Name: - Harsh Patel

A20369913

CS 429 Project Report

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.

2. Overview: -

⇒ Information retrieval (IR) system in computing and 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 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.

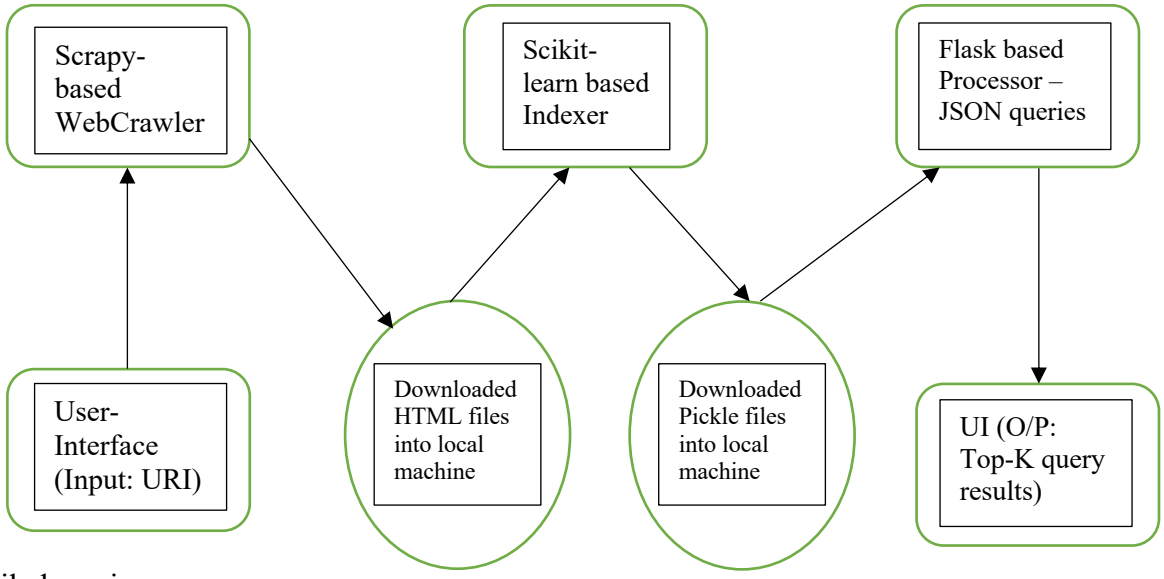
⇒ Scikit-Learn based Indexer: - (Loc: '{project_root_directory}/indexer/main.py')

- 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.

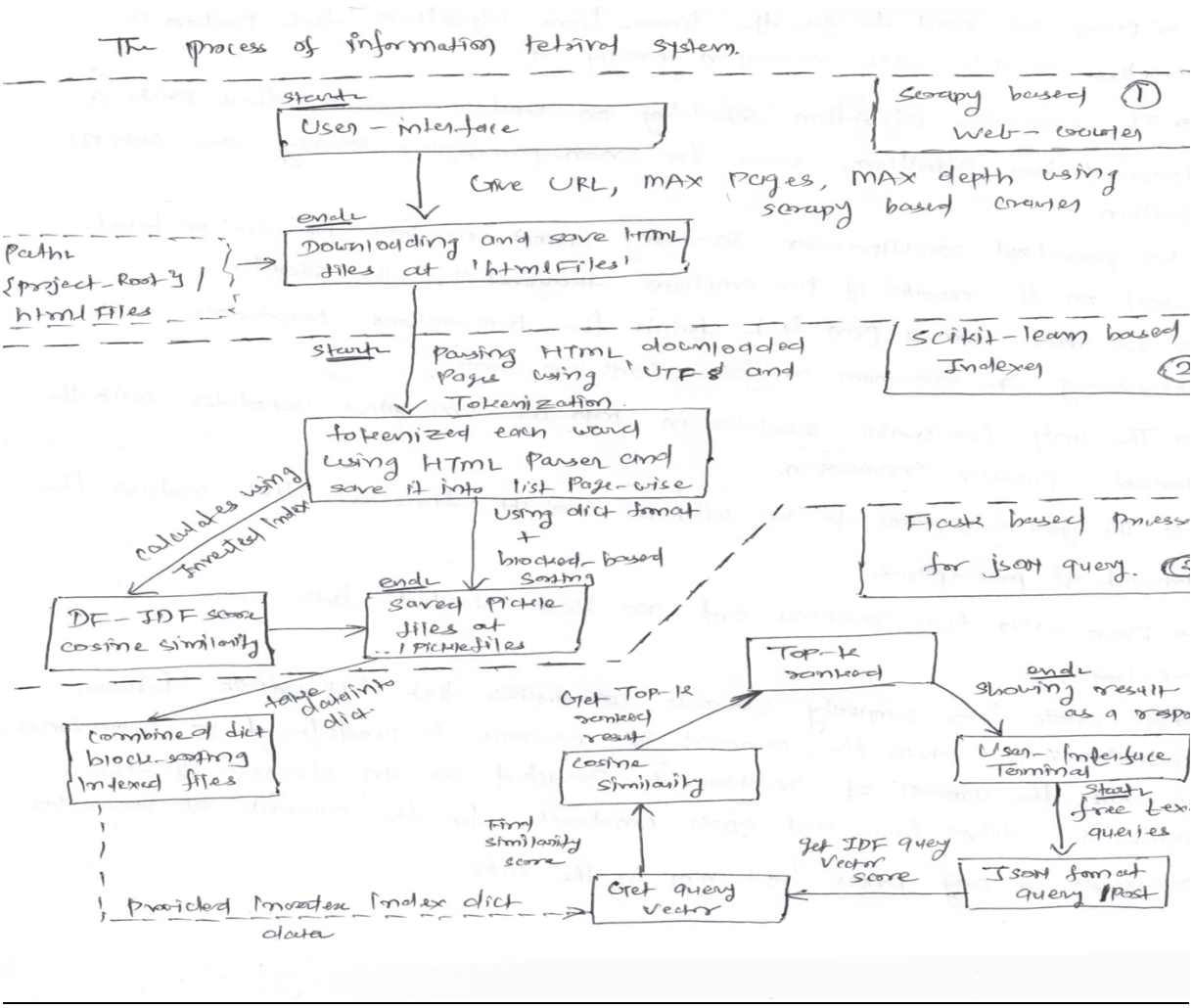
⇒ Flask based Processor: - (Loc: '{project_root_directory}/processor/app.py')

- 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: -



Detailed version: -



3. Design: -

Project Directories structure: -

1. cs429crawler -> **Root Directory**
    a. cs429crawler -> Scrapy based Crawler directory
        i. spiders
1. __init__.py -> spider init class
2. mainspider.py -> **Main python class of spider**
3. test.py -> some test cases.
        ii. __init__.py
        iii. middlewares.py -> spider Middleware class
        iv. pipelines.py -> Spider pipeline class
        v. settings.py -> Spider settings class
    b. htmlFiles -> **Spider HTML downloaded files**
    c. htmlFilesToTest -> Backup of htmlFiles directory.
    d. indexer -> Scikit learn based indexer
        i. __init__.py -> Indexer init class
        ii. main.py -> **Index operation python class**
    e. pickleFiles -> **Saved pickle files from main.py**
    f. processor -> Flask based processor
        i. template -> Template class
1. __init__.py -> Init class of template
2. index.html -> html page of Flask
        ii. __init__.py
        iii. app.py -> **Main python class of Flask app.**
        iv. config.py -> Configuration class
        v. logger.py -> logger class
    g. iitEdu.csv -> Use for HTML parsing (UTF-8)
    h. iitEdu_1.csv -> Use for HTML parsing (UTF-8)
    i. 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.

4 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'.

```
    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
```

⇒ Else you can set limit of 'DEPTH_LIMIT' in 'cs429crawler/settings.py' as well like below.

```
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
```

⇒ 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.

```
            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)
```

⇒ 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.

```
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))
```

⇒ 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.

```
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)))
```

⇒ 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".

```
34          if filename.endswith(".html"):
35              filePath = ROOT_PATH + "/htmlFiles/" + filename
36              # print(filePath)
37              HtmlFile = open(filePath, 'r', encoding='utf-8')
38              source_code = HtmlFile.read()
39              soup = BeautifulSoup(source_code, "html.parser")
```

⇒ 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.

```
46
47              wordList = re.sub("[^\w]", " ",  str(soup.text)).split()
48              stops = ["", "-", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "hi
49              tokenized_word = [i for i in wordList if str(i).lower() not in stops]
50              # print(tokenized_word)
51              # print("length of tokenized_word:= ", len(tokenized_word))
52              html_parsing_list.append(tokenized_word)
53
```

⇒ Then, we create the DF dictionary using html_parsing_list and tokenized_words.

```
55              DF = {}
56              upd_DF = {}
57              upd_upd_DF = {}
58              for w in tokenized_word:
59                  if w in DF:
60                      DF[w].append(w)
61                  else:
62                      DF[w] = [w]
63              for i in DF:
64                  upd_DF[i] = len(DF.get(i))
65              for i in DF:
66                  upd_upd_DF[i] = upd_DF.get(i) / len(tokenized_word)
67              df_list.append(upd_upd_DF)
68
```

⇒ 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.

```
76              # print(" >> Unigram word is present! ")
77              for item in html_parsing_list[init]:
78                  if item in check:
79                      if item not in dict:
80                          dict[item] = []
81                      if item in dict and (init + 1) not in dict.get(item):
82                          dict[item].append(init + 1)
83
84              # Convert Index pickling of dictionary index pages pickle files.
85              fileName = "save_" + str(init) + ".pkl"
86              if not os.path.isfile(os.path.join('picklefiles', fileName)):
87                  with open(os.path.join('picklefiles', fileName), 'wb') as f:
88                      pickle.dump(dict, f, protocol=pickle.HIGHEST_PROTOCOL)
89              else:
90                  os.remove(os.path.join('picklefiles', fileName))
91                  with open(os.path.join('picklefiles', fileName), 'wb') as f:
92                      pickle.dump(html_parsing_list, f, protocol=pickle.HIGHEST_PROTOCOL)
93
94          init += 1
```

⇒ We then get IDF score from html_parsing_list and save them into in “pickleFiles”.

```
    # 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:= ", idf_dict["Binary"])

    # 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)
```

⇒ We the try to get TF-IDF score/weight representation, save it into tf_idf_dict dictionary.

```
    # 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["Binary"])
    print("tf_idf_dict_opt_ext:= ", len(tf_idf_dict_opt_ext))
```

⇒ We then using IDF score, TF-IDF score, we try to implement query vector score using idf_dict dictionary and tokenization into queries.

```
    # The vector should be a dict with query terms as keys and IDF scores as values.
    query_to_check = "Binary Electronic"
    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
    for token in tokens:
        doc_ids_idf_score = [idf_dict[token][0][1]]
        query_vector_dict[token] = doc_ids_idf_score
    print(query_vector_dict)
```

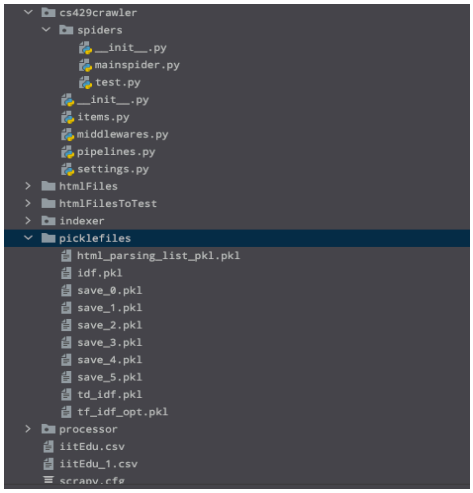
⇒ After that, we get query-vector score, we get the cosine similarity score and even sorted similarity scores using the following snippet of code.

```
    # 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 documen
    # 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(cosine_similarity_dict_parent_final)
```

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

```
Terminal:  Local  Local (2)  +
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler % python indexer/main.py
dict score of term 'Binary':=  [2, 6]
idf_dict score of term 'Binary':=  [[2, 1.8986122886681898], [6, 1.8986122886681898]]
tf_idf_dict score of term 'Binary':=  {2: [0.8887851426756534723], 6: [0.8831053739315367425]}
tf_idf_dict_opt_ext score:=  6
sample Query vector score:=  {'binary': [0.6931471805599453], 'electronic': [0.6931471805599453]}
cosine_similarity_dict_parent_term_list:=  {'binary': {0: 0.0, 1: 0.07818823467774237, 2: 0.09275166451717752, 3: 0.0, 4: 0.0, 5: 0.8328841681433753561}, 'electronic': {0: 0.0, 1: 0.0, 2: 0.5618946278438862, 3: 0.0, 4: 0.0, 5: 0.7288
7863383239446}}
[(5, 0.7689628811757698), (2, 0.6546462923618638), (1, 0.07818823467774237), (3, 0.0), (4, 0.0)]
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler %
```

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



⇒ 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.

```python
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
```

⇒ 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.

```python
@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)
```

⇒ 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.

```
52          # Using block-sorting algorithm, merged/combined together for the final one inverted index.
53          myDir = Path('picklefiles/')
54          fileNames = [file.name for file in myDir.iterdir() if file.name.startswith('save')]
55          for file in fileNames:
56              with open(os.path.join(myDir, file), 'rb') as fileName:
57                  dict_new_inverted_index = pickle.load(fileName)
58                  for k, v in dict_new_inverted_index.items():
59                      try:
60                          main_res_dict_inverted_index[k].extend(v)
61                      except KeyError:
62                          main_res_dict_inverted_index[k] = v
63          res_main_res_dict = {key: list(set(value)) for key, value in main_res_dict_inverted_index.items()}
64          print("Invrted index length: = ", len(res_main_res_dict))
```

⇒ 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.

```
101         #
102         # # Query vector and similarity
103         for idx in range(len(query_list)):
104             query_to_check = query_list[idx]
105             data = re.sub('[^\w]|_', ' ', query_to_check)  # only keep numbers and letters and spaces
106             data = data.lower()
107             data = re.sub(r'[^\x00-\x7f]', r'', data)  # remove non ascii texts
108             tokens = [data_upt for data_upt in data.split(' ') if data_upt]  # split the words and remove empty words
109             for token in tokens:
110                 doc_ids_idf_score = [idf_dict[token][0][1]]
111                 query_vector_dict[token] = doc_ids_idf_score
```

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

```
            # 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 t
            # 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!'

```

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

```
(base) harshpatel@Harshs-MacBook-Pro ~ % curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Binary Electronic\"}" "127.0.0.1:5000/"
[
  [
    5,
    0.7609628011757698
  ],
  [
    2,
    0.6546462923610638
  ],
  [
    1,
    0.07010823467774237
  ],
  [
    3,
    0.0
  ],
  [
    4,
    0.0
  ]
]
```

5 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}
```

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

```
python indexer/main.py
```

6. Then, we got saved pickle files into "picklefiles" directory.
7. 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
  ]
]
```


6 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.

2. 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.

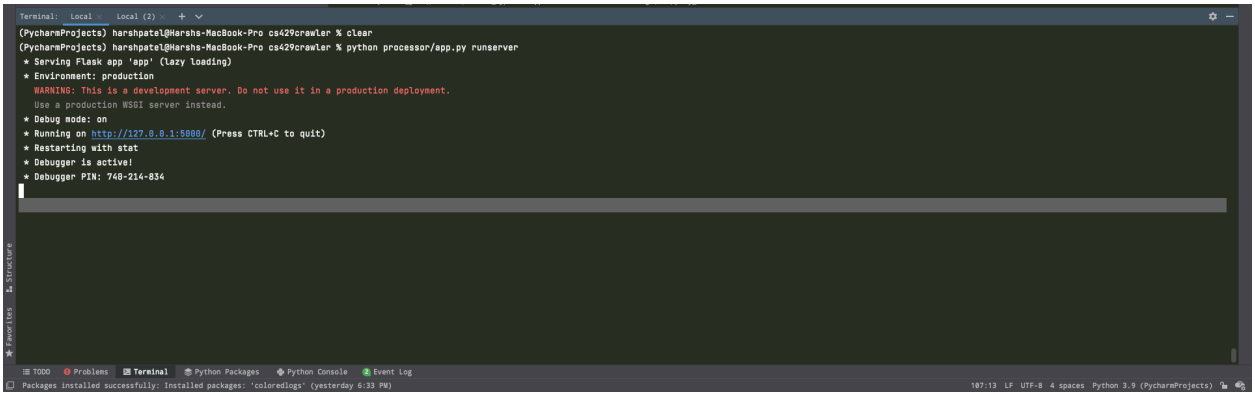
Result: - (TF-IDF score/weight representation, Cosine similarity)

```
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler %
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler % clear
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler % python indexer/main.py
tf_idf_dict score of term 'Oshawa':=  {6: [0.08024848246858651978], 7: [7.9455551533982l7e-05], 18: [0.08866183014851961 8], 20: [0.08850784214887855 24], 24: [0.08852857872892336 84], 26: [0.08848317039913282316], 30: [0.08814194812867189221]}
tf_idf_dict score of term 'Vancouver':=  {2: [0.08854846536428549 02], 3: [0.08128861083492011 15], 4: [0.08163172547861685 43], 6: [6.2257328325772316-05], 7: [0.08832911782153293 5], 12: [8.948171935636428e-05], 15: [0.08038639641302953 6], 17: [9.526313428567535e-05], 18: [0.08068535202977436778], 20: [0.08078759415160149 3], 24: [0.08082184714733841 26], 26: [0.08150102788222675 25], 28: [0.08085408785171566 61], 30: [0.08822948957881648503], 32: [0.08031382946251733 51]}
sample Query vector score:=  {'Oshawa': [1.5884503755608479], 'Vancouver': [0.8183183235139513]}
cosine_similarity_dict_parent_term_list:=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.1336245862612927, 7: 0.044149628231321486, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.3677471755231222, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857709529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374305274877, 31: 0.0, 32: 0.0, 33: 0.0}, 'Vancouver': {0: 0.0, 1: 0.0, 2: 0.25753637677886956, 3: 0.6013212428857586, 4: 0.7661899783786475, 5: 0.0, 6: 0.08292334353434143, 7: 0.1545399524017414, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.042816869781616 16, 13: 0.0, 14: 0.0, 15: 0.18143558133576343, 16: 0.0, 17: 0.044731588227343394, 18: 0.32181262491865975, 19: 0.0, 20: 0.36982124557547 7, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.3855293728597456, 25: 0.0, 26: 0.7048198241874896, 27: 0.0, 28: 0.4818863391284489, 29: 0.0, 30: 0.10353267678726956, 31: 0.0, 32: 0.1473611789656298, 33: 0.0}}
Sorted cosine_similarity_dict_parent_final:=  [(26, 0.9732943748292553), (4, 0.7661899783786475), (18, 0.6895598884417 82), (24, 0.6792351438126684), (20, 0.6515681788924783), (3, 0.6813212428857586), (28, 0.4818863391284489), (2, 0.25753637677886956), (7, 0.19868958863386288), (30, 0.18240641976888133), (15, 0.18143558133576343), (6, 0.16285794168463415), (32, 0.1473611789656298), (17, 0.044731588227343394), (12, 0.042816869781616 16), (1, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (13, 0.0), (14, 0.0), (16, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (29, 0.0), (31, 0.0), (33, 0.0)]
(PycharmProjects) harshpatel@Harshs-MacBook-Pro cs429crawler %
```

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

```
python processor/app.py runserver
```

Result: - (Flask server started)



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

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

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

⇒ 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.

```
 * Debugger is active!
 * Debugger PIN: 748-214-834
Query list(from Json format):=  ['Oshawa Vancouver']
Invrted index length: =  28395



Top-K ranked results(Query word -> similarity result):=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.1336245862612927, 7: 0.044149628231321486, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.3677471755231222, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857709529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374305274877, 31: 0.0, 32: 0.0, 33: 0.0}}
Top-K ranked results(Top-K Document Numbers):=  [(18, 0.3677471755231222), (24, 0.2937857709529149), (20, 0.28173893331784864), (26, 0.26847455864176573), (6, 0.1336245862612927), (30, 0.07887374305274877), (7, 0.044149628231321486), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (12, 0.0), (13, 0.0), (14, 0.0), (15, 0.0), (16, 0.0), (17, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (28, 0.0), (29, 0.0), (31, 0.0), (32, 0.0), (33, 0.0)]



Top-K ranked results(Query word -> similarity result):=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.1336245862612927, 7: 0.044149628231321486, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.3677471755231222, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857709529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374305274877, 31: 0.0, 32: 0.0, 33: 0.0}, 'Vancouver': {0: 0.0, 1: 0.0, 2: 0.25753637677886956, 3: 0.6013212428857586, 4: 0.7661899783786475, 5: 0.0, 6: 0.08292334353434143, 7: 0.1545399524017414, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.042816869781616 16, 13: 0.0, 14: 0.0, 15: 0.18143558133576343, 16: 0.0, 17: 0.044731588227343394, 18: 0.32181262491865975, 19: 0.0, 20: 0.36982124557547 7, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.3855293728597456, 25: 0.0, 26: 0.7048198241874896, 27: 0.0, 28: 0.4818863391284489, 29: 0.0, 30: 0.10353267678726956, 31: 0.0, 32: 0.1473611789656298, 33: 0.0}}
Top-K ranked results(Top-K Document Numbers):=  [(26, 0.9732943748292553), (4, 0.7661899783786475), (18, 0.6895598884417 82), (24, 0.6792351438126684), (20, 0.6515681788924783), (3, 0.6813212428857586), (28, 0.4818863391284489), (2, 0.25753637677886956), (7, 0.19868958863386288), (30, 0.18240641976888133), (15, 0.18143558133576343), (6, 0.16285794168463415), (32, 0.1473611789656298), (17, 0.044731588227343394), (12, 0.042816869781616 16), (1, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (13, 0.0), (14, 0.0), (16, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (29, 0.0), (31, 0.0), (33, 0.0)]
127.0.0.1 - - [30/Apr/2022 17:09:08] "POST / HTTP/1.1" 200 -
```

```
(base) harshpatel@Harshs-MacBook-Pro ~ % curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Oshawa Vancouver\"}" "127.0.0.1:5000/"
[
  [
    26,
    0.9732943748292553
  ],
  [
    4,
    0.7661899783706475
  ],
  [
    18,
    0.689559800441782
  ],
  [
    24,
    0.6792351430126604
  ],
  [
    20,
    0.6515601788924783
  ],
  [
    3,
    0.6013212428057586
  ],
  [
    28,
    0.4010063391284489
  ],
  [
    2,
    0.25753637677806956
  ],
  [
    7,
    0.1986895806306288
  ],
  [
    30,
    0.18240641976000133
  ],
  [
    15,
    0.18143558133576343
  ],
  [
    6,
    0.16285794160463415
  ],
  [
    32,
    0.1473611789656298
  ],
  [
    17,
    0.044731580227343394
  ],
  [
    12,
    0.04201686978161616
  ],
  [
```

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)

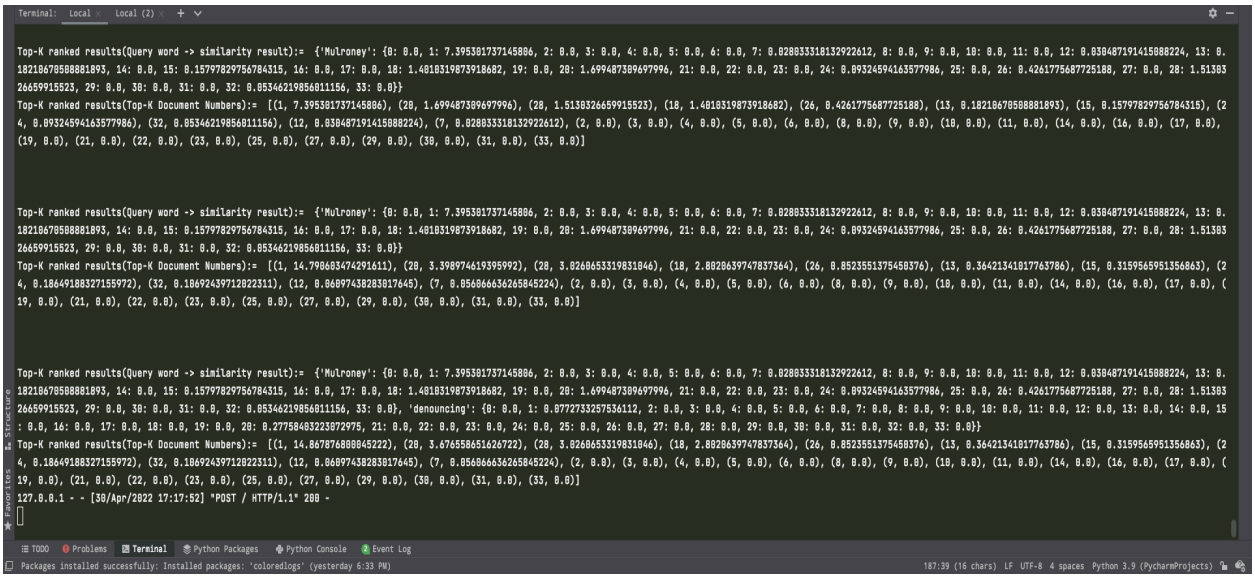
```
Top-K ranked results(Query word -> similarity result):=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.13362458626612927, 7: 0.8441496282313214 86, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.36774717552312 22, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857789529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374385274877, 31: 0.0, 32: 0.0, 33: 0.0}}
Top-K ranked results(Top-K Document Numbers):=  [(18, 0.3677471755231222), (24, 0.2937857789529149), (20, 0.28173893331784864), (26, 0.26847455864176573), (6, 0.13362458626612927), (30, 0.07887374385274877), (7, 0.8441496282313214 86), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (12, 0.0), (13, 0.0), (14, 0.0), (15, 0.0), (16, 0.0), (17, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (28, 0.0), (29, 0.0), (31, 0.0), (32, 0.0), (33, 0.0)]



Top-K ranked results(Query word -> similarity result):=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.13362458626612927, 7: 0.8441496282313214 86, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.36774717552312 22, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857789529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374385274877, 31: 0.0, 32: 0.0, 33: 0.0}}
Top-K ranked results(Top-K Document Numbers):=  [(18, 0.7354943518462444), (24, 0.5874115419858298), (28, 0.5634778666340813), (26, 0.5369491812835315), (6, 0.2672498125225854), (30, 0.15774748618548153), (7, 0.8882992564264297), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (12, 0.0), (13, 0.0), (14, 0.0), (15, 0.0), (16, 0.0), (17, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (28, 0.0), (29, 0.0), (31, 0.0), (32, 0.0), (33, 0.0)]



Top-K ranked results(Query word -> similarity result):=  {'Oshawa': {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.13362458626612927, 7: 0.8441496282313214 86, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.36774717552312 22, 19: 0.0, 20: 0.28173893331784864, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.2937857789529149, 25: 0.0, 26: 0.26847455864176573, 27: 0.0, 28: 0.0, 29: 0.0, 30: 0.07887374385274877, 31: 0.0, 32: 0.0, 33: 0.0}, 'Vancouver': {0: 0.0, 1: 0.0, 2: 0.25753637677806956, 3: 0.6013212428057586, 4: 0.7661899783706475, 5: 0.0, 6: 0.02923343534334143, 7: 0.15453995240174 14, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.04201686978161616, 13: 0.0, 14: 0.0, 15: 0.18143558133576343, 16: 0.0, 17: 0.044731588227343394, 18: 0.32181262491865975, 19: 0.0, 20: 0.3698212455754377, 21: 0.0, 22: 0.0, 23: 0.0, 24: 0.3855293728597456, 25: 0.0, 26: 0.7848198241874896, 27: 0.0, 28: 0.4010063391284489, 29: 0.0, 30: 0.18353267670726956, 31: 0.0, 32: 0.1473611789656298, 33: 0.0}}
Top-K ranked results(Top-K Document Numbers):=  [(26, 1.2417689254718211), (18, 1.057386975964984), (24, 0.9729489139655764), (28, 0.933299112289519), (4, 0.7661899783706475), (3, 0.6813212428057586), (28, 0.4010863391284489), (6, 0.29648244786592687), (30, 0.26128816281274 21), (2, 0.25753637677806956), (7, 0.24283920886438437), (15, 0.18143558133576343), (32, 0.1473611789656298), (17, 0.044731588227343394), (12, 0.04201686978161616), (1, 0.0), (5, 0.0), (8, 0.0), (9, 0.0), (10, 0.0), (11, 0.0), (13, 0.0), (14, 0.0), (16, 0.0), (19, 0.0), (21, 0.0), (22, 0.0), (23, 0.0), (25, 0.0), (27, 0.0), (29, 0.0), (31, 0.0), (33, 0.0)]
127.0.0.1 - - [30/Apr/2022 17:13:27] "POST / HTTP/1.1" 200 -
```

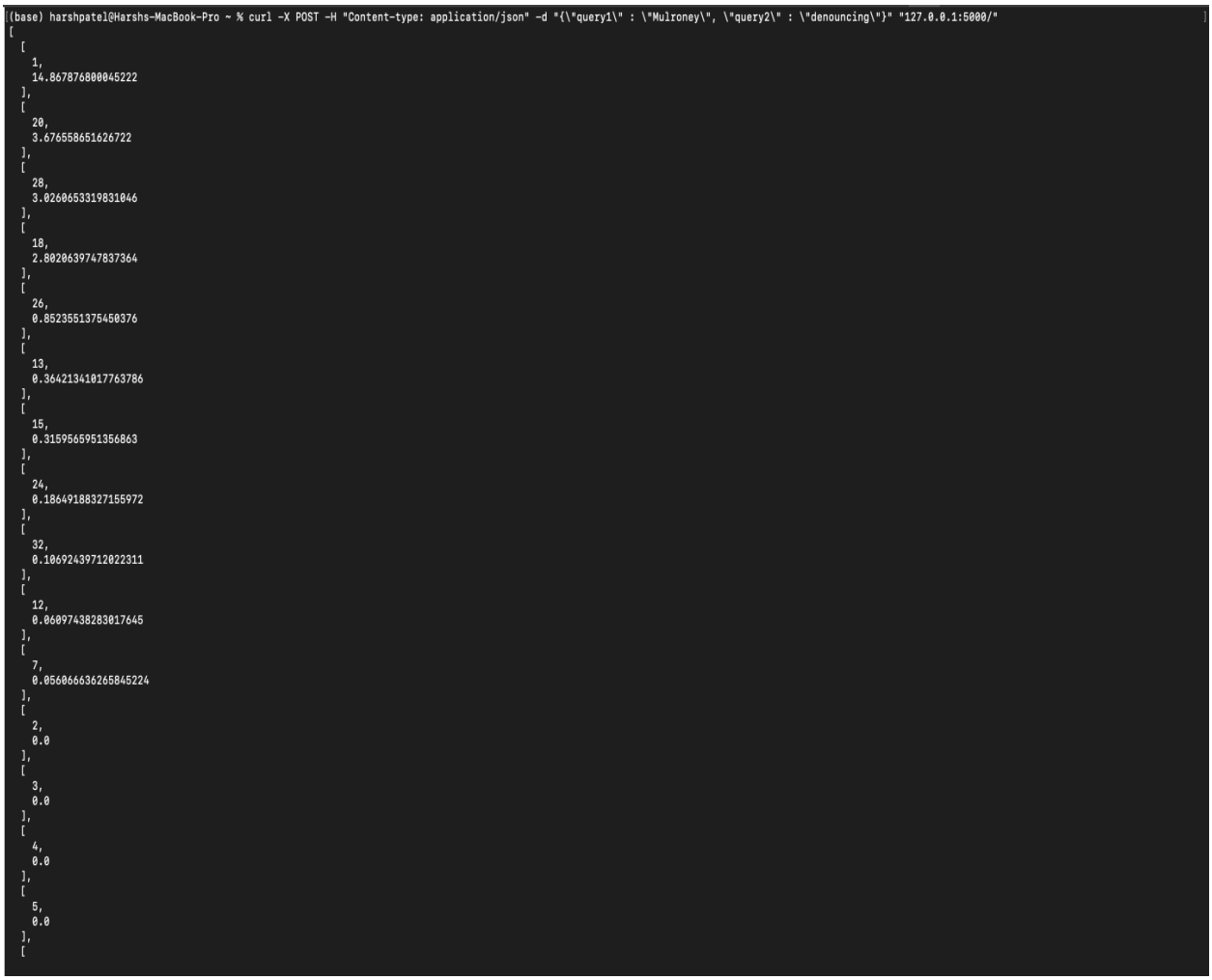
```
(base) harshpatel@Harshs-MacBook-Pro ~ % curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Oshawa\", \"query2\" : \"Vancouver\"}" "127.0.0.1:5000/"
[
  [
    26,
    1.2417689254710211
  ],
  [
    18,
    1.057306975964904
  ],
  [
    24,
    0.9729409139655754
  ],
  [
    20,
    0.933299112209519
  ],
  [
    4,
    0.7661899783706475
  ],
  [
    3,
    0.6013212428057586
  ],
  [
    28,
    0.4010063391284489
  ],
  [
    6,
    0.29648244786592687
  ],
  [
    30,
    0.2612801628127421
  ],
  [
    2,
    0.25753637677806956
  ],
  [
    7,
    0.24283920886438437
  ],
  [
    15,
    0.18143558133576343
  ],
  [
    32,
    0.1473611789656298
  ],
  [
    17,
    0.044731580227343394
  ],
  [
    12,
    0.04201686978161616
  ],
  [
```

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)

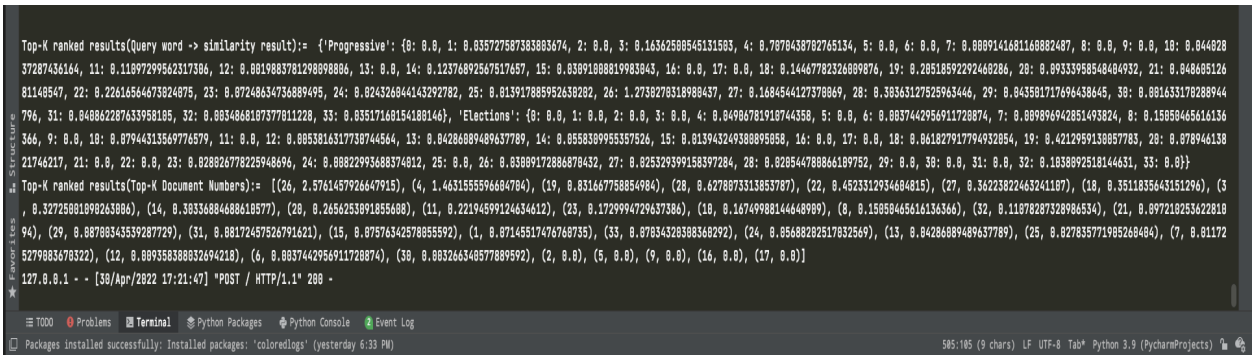




4th Query: -

```
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)

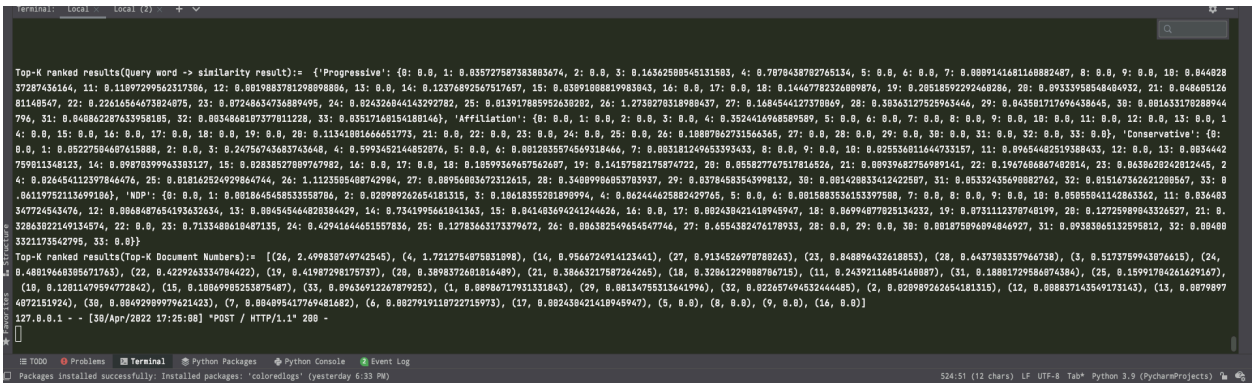


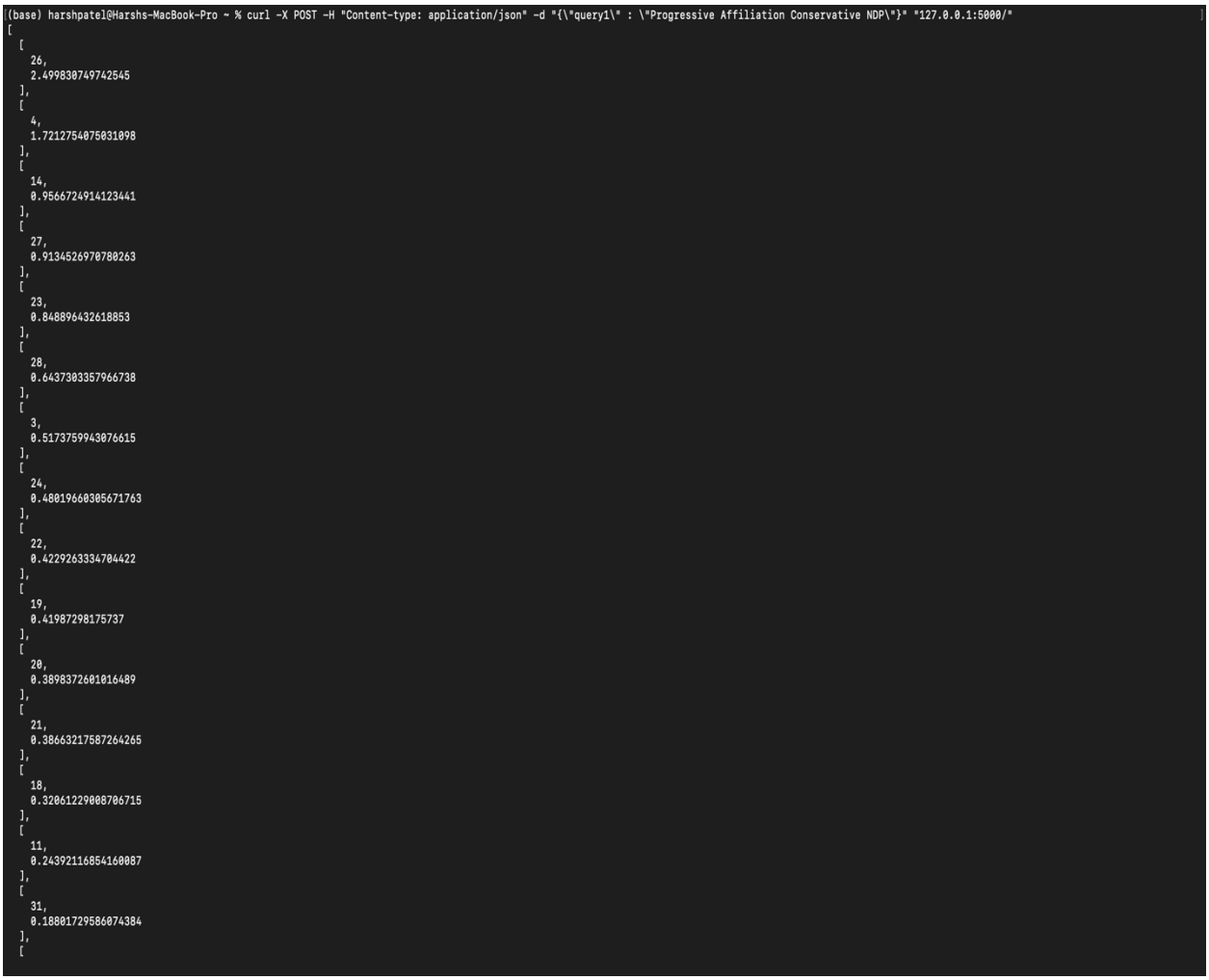
```
(base) harshpatel@Harshs-MacBook-Pro ~ % curl -X POST -H "Content-type: application/json" -d "{\"query1\" : \"Progressive\", \"query2\" : \"Elections\"}" "127.0.0.1:5000/"
[
  [
    26,
    2.5761457926647915
  ],
  [
    4,
    1.4631555596604704
  ],
  [
    19,
    0.831667758854984
  ],
  [
    28,
    0.6278073313853787
  ],
  [
    22,
    0.4523312934604815
  ],
  [
    27,
    0.36223822463241107
  ],
  [
    18,
    0.3511835643151296
  ],
  [
    3,
    0.327250010902630O6
  ],
  [
    14,
    0.30336884688610577
  ],
  [
    20,
    0.2656253091855608
  ],
  [
    11,
    0.22194599124634612
  ],
  [
    23,
    0.1729994729637386
  ],
  [
    10,
    0.16749988144648909
  ],
  [
    8,
    0.15050465616136366
  ],
  [
    32,
    0.11078287328986534
  ],
  [
```

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)

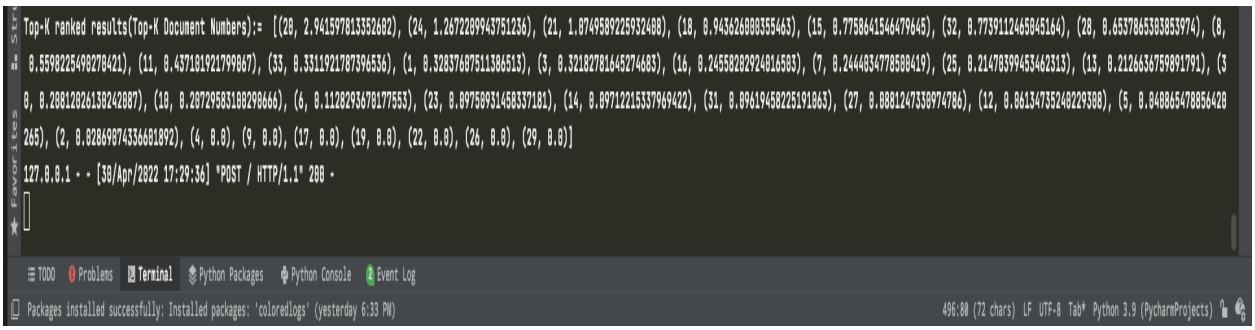


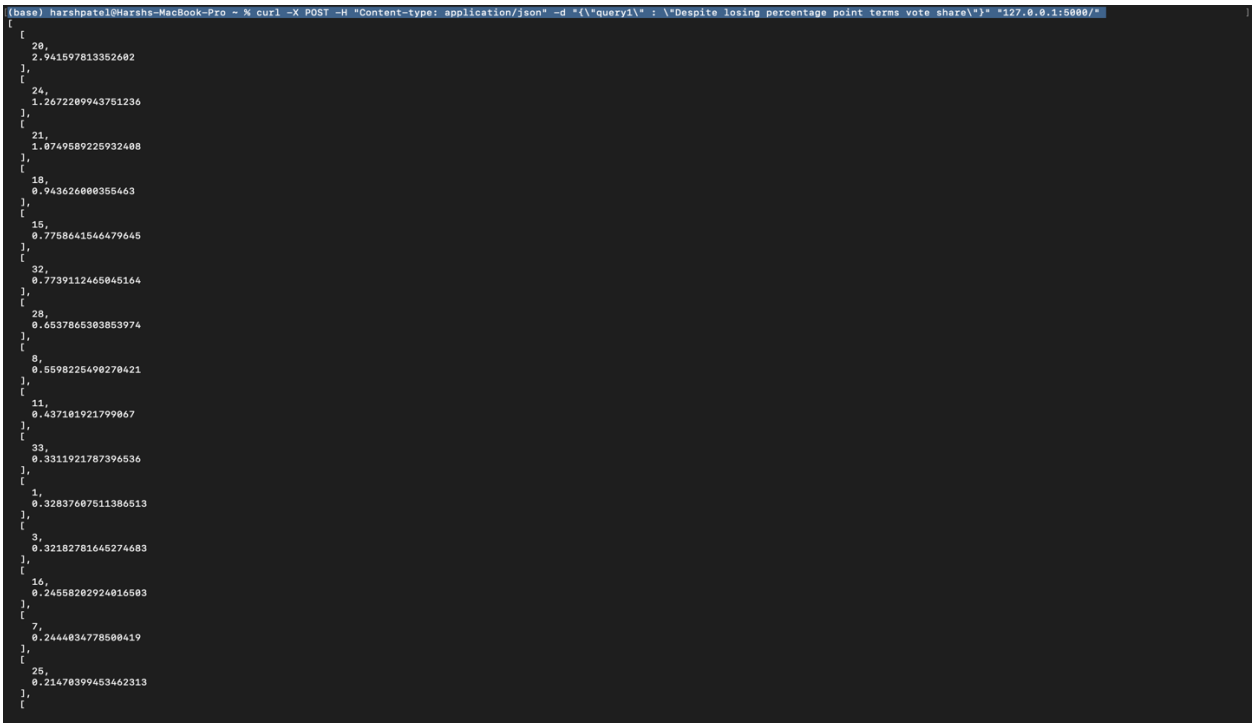


6th Query: -

```
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)





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}"
```

7 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/

8 Test Cases: -

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

9 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**

```python
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

10 Bibliography: -

1. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, 2008 Introduction to Information Retrieval, Cambridge University Press.
2. Donald Metzler, Trevor Strohman, and W. Bruce Croft, 2010, Search Engines Information Retrieval in Practice, Pearson Education, Inc
3. https://en.wikipedia.org/wiki/Information_retrieval