automatically generatedfigure 19 : confusion matrix

Graphical user interface, text, application

Description automatically generated

Figure 20 : accuracy and predict function

Graphical user interface, application

Description automatically generated

Figure 21 : short queries testing

Graphical user interface, application

Description automatically generated A picture containing table

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Graphical user interface, application

Description automatically generated

Figure 22 : different modes of testing

## Working

I'm loading the 20newsgroups dataset, which contains approximately 20,000 newsgroup posts on 20 unique topics. Clustering will only be considered for three categories. Vectorize the documents: Using Scikit-learn's TfidfVectorizer, we convert the document's raw text into a vector of numerical values representing each word's significance. Decompose the matrix using NMF: Using the NMF algorithm, the vectorized matrix is decomposed into a set of basis vectors (W) and their coefficients. (H). This reduces the dimensionality of the data and makes clustering easier.

• Group the documents with MultinomialNB

• Heatmaps are used for confusion matrix plotting

• Obtained precision score 0.99915611814346

• Testing of short and lengthy queries, complex sentences, punctuation, and stop words is performed.

# Appendix

Task1 :

#source url

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

burl\_\_ = requests.get(urlforcrawl)

urlforcrawl\_source\_url= burl\_\_.text

htm='html.parser'

sp= BeautifulSoup(urlforcrawl\_source\_url, htm)

linksfor\_publications=[ ]

tt='title'

for h3tag in sp.find\_all("h3", class\_= tt):

atag= h3tag.find('a')

links= atag.attrs['href']

linksfor\_publications.append(links)

linksfor\_publications

print(len(linksfor\_publications))

linksfor\_publications=linksfor\_publications[:50]

linksfor\_publications

total\_pages= np.arange(1, 5, 1)

pub\_array= []

for pg in total\_pages:

ur='https://pureportal.coventry.ac.uk/en/organisations/coventry-university/persons/?page='

url\_\_= ur + str(pg)

page\_results= requests.get(url\_\_)

sps= BeautifulSoup(page\_results.text,htm)

for h3tag1 in sps.find\_all("h3", class\_= 'title'):

atag1= h3tag1.find('a')

links1= atag1.attrs['href']

pub\_array.append(links1)

len(pub\_array)

pub\_array=pub\_array[:47]

linksfor\_publications= linksfor\_publications + pub\_array

linksfor\_publications

pub=[]

for url\_\_ in linksfor\_publications:

pub\_page = requests.get(url\_\_)

html = pub\_page.text

sp = BeautifulSoup(html, "html.parser")

for it in sp.find\_all('h3', class\_ ='title'):

path= it.find('a')

pub\_url\_= path['href']

ur='https://pureportal.coventry.ac.uk/en/publications/'

if ur not in pub\_url\_:

continue

pub.append(pub\_url\_)

len(pub)

pub

pub\_con= []

array\_cont= []

for cnt in pub:

try:

ctnt = requests.get(cnt)

ht = ctnt.text

sp1 = BeautifulSoup(ht, htm)

rw= 'row'

rd='rendering'

for it in sp1.findAll('div', class\_=rw):

divtitle= it.findChild('div', class\_= rd)

title1 = divtitle.find('h1')

pub\_con.append(title1.text)

rt='relations persons'

for it in sp1.findAll('p', class\_= rt):

a1= it.findChild('span').text

array\_cont.append(a1)

except:

pass

len(array\_cont)

len(pub\_con)

dt1= pd.DataFrame({'Title':pub\_con, 'Author':array\_cont, 'Paperlink':pub })

dt1['id'] = [i for i in range(1, len(dt1.values)+1)]

dt1

# Commented out IPython magic to ensure Python compatibility.

from google.colab import drive

drive.mount('/content/drive')

# %cd /content/drive/My Drive/learn-ai-bbc/

dt1.to\_csv('./links.csv')

dt1= pd.read\_csv('./links.csv')

dt1.head(10)

dt1= dt1.drop('Unnamed: 0', 1)

len(dt1)

etr= dt1.loc[0,:].copy()

def text\_cleaning(tt):

tt = tt.lower()

tt = tt.translate(str.maketrans('', '', string.punctuation))

return tt

def get\_word(wd):

lemmentaiser = pos\_tag([wd])[0][1][0].upper()

dct = {"J": wordnet.ADJ,

"N": wordnet.NOUN,

"V": wordnet.VERB,

"R": wordnet.ADV}

return dct.get(lemmentaiser, wordnet.NOUN)

import nltk

import string

from nltk.corpus import stopwords

from nltk.corpus import wordnet

from nltk import pos\_tag

from nltk.stem import WordNetLemmatizer

nltk.download('averaged\_perceptron\_tagger')

nltk.download('wordnet')

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('omw-1.4')

pt="Testing: {}\n Code: {}\n Is: {}\n ON: {}\n"

print(pt.format(get\_word("Testing"), get\_word("Code"), get\_word("Is"), get\_word("ON")))

stp\_wrds = stopwords.words('english')

lmtizer= WordNetLemmatizer()

def Lemmatize(doc):

tkn = nltk.word\_tokenize(doc)

tmp = ""

for tp in tkn:

if tp not in stp\_wrds:

tmp += lmtizer.lemmatize(tp, get\_word(tp)) + " "

return tmp

# Commented out IPython magic to ensure Python compatibility.

Lemmatize(doc = etr.Title)

def cln\_str(tt):

tt = tt.lower()

tt = tt.translate(str.maketrans('', '', string.punctuation))

tt = Lemmatize(tt)

return tt

# %time cln\_str(etr.Title)

Cl\_dt = dt1.copy()

# Commented out IPython magic to ensure Python compatibility.

def convert\_to\_dataframe(dtfrm):

dtfrm['Author'] = dtfrm['Author'].apply(cln\_str)

dtfrm['Title'] = dtfrm['Title'].apply(cln\_str)

# %time convert\_to\_dataframe(Cl\_dt)

Cl\_dt['txt'] = Cl\_dt["Title"] + " " + Cl\_dt["Author"]

Cl\_dt = Cl\_dt.drop(["Author","Title", "Paperlink"], axis=1)

def convert\_to\_dataframe(dtfrm):

dtfrm = dtfrm

dtfrm['Author'] = dtfrm['Author'].apply(cln\_str)

dtfrm['Title'] = dtfrm['Title'].apply(cln\_str)

dtfrm['txt'] = dtfrm['Author'] + " " + dtfrm['Title']

dtfrm = dtfrm.drop(["Author","Title", "Paperlink"], axis=1)

return dtfrm

dct = Cl\_dt.loc[0,:].copy()

print(dct)

test = {}

def indexfunctionmain(dct, indx):

wd = dct.txt.split()

id = dct.id

for a in wd:

if a in indx.keys():

if id not in indx[a]:

indx[a].append(id)

else:

indx[a] = [id]

return indx

Itd\_indx = indexfunctionmain(dct, indx= {})

print(Itd\_indx)

def indxing1(dtfrm, indx ):

for i in range(len(dtfrm)):

Document1 = dtfrm.loc[i,:]

indx = indexfunctionmain(Document1, indx = indx)

return indx

Cl\_dt

indexss = indxing1(Cl\_dt, indx = {})

len(indexss)

print(indexss)

def index\_process(dtfrm, indx):

combine = convert\_to\_dataframe(dtfrm)

indx = indxing1(combine, indx = indexss)

return indx

IDX1 = index\_process(dtfrm = dt1, indx = {})

test1= 'allocating our test data to him'

print("User qry: {}." .format(test1))

nms = cln\_str(test1)

print("normalized qry: {}." .format(nms))

testsplit = nms.split()

def processorqry(qry):

PQ = cln\_str(qry)

return PQ.split()

rtr = []

for wrd in testsplit:

if wrd in IDX1.keys():

rtr.append(indexss[wrd])

def L\_instion(lists1):

inst = []

if len(lists1) > 0:

inst = list(set.intersection(\*map(set, lists1)))

inst.sort()

return inst

rst = L\_instion(rtr)

print(rst)

def searchengine\_fun(qry, indx=IDX1):

qry\_split = processorqry(qry)

retrieves = []

for w in qry\_split:

if w in indx.keys():

retrieves.append(indx[w])

if len(retrieves)>0:

search\_rslt = L\_instion(retrieves)

else:

search\_rslt = 'No Information Found'

return search\_rslt

qr=input('enter the key word to search:')

rslt= searchengine\_fun(qr, indexss)

print(rslt)

meatadata = dt1.drop(['txt'], axis=1).copy()

meatadata.head(15)

def id\_dataframeset(retrieved\_id, dtfrm):

return dtfrm[dtfrm.id.isin(retrieved\_id)].reset\_index(drop=True)

rst\_metadata = id\_dataframeset(rslt, meatadata)

print(rst\_metadata)

rst\_metadata.head(5)

Task 2:

#selecting dataset from fetch\_20newsgroups build in dataset collection

datasets=fetch\_20newsgroups()

datasets.target\_names

datasets

categories = [

"rec.sport.hockey",

"comp.sys.ibm.pc.hardware",

"sci.space",

]

print("Loading 20 newsgroups dataset for categories:")

print(categories)

#training data on selected categories

train = fetch\_20newsgroups(

subset="all", categories=categories, shuffle=True, random\_state=42)

print("%d documents" % len(train.data))

print("%d categories" % len(train.target\_names))

print()

train.data

"""# DataPreprocessing and Clustering"""

vectorizer=TfidfVectorizer()

X = vectorizer.fit\_transform(train.data)

model=make\_pipeline(TfidfVectorizer(),MultinomialNB())

model.fit(train.data,train.target)

test=fetch\_20newsgroups(

subset="test", categories=categories, shuffle=True, random\_state=42)

test\_label=train.target

labels=model.predict(test.data )

"""# Performance Metrics Evaluation"""

#creating confusion matrix and heat map

from sklearn.metrics import confusion\_matrix

mat=confusion\_matrix(test.target,labels)

sns.heatmap(mat.T,square=True,annot=True,fmt='d',cbar=False,xticklabels=train.target\_names,yticklabels=train.target\_names)

#plotting heatmap on confusion matrix

plt.xlabel('truelabel')

plt.ylabel('predicted label');

#accuracy score computation

metrics.accuracy\_score(test.target,labels)

#predicting category on new datasets

def cat\_select(s,train=train, model=model):

pred=model.predict([s])

return train.target\_names[pred[0]]

"""# Short Queries"""

cat\_select('cricket and football')

cat\_select('weather ')

cat\_select('global warming ')

cat\_select("The weather today will be hot and dry")

cat\_select('tech')

"""# Uppercase and lowercase testing"""

cat\_select('Sun is rising')

cat\_select('WORLD SPORTS DAY')

cat\_select('IPL gamE')

"""# sentence with Stopwords Querying"""

cat\_select('Climate change describes a change in the average conditions')

cat\_select('Hardware stores have sold out of water pumps and tarpaulins. ')

cat\_select('The players are the best in a team')

cat\_select('Football, cricket and hockey are all team sports. 6 The sports meeting was held in the stadium. ')

"""# Complicated sentences """

cat\_select('CUT ALL TYS John Fury calls on Tyson to SACK SugarHill Steward after Usyk fight collapse for ‘denying him like Judas denied Jesus')

cat\_select('The Nation IS PRIDE OF ITS sports team and their talent')

cat\_select('I wish we had sunny weather.')

"""# Long Queries"""

cat\_select('Elon Musk and Tech Leaders Urge Pause in AI ')

cat\_select('TThroughout talks the Gypsy King had appeared to be in training with Steward in the North West. ')

cat\_select('Cricket is more interesting than football evn though football adds more people to team')

cat\_select('Coventry 7 day weather forecast including weather warnings')

"""# Queries with punctuations"""

cat\_select('the weather JKLL;')

cat\_select('THE PUBLIC and hjkkll the tech protocoLSS')

cat\_select('THE climate and hjkkll the government protocoLSS')

cat\_select('Man of \*\*\*\*the match \*\*\*\*\* !1!')

cat\_select('....best >>>player')