Project Report

Matiss M. Mednis
CS 429
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois 60616
mmednis@hawk.iit.edu

Abstract – **This Information Retrieval system utilizes tf-idf score vectors and cosine similarity to retrieve relevant documents from an indexed directory. The directory is developed by a Scrapy Crawler which can be defined by an allowed domain, start url, max depth, and max number of pages downloaded to the local machine as .html files. They are read in and tf-idf scores are calculated and stored on the machine using scikit-learn libraries as a .pkl file. With the files stores locally users are able to produce queries and retrieve top-k relevant results by sending requests to a Flask server. The objectives of this system are to provide a useable and sound information retrieval system locally. Further development of this system may include more sophisticated methods of ranking query results to provide better results for a user's information need and more options when querying such as spelling correction or suggestions. Results are returned to the user as relative paths to the html files.**

Index Terms – Information retrieval system, inverted, index, scrapy, web crawling, Flask

OVERVIEW

IR systems can be implemented in many ways. The methodology for this system and its general solution structure follow the methods and steps defined by the project requirements. As a general overview, the system is made up of three main steps or components.

1. A WebCrawler for downloading web documents in html format.
2. An indexer which generates an inverted index from the downloaded html documents.
3. A Flask based processor which handles user queries in json format.

The theory behind this system is informed by CS429 Information Retrieval lectures, labs, and notes as well as the course text: An Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze.

Web crawling or web scraping is used to extract structured data from web pages. This may include link, body text, the URL, or any other attributes of a web page. In the context of information retrieval since people are often searching for the text of a website or the title, we want to read and process that data so that we have a directory of website text and information which an IR system can reference to give the user the best response to their query. A proper web crawling implementation is necessary for this system so we can follow links between website in order to build up our directory from a root URL as well as extracting the HTML data from that site to get its text contents (Scrapy, n.d.).

Generating an inverted index is necessary in an IR system as it allows for efficient storage and searching of terms and documents. An inverted index uses terms as keys and lists of documents containing the given term as values to those keys. Therefore, terms are mapped to the documents which contain them. This helps queries be processed more efficiently as we search for which documents contain a given term. The inverted nature of this index also allows for TF-IDF scores to be stored efficiently. In this solution, the inverted index will store terms as keys and document-TFIDF score pairs in a list as values to the keys (terms).

User queries represent the information that a user wants to retrieve from our system or from the documents we have indexed. Often there are many possible documents that are relevant but we want to order and rank these for the user and return the most relevant documents. This is done by finding the similarity between the query and our TF-IDF scored documents. Those with the highest similarity are ranked highest and returned to the user first. This solution allows users to define how many of the relevant documents they would like returned and is defined as the k-top ranked documents.

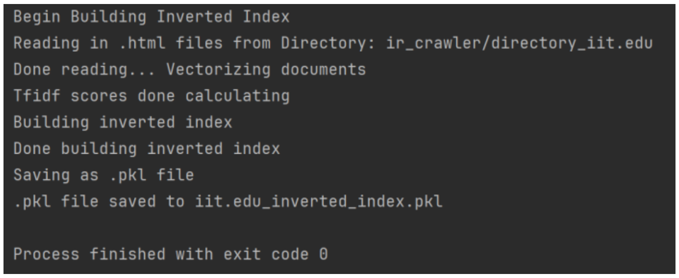
Given the requirements of the solution as outlined in the 3 part list above, Python was used to implement the system requirements. In order to address WebCrawling a Scrapy web crawler is proposed. For the second requirement, the indexer should calculate TF-IDF scores for each term-document pair and store those efficiently on the local machine. To achieve that the system utilizes the TF-IDF Vectorizer package from scikit-learn to generate a TF-IDF matrix for each document and term pair. Those are then put into an inverted index structure and stored as a .pkl file using the Pickle library. Finally, in order to interact with and use this system a Flask based processor is written using the Flask library in Python. The system must be able to accept JSON format requests from the user which include a query as well as an additional parameter defining how many of the top results the user would like returned to them.

Thus, by utilizing the material from the course book, lectures, notes, and labs and implementing it with the methods described in the paragraph previous, the proposed solution develops a usable IR System.

DESIGN

Crawler: The crawler is implemented using Scrapy library version 2.11.1. It is called from the crawler.py script through the Python terminal with *Scrapy crawl crawler*. The script sits in ir_crawler/ir_crawler/spiders/crawler.py. The allowed domains, start urls, maximum number of downloaded pages, and max page depth are all hardcoded into the script. In order to alter any of these the relevant attributes must be altered in the source code. Otherwise, running *crawler.py* will download the pre-defined number of pages from the predefined start url with the defined maximum depth. The system logs when pages are downloaded, skipped due to previously visited and downloaded webpages, the current depth, and current number of downloaded pages as the crawler runs. The downloaded pages are then downloaded to the local machine in a file named ir_crawler/directory_allowed_domains where allowed_domains is the first element of the allowed domains list as defined inside of crawler.py. This helps automate the filenaming and downloaded process when the allowed domains are altered and created new directories for new domains such that multiple directories of webpages may be created from the crawler. Each indivdually downloaded webpage is named page_URL.html where URL is the page's relative URL to the base domain.

Indexer: The indexer script is *inverted_index_generator.py* and reads the dowloaded .html files from the directory defined by the *allowed_domains*. The allowed domains field in the index script must match the allowed domains defined in the crawler for the file reading to work. It will read the .html files downloaded by the crawler and parse the HTML text content using BeautifulSoup. The text is cleaned and stored in a list as a list of documents. Similarly, the filename is stored in a list as well so that it may act as the documentID when generating the inverted index. The TF-IDF scores are calculated using scikit-learn TfidfVectorizer. Stopwords are removed and every term is lowercased. The indexer then builds the inverted index as described in the overview section and downloads it to the local file system in the same directory as the *inverted_index_generator.py* with the name *allowed_domains*_inverted_index.pkl. Again, where *allowed_domain* is pulled from the hardcoded start domain variable in the script which must match the start domain which generated the relevant inverted index you wish to generate. As the index is devloped, the steps completed are printed to the terminal as they occur shown in Figure 1 below



```
Begin Building Inverted Index
Reading in .html files from Directory: ir_crawler/directory_iit.edu
Done reading... Vectorizing documents
Tfidf scores done calculating
Building inverted index
Done building inverted index
Saving as .pkl file
.pkl file saved to iit.edu_inverted_index.pkl

Process finished with exit code 0
```

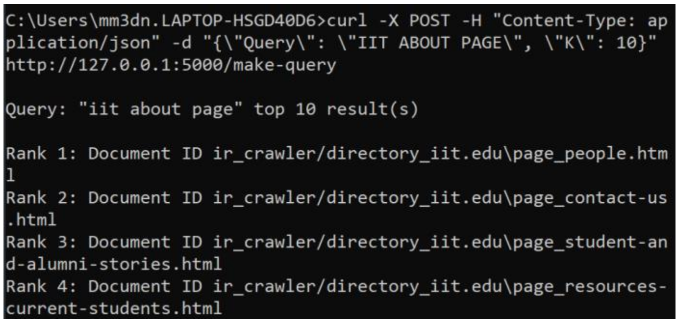
Figure 1. Terminal printout of index steps completing on crawled webpage with allowed domain in this case being iit.edu. Therefore the crawler and indexer both have the allowed domain as iit.edu and the index is saved to the same naming convention at iit.edu_inverted_index.pkl

Query Processor: The query processor is implemented as a Flask application in *flask_app.py*. The processor expects to receive JSON formatted requests to 127.0.0.1:5000/make-query. Thus the server is run on port 5000 of the localmachine at /make-query. The format of the JSON query allows for the user to define the query as well as the top-k results. The required format is shown below

{"Query": "*user query*", "K": *top k results*}

The user query must be a string and the top k results must be an integer. Both Query and K are case sensitive. If the user does not input a JSON formatted request, ommits one of the parameters, or passes the wrong type of either parameter the application will return a descriptive error to the user. These checks are done in the process_data() function.

The query processor also reads in the inverted index .pkl file which is expected to exist with the naming convention as described in the indexer section. To allow for running the server on various potential inverted indexes downloaded from the crawler and indexed by the inverted index script, the path of the inverted index is defined inside of the flask app script as *inverted_index_file_path*. If the file is not found then an error will be returned to the user upon querying that the expected inverted index file does not exist. The capability to send the desired index file to query from was considered but it may make testing and grading harder so this approach was taken.



```
C:\Users\mm3dn.LAPTOP-HSGD40D6>curl -X POST -H "Content-Type: ap
plication/json" -d "{\"Query\": \"IIT ABOUT PAGE\", \"K\": 10}"
http://127.0.0.1:5000/make-query

Query: "iit about page" top 10 result(s)

Rank 1: Document ID ir_crawler/directory_iit.edu\page_people.htm
l
Rank 2: Document ID ir_crawler/directory_iit.edu\page_contact-us
.html
Rank 3: Document ID ir_crawler/directory_iit.edu\page_student-an
d-alumni-stories.html
Rank 4: Document ID ir_crawler/directory_iit.edu\page_resources-
current-students.html
```

Figure 2. Example successful JSON request and response

Each component is expected to run in sequence. This means that to query based of a totally new directory not already included in the github then first the webcrawler must be run. The desired depth, allowed url, start url, and number of pages must be defined inside of the script. Then the indexer must be run with the same allowed_url defined in the script variables. Finally, the Flask app can be run and queried on assuming the script has its variables set to also have the same allowed_url defined when crawling. In this way, as long as the allowed URLs are consistent in each script and they are run sequentially such that each previous dependent file or directory can be made, then the scripts are each integrated properly and new indexes and queries can be made.

ARCHITECTURE

Table 1 outlines the software packages and versions used by this system at the time of development.

Software component	**Version**	**Python Script**
BeautifulSoup4	4.13.0b2	inverted_index_generator.py
Flask	3.03	flask_app.py
Python	3.12	*all scripts*
Scrapy	2.11.1	crawler.py
Scikit-learn	1.4.2	inverted_index_generator.py

Table 1. Python library versions and the scripts which use them for the IR System

Communications between each step in the system is not very cohesive without the hardcoding of the source code. For instance, the depth, page length, start url and domain are determined in the source code and can only be changed by altering it. The path to the downloaded html documents for generating the inverted index is an expected file path imported from the crawler based on its allowed domains. So, once the crawler is run on a given domain then the indexer can be run and will automatically find the file path and names of the html documents and generate and download the inverted index. Then, the flask app source code must point to the location of the inverted index. In other versions this was also referenced by importing the name from another script but I figured it might be easier to query on other .pkl inverted indexes faster if the direct path could just be put into the source code. Considering the need for altering of the source code for communication between all of the files properly, this system requires some altering of the source code to change the desired sites scraped and the index with which the user would be querying once they run the flask app. This should be noted for all alterations, improvements, or variations in application. It may not be the most seamless but it works when the app is just delivered as an IR system to the user.

OPERATION

Installation and use of the system immediately requires the download of the entire GitHub repo and scripts and files found at https://github.com/tissoSmelly/CS429_IR_Final. Upon doing so some initial inverted indexes are downloaded which can be immediately queried by simply running flask_app.py and writing Curl requests as described in the Query Processor subsection of the Design section. To change the index or crawl new pages see the steps below. Otherwise just run the *flask_app.py* and begin making JSON requests.

To Change Inverted Indexes User can Query from:

Altering which inverted index to query from must be done in flask_app.py variable *inverted_index_file_path*. So if your inverted index is named in the file *iit.edu_inverted_index.pkl* then the inverted_index_file_path must be equal to *"iit.edu_inverted_index.pkl"*

To change the depth, max pages, start url, allowed domain of the crawler:

Simply alter the associated and similarly named variables in the source code of *crawler.py*. To change the allowed domains and start urls change ALLOWED_DOMAINS and START_URLS outside of the class not allowed_domains and start_urls inside of the class. This is because those variables are referred to by the indexer to see where the html files are. Adhere to the conventions outlined in the Scrapy documentation for naming of urls and domains. Then run the crawler to download the new set of html files.

To change the html files which should generate the inverted index:

Inverted_index_generator.py should be run after the crawler and determines the name of the html file directory based on the ALLOWED_DOMAINS variable in the crawler. Therefore, as long as the crawler has been run on the given domains in the crawler script or a directory already exists that was created by the crawler for the given domain then the index generator will generate the index automatically from the html files downloaded from the allowed domains in *crawler.py*. So, after crawling just run the indexer and the new inverted index will be created.

To just run queries:

To run queries you only need to point to the desired index in the *flask_app.py* source code as previously mentioned and then run the script. It is not necessary to download the crawler or indexer. You only need a .pkl inverted index and the flask app script.

CONCLUSION

The system successfully allows a user to make JSON query requests and receive ranked documents. The crawler is able to download a given number of maximum pages and crawl a given depth of links from those pages. The inverted index is generated from the downloaded html files and stores TF-IDF scores for each term-document pair. This allows cosine similarities between queries and documents to be calculated and compared for ranked retrieval. The system successfully solves each of the 3 requirements outlined in the overview and outlined in the project requirements.

The system could implement more complex methods of information retrieval and potentially create a better service for the user's information need. At the moment, its methods are fairly simple and could be improved upon. Testing of the system was not done with user judgements, precision, recall, or training data. Rather, some test queries were made aiming to get certain result pages and it was found those often were returned to the user. For instance, in testing a query "research" on an iit.edu index "conducting-research-illinois-tech.html" was returned #4 and "about-research" was returned #1. When querying "initiative quantum commercial" the first ranked result was "computation-and-data.html" which has those exact words occurring on it and was the target #1 result in testing. Even so, more rigorous testing and ranking methods could be implemented to improve the system.

The system also does not allow for a lot of user-based crawling, index creation, or index selection. All of that must be done by directly altering the code. In future systems it may be of interest to implement greater control for the user so they may select what they want to index and crawl so they may query from it more seamlessly. I just wasn't sure how to best implement that for grading.

The system is fairly rudimentary and could be testing more rigorously so using it for high value or mission critical information retrieval is not suggested.

DATA SOURCES

Scrapy documentation: https://docs.scrapy.org/en/latest/

Flask documentation:
https://flask.palletsprojects.com/en/3.0.x/

Scikit-learn documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

BeautifulSoup documentation:
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

TEST CASES

To test the crawler's functionality tests were done to confirm the number of downloaded files was equal to (or less than if the webpage pointed to fewer pages than the max) the max number of webpages parameter. The max depth was checked by printing and logging the number of pages we have gone deep relative to the given link. Different start URLs were tested to ensure that those downloaded html files within the allowed domains and downloaded them according to the expected naming conventions.

The inverted index was tested for reading all of the html documents in and cleaning them and converting them to a list of documents with associated list of document IDs which was the html url. This was confirmed to work as the numbers matched up to the expected number of documents. Additionally once the TF-IDF calculations were done they were checked to be associated with the correct term-document pairs and were consistent with the inverted index without TD-IDF scores.

The Flask application was tested for invalid users inputs and returned errors when the query was missing, k was missing, or when a non-JSON valid request was passed to the server. Returned queries were tested for validity based on personal judgement of information need. Test queries were made with a given document in mind and developed as to mirror what query a user might come up with to target that document. In various cases tested on iit.edu queries involving "harassment complaints assault" returned pages related to Title XI, reporting, or misconduct. Queries involving "mathematics and computer science" returned result pages involving IIT Academic Programs and computer engineering pages. In this way, the results were judged and seemed fit enough for a valid and usable IR system.

SOURCE CODE

The IR System and all scripts and data referenced in this report and required by the assignment are hosted at https://github.com/tissoSmelly/CS429_IR_Final.

To understand how to query and run the flask application look at section Architecture and To Just Run Queries. To understand how to format queries refer to Query Processor subsection in the Design section. Overall. All that is needed to make base test queries is the github repo cloned and *flask_app.py* to be run and JSON query requests to be made on port 5000 to /make-query.

The source code also contains comments to assist with the source code's understanding.

README file on the GitHub also has short instructions if you only wish to run the Flask server and obtain results from the default index.

REFERENCES

Scrapy documentation: https://docs.scrapy.org/en/latest/

Flask documentation:
https://flask.palletsprojects.com/en/3.0.x/

Scikit-learn documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

BeautifulSoup documentation:
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://www.youtube.com/watch?v=cf6KPRjqlP8&t=195s

https://www.youtube.com/watch?v=kaBT35Sabss