**Name: - Harsh Patel**

**A20369913**

**CS 429 Project Report**

[**hpatel100@hawk.iit.edu**](mailto:hpatel100@hawk.iit.edu)

1. **Abstract: -**

* Finding a Top-k results from User defined query from mapped(user-defined) seeds crawler is the core of this project. A closely related problem refers to searching some specific questions on google, yahoo, Bing, etc. and get back response on Top-K query results that matches with millions of webpages or URLs. We study the local machine information retrieval system that based on scrapy crawling system, saved an inverted index of {word, Pages that includes word} into local machine pickles, and user-defined query in Flask based JSON processor format. We give a detailed analysis and show formally how all these three tasks are Information retrieval in general.

1. **Overview: -**

* Information retrieval (IR) system in [computing](https://en.wikipedia.org/wiki/Computing) and [information science](https://en.wikipedia.org/wiki/Information_science) is the process of obtaining in resources that are relevant to information need from a collection of resources like database, web crawling and information resources, and also can be metadata like described web data, or texts, sounds, or images from resources. Searches can be based on [full-text](https://en.wikipedia.org/wiki/Full-text_search) or other content-based indexing.
* We are using “Introduction to Information Retrieval” as a reference for building this information retrieval system.
* We are building some basic information retrieval system using scrapy and flask library. We analyzed how the scrapy based content crawler for downloading web documents in html format into local machine using Max Pages and Max depth of seed URL/Domain, a scikit-based learn based indexer for constructing an inverted index in pickle format into local machine, and Flask based processor for handling free text queries in JSON format and get Top-K results from html inverted index based on downloaded HTML pages.
* The proposed architecture is given below. ‘Loc’ is i.e. location in project directory.
* Scrapy-based crawler: - (Loc: ‘{project\_root\_directory}/spiders/mainspider.py’)
  + We have a user-interface which needs URL, Max Pages, and Max depth as a user-input that can map max depth and max pages to crawl URL seeds pages, using scrapy and saved downloaded HTML pages into local machine at ‘htmlFiles’ directory using renaming it using their domain name. For safe references, and test-cases, I am copying it to ‘htmlFilesToTest’ directory too.
* Scikit-Learn based Indexer: - (Loc: ‘{project\_root\_directory}/indexer/main.py’)
  + We then take all downloaded saved HTML files into consideration and parsed it with HTML parser. And, using block-sorting based indexing, I saved an inverted index of {word, Pages that includes word}as pickle files to the ‘pickleFiles’ directory. We find DF-IDF score and cosine similarity in this block too.
* Flask based Processor: - (Loc: ‘{project\_root\_directory}/processor/app.py’)
  + We then take all saved pickle files and merged/combined for the final one dictionary. I did that for inverted\_index, tf\_idf, idf pickle files. The flask app is running and identify queries are in valid JSON format or not, try to get query vector and then cosine similarity score. And then we can map Top-K based search result for queries.
* Basic Block-Diagram of the system: -

UI (O/P: Top-K query results)

Flask based Processor – JSON queries

Scikit-learn based Indexer

Scrapy-based WebCrawler

User-Interface (Input: URI)

Downloaded HTML files into local machine

Downloaded Pickle files into local machine

Detailed version: -

Diagram, engineering drawing

Description automatically generated

1. **Design: -**

Project Directories structure: -

1. cs429crawler -> **Root Directory**
   1. cs429crawler -> Scrapy based Crawler directory
      1. spiders
         1. \_\_init\_\_.py -> spider init class
         2. mainspider.py -> **Main python class of spider**
         3. test.py -> some test cases.
      2. \_\_init\_\_.py
      3. middlewares.py -> spider Middleware class
      4. pipelines.py -> Spider pipeline class
      5. settings.py -> Spider settings class
   2. htmlFiles -> **Spider HTML downloaded files**
   3. htmlFilesToTest -> Backup of htmlFiles directory.
   4. indexer -> Scikit learn based indexer
      1. \_\_init\_\_.py -> Indexer init class
      2. main.py -> **Index operation python class**
   5. pickleFiles -> **Saved pickle files from main.py**
   6. processor -> Flask based processor
      1. template -> Template class
         1. \_\_init\_\_.py -> Init class of template
         2. index.html -> html page of Flask
      2. \_\_init\_\_.py
      3. app.py -> **Main python class of Flask app.**
      4. config.py -> Configuration class
      5. logger.py -> logger class
   7. iitEdu.csv -> Use for HTML parsing (UTF-8)
   8. iitEdu\_1.csv -> Use for HTML parsing (UTF-8)
   9. scrapy.cfg -> scrapy config
2. As we have seen here, “cs429crawler” contains the main root directory. We run this program by using command-line using terminal input commands.
3. “cs429crawler/ cs429crawler” is the root directory of the scrapy based crawler, in which we run it by using *mainspider.py* class, and downloaded html files will be get saved into “cs429crawler/ htmlFiles” directory. Middleware.py, pipelines.py and settings.py are the scrapy architectural configuration files.
4. “cs429crawler/ indexer” is the root directory of the Scikit-Lean based Indexer, in which we run it by using *main.py* class, and downloaded inverted index directories pickle files will be get saved into “cs429crawler/ pickleFiles” directory. We simply use block-sorting based algorithm.
5. “cs429crawler/processor” is the root directory of the Flask based Processor, in which we run it by using *app.py (our main Flask app main class*), we try to post JSON queries from command-line input and get response as a query result, first it will validate the query, and then get the top-k results. index.html is template, and config.py is config class.
6. **Architecture: -**

* As we discussed, we have 3 components in this IR system.
* Right now, in Scrapy based Crawling in mainspider.py class, we include allowed domains are ['en.wikipedia.org', 'www.iit.edu', 'www.phdessay.com']. We can add any domain under it. For test purposes, I tested scrapy based crawling on base URL of ‘en.wikipedia.org’. The spider class name is “MainSpider”.
* The snippet of the code is given below. It also included the 'DEPTH\_LIMIT': '2'. We can also do it by passing it. 'DEPTH\_LIMIT' from command line or pre-defined in ‘cs429crawler/settings.py’.

**Text

Description automatically generated**

* Else you can set limit of 'DEPTH\_LIMIT' in ‘cs429crawler/settings.py’ as well like below.

Text

Description automatically generated

* The following snippet shows how every time we scrawl new URL checking “status code == 200”, old files get deleted and new files pops up into ‘htmlFiles’ directory.

Text

Description automatically generated

* We also set depth limit into spider Middleware.py class by using filter function and set MAX\_DEPTH as requested from response from requested fetched URL’s meta.

Text

Description automatically generated

* When we parse, user argument given webpage, we try to parse it by, using to identify href link for next depth pages via “link.xpath('.//@href').get()” function. We also try to get html webpage name from the URL link that we gotv from extraction of URL links and saved them into ‘htmlFiles’ directory.

Text

Description automatically generated

* As a result, All URL links we got from scrapy, downloaded as HTML pages into “htmlFiles” directory automatically deleting last saved files. (in “htmlFiles” directory”)
* Now, to start Scikit-learn based Indexer process, we get all the files ends with “.html” extension in the htmlFiles directory and parse with HTML parser encoding with “utf-8”.

Graphical user interface

Description automatically generated with medium confidence

* Then, we create a tokenized wordlist from each document and add them into list of html\_parsing\_list, index wise, by removing stop-words, in “indexer/main.py” file.

Text

Description automatically generated

* Then, we create the DF dictionary using html\_parsing\_list and tokenized\_words.

Text

Description automatically generated

* We the created inverted index dictionary and save them into pickle files in “pickleFiles”.
* e.g. like, using block-sorted index pickle files.
  + *save\_0.pkl, save\_1.pkl, save\_2.pkl*, etc.

A screenshot of a computer

Description automatically generated with medium confidence

* We then get IDF score from html\_parsing\_list and save them into in “pickleFiles”.

Text

Description automatically generated

* We the try to get TF-IDF score/weight representation, save it into tf\_idf\_dict dictionary.

Text

Description automatically generated

* We then using IDF score, TF-IDF score, we try to implement query vector score using idf\_dict dictionary and tokenization into queries.

Text

Description automatically generated

* After that, we get query-vector score, we get the cosine similarity score and even sorted similarity scores using the following snippet of code.

Text

Description automatically generated

**Result of Scikit-Learn based Indexer (TF-IDF/Weight, Cosine similarity): -**

Text

Description automatically generated

* We got the result of pickle files saved into “pickleFiles” directory. The snapshot is below.

Text

Description automatically generated

* Now, after that I work on Flask based processor. I have app.py as a main app in “processor” directory and config.py as a configuration file, and logger.py as debug logger file.
* The configuration of flask processor is in “processor/config.py” file. We defined log\_level, log\_format, application\_root, flask\_run\_mode, and web\_port in config.py file.

Graphical user interface, text

Description automatically generated

* I run flask.app by using app.run(debug.True) in if \_\_name\_\_ == '\_\_main\_\_': class, and app = Flask(\_\_name\_\_).
* We got multiple queries defined in JSON Post processing by given below snippet, in which we check weather the post request has JSON query object or not, and put all the queries into list.

Text

Description automatically generated

* Query example: - (I used command-line to use it). So, far we have only Flask Post request model.

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Binary Electronic\"}" "127.0.0.1:5000/"

response:

[

[

5,

0.7609628011757698

],

[

2,

0.6546462923610638

],

[

1,

0.07010823467774237

],

[

3,

0.0

],

[

4,

0.0

]

]

* We then using block-sorting algorithm, merged/combined together for the final one inverted index.

Text

Description automatically generated

* We then calculate TF-IDF, and then calculate the IDF score, TF-IDF score, we try to implement query vector score using idf\_dict dictionary and tokenization into queries. We run loop through query list as we might have multiple queries. The image is given below of how to calculate query vector.

Graphical user interface, text

Description automatically generated

* And, from then, we need to calculate similarity score, in this case, I chose to do cosine similarity. The code snippet is given below.

Text

Description automatically generated

And, the we get following results of Top-k: -

Text

Description automatically generated

1. **Operations: - (Steps to how to work on local machine)**

* Please extract the CS429\_Harsh\_Patel\_Project.zip and extract it.
* I am running it in PyCharm IDE PyCharm 2021.2.3 (Professional Edition), Build #PY-212.5457.59, built on October 19, 2021
* To check, Python 3.0+/sklearn 1.0+/Scrapy 2.0+/Flask 2.0+,

**Python: -**

* First check if python installed in your local machine or not using

Python --version

* If not, use the following command:

brew install python

* I have Python 3.9.12 in my local machine [https://www.python.org/downloads/release/python-3912/], released on March 23, 2022.
* Again, check python is installed or not by using following commands,

Python --version

**Scrapy: -**

* Scrapy requires Python 3.6+, either the CPython implementation (default) or the PyPy 7.2.0+ implementation.
* First check if scrapy installed in your local machine or not using,

scrapy version

* If not, use the following command:

pip install Scrapy

OR

conda install -c conda-forge scrapy

* I have Scrapy 2.6.1 in my local machine [https://www.python.org/downloads/release/python-3912/], released on March 23, 2022.
* Again, check scrapy is installed or not by using following commands,

scrapy version

**Flask: -**

* First check if flask installed in your local machine or not using

flask –version

* In my machine, I got flask version,

*Python 3.9.12*

*Flask 2.0.3*

*Werkzeug 2.0.3*

* If not, use the following command:

pip install Flask

* I have Python 3.9.12 in my local machine [https://www.python.org/downloads/release/python-3912/], released on March 23, 2022.
* Again, check python is installed or not by using following commands,

flask --version

***After checking these basic requirements,***

Check BeautifulSoup installed or not, and if not, use this command, I have beautifulsoup4 4.11.1

pip install beautifulsoup4

**How to run command in this project: -**

* + - 1. A Scrapy based Crawler for downloading web documents in html format, go to **project root directory ‘cs429crawler’** and run the following command.
      2. Please type or give URL as an input argument in command prompt.

scrapy crawl MainSpider -a url={please give your URL/domain name}

* + - 1. Now, the downloaded HTML URLs will get stored in ‘htmlFiles’ directory.
      2. Then, to run Scikit-Learn based Indexer, we use following command.
      3. **project root directory**: cs429crawler / **package:** indexer

python indexer/main.py

* + - 1. Then, we got saved pickle files into “picklefiles” directory.
      2. Now, to run Flask processor server, please use the following command.

python processor/app.py runserver

If you get the address already in use ERROR:

* Type the following command:

ps -fA | grep python

and, if you found out, “python processor/app.py runserver” process, kill that process using the port number.

Like example, given below.

ps -fA | grep python

501 ***58091*** 46777 0 3:13PM ttys007 0:00.21 python processor/app.py runserver

(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler % kill -9 ***58091***

(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler %

[1] + killed python processor/app.py runserver

And, try again, you will get following result.

(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler % python processor/app.py runserver

\* Serving Flask app 'app' (lazy loading)

\* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

\* Debug mode: on

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 740-214-834

* You will check on <http://127.0.0.1:5000/>
* **Now, To get query, please type following curl JSON Post command in terminal: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"{type your query} \"}" "127.0.0.1:5000/"

* You will get response result like, **ex., docs 5, 2, 1 are relevant; 3 and 4 are not relevant.**

127.0.0.1 - - [30/Apr/2022 16:23:04] "POST / HTTP/1.1" 200 -

[

[

5,

0.7609628011757698

],

[

2,

0.6546462923610638

],

[

1,

0.07010823467774237

],

[

3,

0.0

],

[

4,

0.0

]

]

1. **Conclusion: - (Steps to how to work on local machine)**

**Test case: -**

1. Go into project root directory, type the following command.

scrapy crawl MainSpider -a url=https://en.wikipedia.org/wiki/George\_Minaker

* will be downloaded 34 HTML documents into ‘htmlFiles’ into my local disk.

1. Now, after we got html pages, we now start indexing process.

python indexer/main.py

* will be downloaded 37 pickle files into ‘pickleFiles’ into local machines.

Result: - (TF-IDF score/weight representation, Cosine similarity)

Text

Description automatically generated

1. Now, after we got html pages, we now start indexing process.

python processor/app.py runserver

Result: - (Flask server started)

Graphical user interface, text

Description automatically generated

1. If we send query over in Flask JSON Post server, result would be, 1st query is given below,

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Oshawa Vancouver\"}" "127.0.0.1:5000/"

1st Query Result would be: - (In response in pycharm terminal and in terminal)

* NOTE: In Top-K results, it’s in sorting decreasing order, 0 means no similarity[irrelevant], positive means relevant that much of similarity in decreasing order, better the score comes first.

Text

Description automatically generated

Text

Description automatically generated

**2nd Query: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Oshawa\", \"query2\" : \"Vancouver\"}" "127.0.0.1:5000/"

2nd Query Result would be: - (In response in pycharm terminal and in terminal)

Text

Description automatically generated

Text

Description automatically generated

**3rd Query: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Mulroney\", \"query2\" : \"denouncing\"}" "127.0.0.1:5000/"

3rd Query Result would be: - (In response in pycharm terminal and in terminal)

Text

Description automatically generated

Text

Description automatically generated

**4th Query: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Progressive\", \"query2\" : \"Elections\"}" "127.0.0.1:5000/"

4th Query Result would be: - (In response in pycharm terminal and in terminal)

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

**5th Query: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Progressive Affiliation Conservative NDP\"}" "127.0.0.1:5000/"

5th Query Result would be: - (In response in pycharm terminal and in terminal)

Text

Description automatically generated

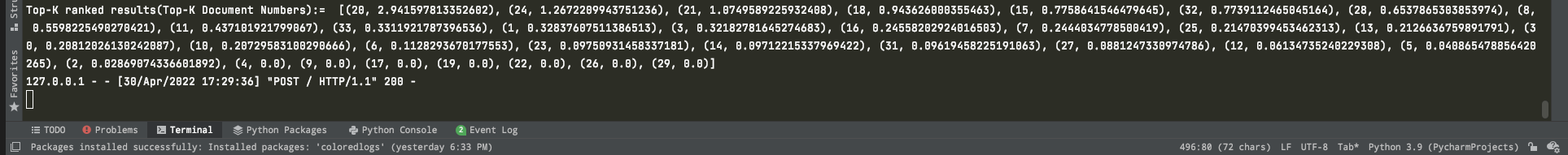
Text

Description automatically generated

**6th Query: -**

curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Despite losing percentage point terms vote share\"}" "127.0.0.1:5000/"

6th Query Result would be: - (In response in pycharm terminal and in terminal)



Text

Description automatically generated

**Success: - The system is almost working fine except following flaws.**

1. It will remove stop words from query even if present there. Top-K results has almost all accuracy. I got almost all the result relating big corpus too.

**Cautions/Flaws [Test needed to be taken care for future, while checking for large corpus]: -**

My system is not fault-proof. I have some bug to work on, pops on when I tested large corpus on last minute.

1. The major thing I faced today is, if I search one query for top-k results in flask server, and get result, and if I search another one, **I got Error like AttributeError: 'dict' object has no attribute 'extend', but if I think it’s my system design fault. If you try multiple queries JSON, please bear to restart flask server again to get the query result, and check if no server address already in use**.
2. If you run another URL, in case, please try to change,

query\_to\_check = "Oshawa Vancouver"

Change it to, words in new wordlist from new corpus from new URL.

query\_to\_check = "{words that are in new token list/You can search it from any words from new webpage that saved}"

1. **Data Sources: -**

* For Indexing, Processing -> I use primarily Dictionary data structure for Inverted Index for each, like, tokenized words dictionary, DF for each word, IDF score, TD-IDF score, query vector and cosine similarity. I almost use dictionary for indexing and query processing in Flask.
* For corpus data, I used user defined argument to give user-input url. I used url as, <https://en.wikipedia.org/wiki/George_Minaker>
* -a define user-define argument for url. Example is given below.

scrapy crawl MainSpider -a url=https://en.wikipedia.org/wiki/George\_Minaker

* I use BeautifulSoup package for html parser for locally saved HTML pages from tokenization of words from like ‘UTF-8’. The import package is given below.

from bs4 import BeautifulSoup

* BeautifulSoup Link: - <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

1. **Test Cases: -**

* Please check Test case with 6 different queries in conclusion part, on page 16.

1. **Source Code: -**

* Project directory structure is shown in Page 4.
* The source code is given below.

**FOR SCRAPY BASED CRAWLER: -**

* **Python File: cs429crawler/spiders/mainspider.py**

*import* scrapy  
*import* os  
*import* requests  
*import* logging  
  
*class* MainSpider(scrapy.Spider):  
 name = "MainSpider"  
 logging.getLogger("urllib3").setLevel(logging.WARNING)  
  
 custom\_settings = {  
 'DEPTH\_LIMIT': '2',  
 }  
  
 *def \_\_init\_\_*(self, *url*='', *\*\*kwargs*):  
 self.links=[]  
 self.allowed\_domains = ['en.wikipedia.org', 'www.iit.edu', 'www.phdessay.com']  
 self.start\_urls = [*url*] # py36  
 self.base\_url = 'en.wikipedia.org'  
 super(MainSpider, self).\_\_init\_\_(\*\**kwargs*) # python3  
  
 save\_path = 'htmlFiles'  
  
 *def* parse(self, *response*, *dest\_path* = *save\_path*, *save\_path*=*save\_path*):  
 init = 0  
 *for* link *in response*.xpath('//div/p/a'):  
 # if init == 2:  
 # break  
 *yield* {  
 "link": self.base\_url + link.xpath('.//@href').get()  
 }  
 print(self.base\_url + link.xpath('.//@href').get())  
 filePath = self.base\_url + link.xpath('.//@href').get()  
 words = ['tel', 'phone', 'mob', 'mailto', 'Special']  
 filePath\_upd = filePath.rsplit('/', 1)[-1] + '.html'  
 *for* word *in* words:  
 *if* word *in* filePath:  
 filePath = self.base\_url  
 print(requests.get('https://{}'.format(filePath)))  
 *if* requests.get('https://{}'.format(filePath)).status\_code == 200:  
 *for* f *in* os.listdir(*save\_path*):  
 *if* init == 0:  
 *if not* f.endswith(".html"):  
 *continue* os.remove(os.path.join(*save\_path*, f))  
 *if* os.path.join('htmlFiles', filePath\_upd):  
 *with* open(os.path.join('htmlFiles', filePath\_upd), 'w'):  
 *pass  
 with* open(os.path.join('htmlFiles', filePath\_upd), "wb") *as* file:  
 # response = get(self.base\_url + link.xpath('.//@href').get())  
 response = requests.get('https://{}'.format(filePath))  
 file.write(*response*.content)  
 # if requests.get('http://{}'.format(filePath)).status\_code == 200:  
 # for f in os.listdir(save\_path):  
 # if init == 0:  
 # if not f.endswith(".html"):  
 # continue  
 # os.remove(os.path.join(save\_path, f))  
 # if os.path.join('htmlFiles', filePath\_upd):  
 # with open(os.path.join('htmlFiles', filePath\_upd), 'w'):  
 # pass  
 # with open(os.path.join('htmlFiles', filePath\_upd), "wb") as file:  
 # # response = get(self.base\_url + link.xpath('.//@href').get())  
 # response = requests.get('http://{}'.format(filePath))  
 # file.write(response.content)  
 *else*:  
 *pass* init += 1

* **Python File: cs429crawler/middlewares.py**

# Define here the models for your spider middleware  
#  
# See documentation in:  
# https://docs.scrapy.org/en/latest/topics/spider-middleware.html  
  
*from* scrapy *import* signals  
*from* scrapy.http *import* Request  
*import* tldextract  
*import* logging  
  
logger = logging.getLogger(\_\_name\_\_)  
# useful for handling different item types with a single interface  
*from* itemadapter *import* is\_item, ItemAdapter  
  
  
*class* Cs429CrawlerSpiderMiddleware:  
 # Not all methods need to be defined. If a method is not defined,  
 # scrapy acts as if the spider middleware does not modify the  
 # passed objects.  
  
 *def \_\_init\_\_*(self, *maxdepth*, *stats*, *verbose\_stats*=*False*, *prio*=1):  
 self.maxdepth = *maxdepth* self.stats = *stats* self.verbose\_stats = *verbose\_stats* self.prio = *prio* @classmethod  
 *def* from\_crawler(cls, *crawler*):  
 # This method is used by Scrapy to create your spiders.  
  
 # s = cls()  
 # crawler.signals.connect(s.spider\_opened, signal=signals.spider\_opened)  
 # return s  
  
 settings = *crawler*.settings  
 maxdepth = settings.getint('DEPTH\_LIMIT')  
 verbose = settings.getbool('DEPTH\_STATS\_VERBOSE')  
 prio = settings.getint('DEPTH\_PRIORITY')  
 *return* cls(maxdepth, *crawler*.stats, verbose, prio)  
  
 *def* process\_spider\_input(self, response, spider):  
 # Called for each response that goes through the spider  
 # middleware and into the spider.  
  
 # Should return None or raise an exception.  
 *return None  
  
 def* process\_spider\_output(self, *response*, *result*, *spider*):  
 *def* \_filter(*request*):  
 *if* isinstance(*request*, Request):  
 depth = response.meta['depth'] + 1  
 *request*.meta['depth'] = depth  
 *if* self.prio:  
 *request*.priority -= depth \* self.prio  
 *if* self.maxdepth *and* depth > self.maxdepth:  
 logging.debug(  
 "Ignoring link (depth > %(maxdepth)d): %(requrl)s ",  
 {'maxdepth': self.maxdepth, 'requrl': *request*.url},  
 extra={'spider': spider}  
 )  
 *return False  
 else*:  
 *if* self.verbose\_stats:  
 self.stats.inc\_value(f'request\_depth\_count/{depth}',  
 spider=spider)  
 self.stats.max\_value('request\_depth\_max', depth,  
 spider=spider)  
 *return True* # base case (depth=0)  
 *if* 'depth' *not in response*.meta:  
 *response*.meta['depth'] = 0  
 *if* self.verbose\_stats:  
 self.stats.inc\_value('request\_depth\_count/0', spider=*spider*)  
  
 *return* (r *for* r *in result or* () *if* \_filter(r))  
  
 *def* process\_spider\_exception(self, *response*, *exception*, *spider*):  
 # Called when a spider or process\_spider\_input() method  
 # (from other spider middleware) raises an exception.  
  
 # Should return either None or an iterable of Request or item objects.  
 *pass  
  
 def* process\_start\_requests(self, *start\_requests*, spider):  
 # Called with the start requests of the spider, and works  
 # similarly to the process\_spider\_output() method, except  
 # that it doesn’t have a response associated.  
  
 # Must return only requests (not items).  
 *for* r *in start\_requests*:  
 *yield* r  
  
 *def* spider\_opened(self, *spider*):  
 *spider*.logger.info('Spider opened: %s' % *spider*.name)  
  
  
*class* Cs429CrawlerDownloaderMiddleware:  
 # Not all methods need to be defined. If a method is not defined,  
 # scrapy acts as if the downloader middleware does not modify the  
 # passed objects.  
  
 @classmethod  
 *def* from\_crawler(cls, *crawler*):  
 # This method is used by Scrapy to create your spiders.  
 s = cls()  
 *crawler*.signals.connect(s.spider\_opened, signal=signals.spider\_opened)  
 *return* s  
  
 *def* process\_request(self, request, spider):  
 # Called for each request that goes through the downloader  
 # middleware.  
  
 # Must either:  
 # - return None: continue processing this request  
 # - or return a Response object  
 # - or return a Request object  
 # - or raise IgnoreRequest: process\_exception() methods of  
 # installed downloader middleware will be called  
 *return None  
  
 def* process\_response(self, request, *response*, spider):  
 # Called with the response returned from the downloader.  
  
 # Must either;  
 # - return a Response object  
 # - return a Request object  
 # - or raise IgnoreRequest  
 *return response  
  
 def* process\_exception(self, *request*, *exception*, *spider*):  
 # Called when a download handler or a process\_request()  
 # (from other downloader middleware) raises an exception.  
  
 # Must either:  
 # - return None: continue processing this exception  
 # - return a Response object: stops process\_exception() chain  
 # - return a Request object: stops process\_exception() chain  
 *pass  
  
 def* spider\_opened(self, *spider*):  
 *spider*.logger.info('Spider opened: %s' % *spider*.name)

* **Python File: cs429crawler/settings.py**

# Scrapy settings for cs429crawler project  
#  
# For simplicity, this file contains only settings considered important or  
# commonly used. You can find more settings consulting the documentation:  
#  
# https://docs.scrapy.org/en/latest/topics/settings.html  
# https://docs.scrapy.org/en/latest/topics/downloader-middleware.html  
# https://docs.scrapy.org/en/latest/topics/spider-middleware.html  
  
BOT\_NAME = 'cs429crawler'  
  
SPIDER\_MODULES = ['cs429crawler.spiders']  
NEWSPIDER\_MODULE = 'cs429crawler.spiders'  
  
DEPTH\_LIMIT = 2  
SCHEDULER\_DISK\_QUEUE = 'scrapy.squeues.PickleFifoDiskQueue'  
SCHEDULER\_MEMORY\_QUEUE = 'scrapy.squeues.FifoMemoryQueue'  
  
SPIDER\_MIDDLEWARES = {'cs429crawler.middlewares.Cs429CrawlerSpiderMiddleware' : 0 }  
  
# FEED\_FORMAT="csv"  
# FEED\_URI="iitEdu.csv"  
  
# Crawl responsibly by identifying yourself (and your website) on the user-agent  
#USER\_AGENT = 'cs429crawler (+http://www.yourdomain.com)'  
  
# Obey robots.txt rules  
ROBOTSTXT\_OBEY = *True*# Configure maximum concurrent requests performed by Scrapy (default: 16)  
#CONCURRENT\_REQUESTS = 32  
  
# Configure a delay for requests for the same website (default: 0)  
# See https://docs.scrapy.org/en/latest/topics/settings.html#download-delay  
# See also autothrottle settings and docs  
#DOWNLOAD\_DELAY = 3  
# The download delay setting will honor only one of:  
#CONCURRENT\_REQUESTS\_PER\_DOMAIN = 16  
#CONCURRENT\_REQUESTS\_PER\_IP = 16  
  
# Disable cookies (enabled by default)  
#COOKIES\_ENABLED = False  
  
# Disable Telnet Console (enabled by default)  
#TELNETCONSOLE\_ENABLED = False  
  
# Override the default request headers:  
#DEFAULT\_REQUEST\_HEADERS = {  
# 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',  
# 'Accept-Language': 'en',  
#}  
  
# Enable or disable spider middlewares  
# See https://docs.scrapy.org/en/latest/topics/spider-middleware.html  
#SPIDER\_MIDDLEWARES = {  
# 'cs429crawler.middlewares.Cs429CrawlerSpiderMiddleware': 543,  
#}  
  
# Enable or disable downloader middlewares  
# See https://docs.scrapy.org/en/latest/topics/downloader-middleware.html  
#DOWNLOADER\_MIDDLEWARES = {  
# 'cs429crawler.middlewares.Cs429CrawlerDownloaderMiddleware': 543,  
#}  
  
# Enable or disable extensions  
# See https://docs.scrapy.org/en/latest/topics/extensions.html  
#EXTENSIONS = {  
# 'scrapy.extensions.telnet.TelnetConsole': None,  
#}  
  
# Configure item pipelines  
# See https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
#ITEM\_PIPELINES = {  
# 'cs429crawler.pipelines.Cs429CrawlerPipeline': 300,  
#}  
  
# Enable and configure the AutoThrottle extension (disabled by default)  
# See https://docs.scrapy.org/en/latest/topics/autothrottle.html  
#AUTOTHROTTLE\_ENABLED = True  
# The initial download delay  
#AUTOTHROTTLE\_START\_DELAY = 5  
# The maximum download delay to be set in case of high latencies  
#AUTOTHROTTLE\_MAX\_DELAY = 60  
# The average number of requests Scrapy should be sending in parallel to  
# each remote server  
#AUTOTHROTTLE\_TARGET\_CONCURRENCY = 1.0  
# Enable showing throttling stats for every response received:  
#AUTOTHROTTLE\_DEBUG = False  
  
# Enable and configure HTTP caching (disabled by default)  
# See https://docs.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings  
#HTTPCACHE\_ENABLED = True  
#HTTPCACHE\_EXPIRATION\_SECS = 0  
#HTTPCACHE\_DIR = 'httpcache'  
#HTTPCACHE\_IGNORE\_HTTP\_CODES = []  
#HTTPCACHE\_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'

* **Python File: cs429crawler/items.py**

# Define here the models for your scraped items  
#  
# See documentation in:  
# https://docs.scrapy.org/en/latest/topics/items.html  
  
*import* scrapy  
  
  
*class* Cs429CrawlerItem(scrapy.Item):  
 # define the fields for your item here like:  
 # name = scrapy.Field()  
 *pass*

**FOR SCIKIT-LEARN BASED INDEXER: -**

* **Python File: indexer/main.py**
* *from* bs4 *import* BeautifulSoup  
  *import* re  
  *import* os  
  *import* sys  
  *import* numpy *as* np  
  *import* pickle  
    
  html\_parsing\_list = list()  
  df\_list = list()  
  directory = os.fsencode('htmlFiles')  
  ROOT\_PATH = os.path.dirname(os.path.abspath('htmlFiles'))  
    
  *def* main():  
   init = 0  
   start = 0  
   dict = {}  
   idf\_dict = {}  
   tf\_idf\_dict = {}  
   tf\_idf\_dict\_opt\_ext = {}  
   query\_vector\_dict = {}  
   cosine\_similarity\_dict\_parent = {}  
   cosine\_similarity\_dict\_parent\_2 = {}  
   cosine\_similarity\_dict\_parent\_final = {}  
   cosine\_similarity\_dict\_parent\_term\_list = {}  
   # word = "Oklahoma"  
    
   # Tokenized-words and DF-list  
   *for* file *in* os.listdir(directory):  
   # if init == 1:  
   # break  
   *with* open('iitEdu.csv', 'w') *as* f:  
   f.truncate()  
   filename = os.fsdecode(file)  
   *if* filename.endswith(".html"):  
   filePath = ROOT\_PATH + "/htmlFiles/" + filename  
   # print(filePath)  
   HtmlFile = open(filePath, 'r', encoding='utf-8')  
   source\_code = HtmlFile.read()  
   soup = BeautifulSoup(source\_code, "html.parser")  
   *with* open('iitEdu.csv', 'w') *as* f:  
   *for* line *in* soup.text:  
   f.write(line)  
   # upd\_clear\_str = re.sub("[^\w]", " ", str(soup.text))  
   # wordList = re.split('\s+', upd\_clear\_str)  
   # print("length of wordList:= ", len(wordList))  
    
   wordList = re.sub("[^\w]", " ", str(soup.text)).split()  
   stops = ["", "-", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]  
   tokenized\_word = [i *for* i *in* wordList *if* str(i).lower() *not in* stops]  
   # print(tokenized\_word)  
   # print("length of tokenized\_word:= ", len(tokenized\_word))  
   html\_parsing\_list.append(tokenized\_word)  
    
   # DF - list implementation  
   DF = {}  
   upd\_DF = {}  
   upd\_upd\_DF = {}  
   *for* w *in* tokenized\_word:  
   *if* w *in* DF:  
   DF[w].append(w)  
   *else*:  
   DF[w] = [w]  
   *for* i *in* DF:  
   upd\_DF[i] = len(DF.get(i))  
   *for* i *in* DF:  
   upd\_upd\_DF[i] = upd\_DF.get(i) / len(tokenized\_word)  
   df\_list.append(upd\_upd\_DF)  
    
   # Create a dictionary for all the words.  
   check = str(soup.text)  
   # with open('iitEdu\_1.csv', 'w') as f:  
   # for line in check:  
   # f.write(line)  
   # print("html\_parsing\_list[init]:= ", html\_parsing\_list[init])  
   # if word in check:  
   # print(">> Oklahama word is present!")  
   *for* item *in* html\_parsing\_list[init]:  
   *if* item *in* check:  
   *if* item *not in* dict:  
   dict[item] = []  
   *if* item *in* dict *and* (init + 1) *not in* dict.get(item):  
   dict[item].append(init + 1)  
    
   # Convert Index pickling of dictionary index pages pickle files.  
   fileName = "save\_" + str(init) + ".pkl"  
   *if not* os.path.isfile(os.path.join('picklefiles', fileName)):  
   *with* open(os.path.join('picklefiles', fileName), 'wb') *as* f:  
   pickle.dump(dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
   *else*:  
   os.remove(os.path.join('picklefiles', fileName))  
   *with* open(os.path.join('picklefiles', fileName), 'wb') *as* f:  
   pickle.dump(dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
    
   init += 1  
    
    
   # if word in dict:  
   # print("The word is in the list!")  
   # print("dict score of term 'Binary':= ", dict["Binary"])  
   # print("dict score of term 'Binary':= ", dict["Oshawa"])  
    
   # print("df\_list:= ", df\_list[18])  
    
   # Pickle file - html\_parsing\_list list  
   *if not* os.path.isfile('picklefiles/html\_parsing\_list\_pkl.pkl'):  
   *with* open('picklefiles/html\_parsing\_list\_pkl.pkl', 'wb') *as* f:  
   pickle.dump(html\_parsing\_list, f, protocol=pickle.HIGHEST\_PROTOCOL)  
   *else*:  
   os.remove('picklefiles/html\_parsing\_list\_pkl.pkl')  
   *with* open('picklefiles/html\_parsing\_list\_pkl.pkl', 'wb') *as* f:  
   pickle.dump(html\_parsing\_list, f, protocol=pickle.HIGHEST\_PROTOCOL)  
    
   # IDF\_score logic  
   *for* key, values *in* dict.items():  
   idf\_score = np.log(len(html\_parsing\_list) / len(values))  
   *if* key *not in* idf\_dict:  
   idf\_dict[key] = list()  
   *for* value *in* values:  
   idf\_dict[key].append([value, idf\_score])  
   # print("idf\_dict score of term 'Binary':= ", idf\_dict["Binary"])  
   # print("idf\_dict score of term 'Binary':= ", idf\_dict["Oshawa"])  
    
   # Pickle file - IDF  
   *if not* os.path.isfile('picklefiles/idf.pkl'):  
   *with* open('picklefiles/idf.pkl', 'wb') *as* f:  
   pickle.dump(idf\_dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
   *else*:  
   os.remove('picklefiles/idf.pkl')  
   *with* open('picklefiles/idf.pkl', 'wb') *as* f:  
   pickle.dump(idf\_dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
    
   # Find TF-IDF score  
   init\_token = 0  
   *for* token\_item *in* range(len(html\_parsing\_list)):  
   # if init\_token == 1:  
   # break  
   *for* item *in* df\_list[token\_item]:  
   *if* item *not in* tf\_idf\_dict:  
   tf\_idf\_dict[item] = {}  
   *if* item *in* tf\_idf\_dict:  
   tf\_idf\_dict\_opt = {}  
   doc\_ids = [docs[0] *for* docs *in* idf\_dict[item]]  
   doc\_ids\_idf\_score = [docs[1] *for* docs *in* idf\_dict[item]]  
   # print("item:= ", item)  
   # print("doc\_ids:= ", doc\_ids)  
   # print("doc\_ids\_idf\_score:= ", doc\_ids\_idf\_score)  
   count = 0  
   *for* number *in* doc\_ids:  
   *if* number *not in* tf\_idf\_dict\_opt:  
   tf\_idf\_dict\_opt[number] = []  
   *if* number *not in* tf\_idf\_dict\_opt\_ext:  
   tf\_idf\_dict\_opt\_ext[number] = []  
   *if* number *in* tf\_idf\_dict\_opt:  
   # print("doc\_ids\_idf\_score[count]:= ", doc\_ids\_idf\_score[count])  
   # print("df\_list[number - 1][item]:= ", df\_list[number - 1][item])  
   tf\_idf\_dict\_opt[number].append(doc\_ids\_idf\_score[count] \* df\_list[number - 1][item])  
   tf\_idf\_dict\_opt\_ext[number].append(doc\_ids\_idf\_score[count] \* df\_list[number - 1][item])  
   count = count + 1  
   tf\_idf\_dict[item] = tf\_idf\_dict\_opt  
   init\_token += 1  
   #elementsArticles  
   # print("tf\_idf\_dict\_opt\_ext:= ", tf\_idf\_dict\_opt\_ext)  
   # print("tf\_idf\_dict score of term 'Binary':= ", tf\_idf\_dict["Binary"])  
   # print("tf\_idf\_dict score of term 'Oshawa':= ", tf\_idf\_dict["Oshawa"])  
   # print("tf\_idf\_dict score of term 'Vancouver':= ", tf\_idf\_dict["Vancouver"])  
   # print("tf\_idf\_dict\_opt\_ext score:= ", len(tf\_idf\_dict\_opt\_ext))  
    
   # Pickle file - TF/IDF  
   *if not* os.path.isfile('picklefiles/td\_idf.pkl'):  
   *with* open('picklefiles/td\_idf.pkl', 'wb') *as* f:  
   pickle.dump(tf\_idf\_dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
   *else*:  
   os.remove('picklefiles/td\_idf.pkl')  
   *with* open('picklefiles/td\_idf.pkl', 'wb') *as* f:  
   pickle.dump(tf\_idf\_dict, f, protocol=pickle.HIGHEST\_PROTOCOL)  
    
   # Pickle file - TF\_IDF\_OPT  
   *if not* os.path.isfile('picklefiles/tf\_idf\_opt.pkl'):  
   *with* open('picklefiles/tf\_idf\_opt.pkl', 'wb') *as* f:  
   pickle.dump(tf\_idf\_dict\_opt\_ext, f, protocol=pickle.HIGHEST\_PROTOCOL)  
   *else*:  
   os.remove('picklefiles/tf\_idf\_opt.pkl')  
   *with* open('picklefiles/tf\_idf\_opt.pkl', 'wb') *as* f:  
   pickle.dump(tf\_idf\_dict\_opt\_ext, f, protocol=pickle.HIGHEST\_PROTOCOL)  
    
   # The vector should be a dict with query terms as keys and IDF scores as values.  
   query\_to\_check = "Oshawa Vancouver"  
   stops = ["", "-", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself",  
   "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself",  
   "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these",  
   "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do",  
   "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while",  
   "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before",  
   "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again",  
   "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each",  
   "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than",  
   "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]  
   data = re.sub('[^\w]|\_', ' ', query\_to\_check) # only keep numbers and letters and spaces  
   # data = data.lower()  
   data = re.sub(r'[^\x00-\x7f]', r'', data) # remove non ascii texts  
   tokens = [data\_upt *for* data\_upt *in* data.split(' ') *if* data\_upt] # split the words and remove empty words  
   tokens = [word *for* word *in* tokens *if* word.lower() *not in* stops]  
   *for* token *in* tokens:  
   doc\_ids\_idf\_score = [idf\_dict[token][0][1]]  
   query\_vector\_dict[token] = doc\_ids\_idf\_score  
   print("sample Query vector score:= ", query\_vector\_dict)  
    
   # Cosine similarity  
   # The result of the dot product between the query vector and the documents (per term) need to be converted into a final sorted list of pairs containing the document identifier and score  
   # The sorting is based on highest score to lowest.  
   terms = [docs *for* docs *in* query\_vector\_dict]  
   terms\_idf\_val = [query\_vector\_dict.get(docs)[0] *for* docs *in* query\_vector\_dict]  
   *for* term\_num *in* range(len(terms)):  
   doc\_length = []  
   cosine\_similarity\_dict\_clone\_res = {}  
   dict\_keys = list(tf\_idf\_dict[terms[term\_num]].keys())  
   cosine\_similarity\_dict = {}  
   wt = terms\_idf\_val[term\_num]  
   *for* idx *in* range(len(html\_parsing\_list)):  
   *if* idx *not in* dict\_keys:  
   cosine\_similarity\_dict[idx] = 0  
   *else*:  
   cosine\_similarity\_dict[idx] = terms\_idf\_val[0] \* tf\_idf\_dict[terms[term\_num]].get(idx)[0]  
   doc\_length.append(tf\_idf\_dict\_opt\_ext.get(idx)[0])  
   doc\_square = [i \*\* 2 *for* i *in* doc\_length]  
   doc\_length\_sqrt\_val = pow(sum(doc\_square), 0.5)  
   *for* key, values *in* cosine\_similarity\_dict.items():  
   cosine\_similarity\_dict\_clone\_res[key] = cosine\_similarity\_dict.get(key) / doc\_length\_sqrt\_val  
   *for* key, values *in* cosine\_similarity\_dict\_clone\_res.items():  
   *if* key *not in* cosine\_similarity\_dict\_parent *and* key != 0:  
   cosine\_similarity\_dict\_parent[key] = []  
   *if* key *in* cosine\_similarity\_dict\_parent:  
   cosine\_similarity\_dict\_parent[key].append(values)  
   *for* key, values *in* cosine\_similarity\_dict\_parent.items():  
   *if* key *not in* cosine\_similarity\_dict\_parent:  
   cosine\_similarity\_dict\_parent\_2[key] = 0  
   *if* key *in* cosine\_similarity\_dict\_parent:  
   cosine\_similarity\_dict\_parent\_2[key] = sum(cosine\_similarity\_dict\_parent.get(key))  
   cosine\_similarity\_dict\_parent\_term\_list[terms[term\_num]] = cosine\_similarity\_dict\_clone\_res  
   print("cosine\_similarity\_dict\_parent\_term\_list:= ", cosine\_similarity\_dict\_parent\_term\_list)  
   cosine\_similarity\_dict\_parent\_final = sorted(cosine\_similarity\_dict\_parent\_2.items(), key=*lambda t*: t[1],  
   reverse=*True*)  
    
   # "document identifier, score" for query and document pages for combine query vector and document vector.  
   print("Sorted cosine\_similarity\_dict\_parent\_final:= ", cosine\_similarity\_dict\_parent\_final)  
    
    
  *if* \_\_name\_\_ == '\_\_main\_\_':  
   main()

**FOR FLASK BASED PROCESSOR: -**

* **Python File: processor/config.py**

*import* os  
  
LOG\_LEVEL = 'DEBUG'  
LOG\_FORMAT = '%(asctime)s %(levelname)s : %(message)s'  
APPLICATION\_ROOT = '/cs429crawler'  
FLASK\_RUN\_MODE = os.environ.get('MODE') *or* 'PROD'  
WEB\_PORT = os.environ.get('PORT') *or* 5000

* **Python File: processor/logger.py**

*import* coloredlogs  
*import* logging  
  
*from* config *import* LOG\_LEVEL, LOG\_FORMAT  
  
# Create a logger object.  
logger = logging.getLogger(\_\_name\_\_)  
  
# inject color log  
coloredlogs.install(level=LOG\_LEVEL, fmt=LOG\_FORMAT)

* **Python File: processor/app.py**

*from* flask *import* Flask, request, jsonify, make\_response  
*import* pickle  
*import* os  
*import* re  
*from* pathlib *import* Path  
  
app = Flask(\_\_name\_\_)  
  
# TF-IDF dict objects <- To use pickle files  
main\_res\_dict\_tf\_idf = {}  
tf\_idf\_dict = {}  
dict\_new\_tf\_idf = {}  
  
# IDF dict objects <- To use pickle files  
idf\_dict = {}  
dict\_new\_idf = {}  
  
# TF\_IDF\_OPT dict objects <- To use pickle files  
main\_res\_dict\_tf\_idf\_opt = {}  
tf\_idf\_dict\_opt\_ext = {}  
dict\_new\_tf\_idf\_opt = {}  
  
# Query vector objects <- To use pickle files  
query\_vector\_dict = {}  
dict\_new\_inverted\_index = {}  
main\_res\_dict\_inverted\_index = {}  
  
# Cosine similarity objects -> To get similarity score.  
cosine\_similarity\_dict\_parent = {}  
cosine\_similarity\_dict\_parent\_2 = {}  
cosine\_similarity\_dict\_parent\_final = {}  
cosine\_similarity\_dict\_parent\_term\_list = {}  
  
# html\_parsing\_list object <- To use pickle files  
html\_parsing\_list = list()  
  
@app.route('/', methods=['POST'])  
*def* process\_json():  
 *global* cosine\_similarity\_dict\_parent\_final  
 content\_type = request.headers.get('Content-Type')  
 *if* (content\_type == 'application/json'):  
 query\_list = list()  
 json = request.json  
 # query\_1 = json['query1']  
 # print("query\_1:= ", query\_1)  
 *for* idx *in* range(len(json)):  
 init = idx + 1  
 queryName = "query" + str(init)  
 query\_list.append(json[queryName])  
 print("Query list(from Json format):= ", query\_list)  
  
 # Using block-sorting algorithm, merged/combined together for the final one inverted index.  
 myDir = Path('picklefiles/')  
 fileNames = [file.name *for* file *in* myDir.iterdir() *if* file.name.startswith('save')]  
 *for* file *in* fileNames:  
 *with* open(os.path.join(myDir, file), 'rb') *as* fileName:  
 dict\_new\_inverted\_index = pickle.load(fileName)  
 *for* k, v *in* dict\_new\_inverted\_index.items():  
 *try*:  
 main\_res\_dict\_inverted\_index[k].extend(v)  
 *except* KeyError:  
 main\_res\_dict\_inverted\_index[k] = v  
 res\_main\_res\_dict = {key: list(set(value)) *for* key, value *in* main\_res\_dict\_inverted\_index.items()}  
 print("Invrted index length: = ", len(res\_main\_res\_dict))  
  
 # TF-IDF file  
 *if* os.path.isfile('picklefiles/html\_parsing\_list\_pkl.pkl'):  
 *with* open('picklefiles/html\_parsing\_list\_pkl.pkl', 'rb') *as* fileName:  
 html\_parsing\_list = pickle.load(fileName)  
  
 # TF-IDF file to tf\_idf\_dict  
 *if* os.path.isfile('picklefiles/td\_idf.pkl'):  
 *with* open('picklefiles/td\_idf.pkl', 'rb') *as* fileName:  
 dict\_new\_tf\_idf = pickle.load(fileName)  
 *for* k, v *in* dict\_new\_tf\_idf.items():  
 *try*:  
 tf\_idf\_dict[k].extend(v)  
 *except* KeyError:  
 tf\_idf\_dict[k] = v  
 # tf\_idf\_dict = {key: list(set(value)) for key, value in main\_res\_dict\_tf\_idf.items()}  
  
 # IDF file to idf\_dict  
 *if* os.path.isfile('picklefiles/idf.pkl'):  
 *with* open('picklefiles/idf.pkl', 'rb') *as* fileName:  
 dict\_new\_idf = pickle.load(fileName)  
 *for* k, v *in* dict\_new\_idf.items():  
 *try*:  
 idf\_dict[k].extend(v)  
 *except* KeyError:  
 idf\_dict[k] = v  
 #  
 # # TF\_IDF\_OPT dict to tf\_idf\_dict\_opt\_ext  
 *if* os.path.isfile('picklefiles/tf\_idf\_opt.pkl'):  
 *with* open('picklefiles/tf\_idf\_opt.pkl', 'rb') *as* fileName:  
 dict\_new\_tf\_idf\_opt = pickle.load(fileName)  
 *for* k, v *in* dict\_new\_tf\_idf\_opt.items():  
 *try*:  
 tf\_idf\_dict\_opt\_ext[k].extend(v)  
 *except* KeyError:  
 tf\_idf\_dict\_opt\_ext[k] = v  
 #  
 # # Query vector and similarity  
 *for* idx *in* range(len(query\_list)):  
 stops = ["", "-", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours",  
 "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its",  
 "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that",  
 "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having",  
 "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until",  
 "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during",  
 "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under",  
 "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both",  
 "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so",  
 "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]  
 query\_to\_check = query\_list[idx]  
 data = re.sub('[^\w]|\_', ' ', query\_to\_check) # only keep numbers and letters and spaces  
 # data = data.lower()  
 data = re.sub(r'[^\x00-\x7f]', r'', data) # remove non ascii texts  
 tokens = [data\_upt *for* data\_upt *in* data.split(' ') *if* data\_upt] # split the words and remove empty words  
 tokens = [word *for* word *in* tokens *if* word.lower() *not in* stops]  
 *for* token *in* tokens:  
 doc\_ids\_idf\_score = [idf\_dict[token][0][1]]  
 query\_vector\_dict[token] = doc\_ids\_idf\_score  
 #  
 # Cosine similarity  
 # The result of the dot product between the query vector and the documents (per term) need to be converted into a final sorted list of pairs containing the document identifier and score  
 # The sorting is based on highest score to lowest.  
 terms = [docs *for* docs *in* query\_vector\_dict]  
 terms\_idf\_val = [query\_vector\_dict.get(docs)[0] *for* docs *in* query\_vector\_dict]  
 *for* term\_num *in* range(len(terms)):  
 doc\_length = []  
 cosine\_similarity\_dict\_clone\_res = {}  
 dict\_keys = list(tf\_idf\_dict[terms[term\_num]].keys())  
 cosine\_similarity\_dict = {}  
 *for* idx *in* range(len(html\_parsing\_list)):  
 *if* idx *not in* dict\_keys:  
 cosine\_similarity\_dict[idx] = 0  
 *else*:  
 cosine\_similarity\_dict[idx] = terms\_idf\_val[0] \* tf\_idf\_dict[terms[term\_num]].get(idx)[0]  
 doc\_length.append(tf\_idf\_dict\_opt\_ext.get(idx)[0])  
 doc\_square = [i \*\* 2 *for* i *in* doc\_length]  
 doc\_length\_sqrt\_val = pow(sum(doc\_square), 0.5)  
 *for* key, values *in* cosine\_similarity\_dict.items():  
 cosine\_similarity\_dict\_clone\_res[key] = cosine\_similarity\_dict.get(key) / doc\_length\_sqrt\_val  
 *for* key, values *in* cosine\_similarity\_dict\_clone\_res.items():  
 *if* key *not in* cosine\_similarity\_dict\_parent *and* key != 0:  
 cosine\_similarity\_dict\_parent[key] = []  
 *if* key *in* cosine\_similarity\_dict\_parent:  
 cosine\_similarity\_dict\_parent[key].append(values)  
 *for* key, values *in* cosine\_similarity\_dict\_parent.items():  
 *if* key *not in* cosine\_similarity\_dict\_parent:  
 cosine\_similarity\_dict\_parent\_2[key] = 0  
 *if* key *in* cosine\_similarity\_dict\_parent:  
 cosine\_similarity\_dict\_parent\_2[key] = sum(cosine\_similarity\_dict\_parent.get(key))  
 cosine\_similarity\_dict\_parent\_term\_list[terms[term\_num]] = cosine\_similarity\_dict\_clone\_res  
 print("\n\n\nTop-K ranked results(Query word -> similarity result):= ",cosine\_similarity\_dict\_parent\_term\_list)  
 cosine\_similarity\_dict\_parent\_final = sorted(cosine\_similarity\_dict\_parent\_2.items(), key=*lambda t*: t[1],  
 reverse=*True*)  
 print("Top-K ranked results(Top-K Document Numbers):= ", cosine\_similarity\_dict\_parent\_final)  
 # return json  
 jsonify("The result is given below (HTML page saved and their K-top results using similarity")  
 *return* jsonify(cosine\_similarity\_dict\_parent\_final), 200  
 *else*:  
 *return* 'Content-Type not supported!'  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=*True*)

**Packages installed in my machine: - (In the case if you need)**

|  |  |  |
| --- | --- | --- |
| appdirs | 1.4.4 | 1.4.4 |
| attrs | 21.4.0 | 21.4.0 |
| automat | 20.2.0 | 20.2.0 |
| bcrypt | 3.2.0 | 3.2.0 |
| beautifulsoup4 | 4.11.1 | 4.11.1 |
| blas | 1.0 | 1.0 |
| bottleneck | 1.3.4 | 1.3.4 |
| brotlipy | 0.7.0 | 0.7.0 |
| ca-certificates | 2022.3.29 | 2022.3.29 |
| certifi | 2021.10.8 | 2021.10.8 |
| cffi | 1.15.0 | 1.15.0 |
| charset-normalizer | 2.0.4 | 2.0.4 |
| click | 8.1.2 | 8.0.4 |
| colorama | 0.4.4 | 0.4.4 |
| coloredlogs | 15.0.1 | 15.0.1 |
| constantly | 15.1.0 | 15.1.0 |
| cryptography | 36.0.0 | 36.0.0 |
| cssselect | 1.1.0 | 1.1.0 |
| dataclasses | 0.8 | 0.8 |
| dm-tree | 0.1.7 | 0.1.5 |
| filelock | 3.6.0 | 3.6.0 |
| flask | 2.0.3 | 2.0.3 |
| greenlet | 1.1.1 | 1.1.1 |
| humanfriendly | 10.0 | 10.0 |
| hyperlink | 21.0.0 | 21.0.0 |
| icu | 58.2 | 68.1 |
| idna | 3.3 | 3.3 |
| incremental | 21.3.0 | 21.3.0 |
| intel-openmp | 2021.4.0 | 2022.0.0 |
| itemadapter | 0.3.0 | 0.3.0 |
| itemloaders | 1.0.4 | 1.0.4 |
| itsdangerous | 2.0.1 | 2.0.1 |
| jinja2 | 3.0.3 | 3.0.3 |
| jmespath | 0.10.0 | 0.10.0 |
| joblib | 1.1.0 | 1.1.0 |
| libcxx | 12.0.0 | 12.0.0 |
| libffi | 3.3 | 3.4.2 |
| libiconv | 1.16 | 1.16 |
| libxml2 | 2.9.12 | 2.9.12 |
| libxslt | 1.1.34 | 1.1.34 |
| lxml | 4.8.0 | 4.8.0 |
| markupsafe | 2.0.1 | 2.0.1 |
| mkl | 2021.4.0 | 2022.0.0 |
| mkl-service | 2.4.0 | 2.4.0 |
| mkl\_fft | 1.3.1 | 1.3.1 |
| mkl\_random | 1.2.2 | 1.2.2 |
| ncurses | 6.3 | 6.3 |
| nltk | 3.7 | 3.7 |
| numexpr | 2.8.1 | 2.8.1 |
| numpy | 1.21.5 | 1.21.5 |
| numpy-base | 1.21.5 | 1.21.5 |
| openssl | 1.1.1n | 1.1.1n |
| packaging | 21.3 | 21.3 |
| pandas | 1.4.2 | 1.4.2 |
| parsel | 1.6.0 | 1.6.0 |
| pillow | 9.1.0 | 9.0.1 |
| pip | 21.2.4 | 21.2.4 |
| protego | 0.1.16 | 0.1.16 |
| pyasn1 | 0.4.8 | 0.4.8 |
| pyasn1-modules | 0.2.8 | 0.2.8 |
| pycparser | 2.21 | 2.21 |
| pydispatcher | 2.0.5 | 2.0.5 |
| pyhamcrest | 2.0.2 | 2.0.2 |
| pyopenssl | 22.0.0 | 22.0.0 |
| pyparsing | 3.0.4 | 3.0.4 |
| pysocks | 1.7.1 | 1.7.1 |
| python | 3.9.12 | 3.10.4 |
| python-dateutil | 2.8.2 | 2.8.2 |
| pytz | 2021.3 | 2021.3 |
| queuelib | 1.5.0 | 1.5.0 |
| readline | 8.1.2 | 8.1.2 |
| regex | 2022.3.15 | 2022.3.15 |
| requests | 2.27.1 | 2.27.1 |
| requests-file | 1.5.1 | 1.5.1 |
| scrapy | 2.6.1 | 2.6.1 |
| service\_identity | 18.1.0 | 18.1.0 |
| setuptools | 61.2.0 | 61.2.0 |
| six | 1.16.0 | 1.16.0 |
| soupsieve | 2.3.1 | 2.3.1 |
| sqlalchemy | 1.4.32 | 1.4.32 |
| sqlite | 3.38.2 | 3.38.2 |
| svgwrite | 1.4.2 |  |
| tk | 8.6.11 | 8.6.11 |
| tldextract | 3.2.0 | 3.2.0 |
| tqdm | 4.64.0 | 4.64.0 |
| tree | 0.2.4 |  |
| twisted | 22.2.0 | 22.2.0 |
| typing\_extensions | 4.1.1 | 4.1.1 |
| tzdata | 2022a | 2022a |
| urllib3 | 1.26.9 | 1.26.9 |
| w3lib | 1.21.0 | 1.21.0 |
| werkzeug | 2.0.3 | 2.0.3 |
| wheel | 0.37.1 | 0.37.1 |
| xz | 5.2.5 | 5.2.5 |
| zlib | 1.2.12 | 1.2.12 |
| zope | 1.0 | 1.0 |
| zope.interface | 5.4.0 | 5.4.0 |

1. **Bibliography: -**
2. [Christopher D. Manning](http://nlp.stanford.edu/~manning/), [Prabhakar Raghavan](http://theory.stanford.edu/~pragh/) and [Hinrich Schütze](http://www.cis.uni-muenchen.de/personen/professoren/schuetze/), 2008 Introduction to Information Retrieval, Cambridge University Press.
3. Donald Metzler, Trevor Strohman, and W. Bruce Croft, 2010, Search Engines Information Retrieval in Practice, Pearson Education, Inc
4. <https://en.wikipedia.org/wiki/Information_retrieval>