Paper Searching Website Application *

Wang Yuhan[1] and Yan Juefei[2]

[1]USC Viterbi School of Engineering, USC ID: 9271532881, wang499@usc.edu
[2]USC Viterbi School of Engineering, USC ID: 9672393395, juefeiya@usc.edu

ABSTRACT

The idea is to scrape data from websites, download PDF files from websites, make a program to extract metadata from PDF files, upload metadata to a cloud database, build searching application to search technical papers by facets and keywords, and finally design user interface to present the searching result with access for downloading.

Keywords: Paper searching, Firebase, Keywords and faceted searching.

1. PROJECT OVERVIEW

This project is to search papers through keywords and specific facets. Project implementation is divided in to 3 parts:

- **Metadata Collecting:** This part contains scraping data from websites, extracting metadata from pdf files and websites, preprocessing metadata and indexing papers, uploading to firebase.
- **Searching Function:** This part contains implementation of keywords and faceted searching. Two logic modes for either searching method are implemented and can be chosen by users.
- **User Interface:** At last, a well-designed user interface is used for gathering users' queries and giving queried results.

The following report will give detailed explanation for these three parts and give demos in the end.

Code details is available in Github[†].

2. METADATA

In this project, data are scraped using web crawler from sciencemag.org where we can reach free papers and download these papers easily. Then metadata are extracted from PDF files and uploaded to firebase for further use. The programs in this part are mainly in Python 3.6. Packages used in this part include urllib, bs4, pdfminer, requests.

2.1 Web Crawler

As a first step of scraping data from website, we analyzed the html structure of sciencemag.org. After examining the webpage carefully, we divided the work of scraping data into 3 steps:

- Scraping the URLs of papers from sciencemag.org and its sub-pages. Storing URLs in the local disk.
- With paper URLs, scraping some extra metadata especially the PDF links and storing in the local disk.
- Downloading PDF documents by the PDF links of papers.

During the above steps, we use urllib package to open URLs and download PDF documents, and use BeautifulSoup package to analyze html webpage in Python 3.

The reason we divided the work is that the data we wanted to scrape is very large, which contains more than 4,000 webpages, and its hard to open webpages and download all files in one time which will cost plenty of time and greatly increase the server pressure. After dividing the work, we were able to did our work in several times.

* Final Project in INF 551: Foundations of Data Management, supervised by Professor Wensheng Wu.
† https://github.com/yuhanWg/INF551_Paper_Searching

2.2 Extracting Metadata

There are two ways to get metadata. One is from PDF files. In this way, we use pdfminer python package to extract metadata (we use PDFParser and PDFDocment objects in pdfminer.pdfparser specifically). We first looped the directory storing PDF files and then extracted metadata from each PDF file. The metadata include publish date, title and etc.

There are some metadata we can't get in the above way, so we also scraped additional paper information from websites directly. Here, python package bs4.BeautifulSoup is used. The corresponding metadata are like DOI, paper type, pdf link for downloading.

2.3 Uploading to Firebase

Next step is to upload metadata to the Real-time Database on Firebase. Before uploading, metadata are presented in JSON format. In order to implement the search app and speed up the searching process, first we need to restructure the JSON file to make it easier to retrieve papers by metadata. So, metadata are preprocessed before uploading. Preprocessing manipulation are in following steps:

- For each paper, a unique ID is given to help paper retrieving. In firebase, they were stored in the form: {*ID: paper*}. Here ID is a number starting from zero, and paper is a JSON object which stores all metadata of the paper, details in next bullet point.
- Metadata of each paper are stored in a JSON object where the key is specific metadata type like *"author"*, *"type"* and the value is about metadata like *"John Smith"*, *"Research Article"*. Full text can also be reached from this object.
- For each paper, title and full text are separated into words according to special signs, stopped words are removed using **nltk** package and all words are converted to lowercase (since keyword searching is case-insensitive). This part are stored as a JSON object in the form: {*word: [ID, ID, ID,...]*}. Here the key is word and the value is a list which contains all IDs of the papers containing the key word.
- For each paper and each type of metadata like paper type, published year and author name, a key-value pair is created for faceted searching. Here key is specific metadata content like *2017*, *"john smith"*, and the value is the list of IDs of corresponding papers. Notice that, here not all metadata are preprocessed in this way, since it's meaningless for faceted searching through some metadata like pdf link.

Screen shots of Firebase are in **Figure 1**.

3. SEARCHING APPLICATION

This section is to introduce how to implement searching function in our project.

3.1 Keyword Searching

As mentioned above in **section 2.3**, each paper is indexed by an ID. This will help to paper searching. Also, in previous section, we have mentioned that, paper texts and titles are split into lowercased words without stopped words. It's totally for the implementation of keyword searching.

The keyword searching without facets process is shown below:

- **Receive keywords from Interface:** After the user input the keywords and click the confirm button, the back-end program can receive sentence in the form *"Example of Searching Keywords"*. First step is to split the sentence into several keywords, convert them into lowercase and remove stopped words. The result in this step is presented in an array like *["example", "searching", "keywords"]*. Result is sent to next step.
- **Search IDs by each keyword:** For each keyword, an ID list can be retrieved. With multiple keywords, several ID lists are sent to next step.

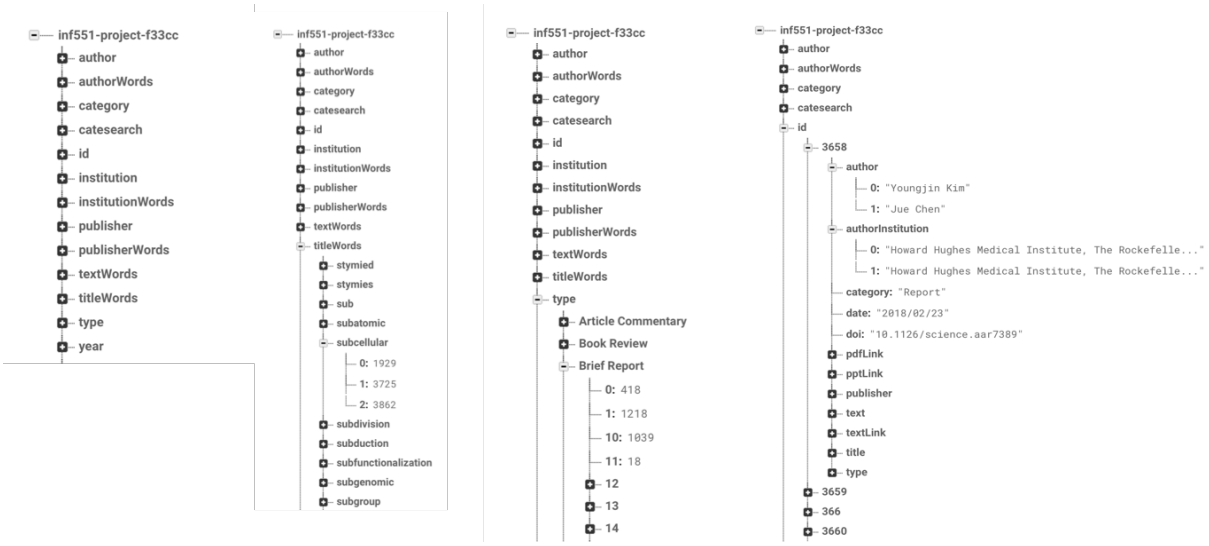

Figure 1. From left to right: **(1)** Root of database: each metadata type are presented as a key which is an entry for specific faceted searching;**(2)** Example of keyword. Here taking "subcellular" as an input, IDs like 1929, 3725, 3862 are retrieved for further paper searching. **(3)**Example of metadata, paper type. Here taking "Brief Report" as a key, IDs like 418, 1218, 1039 are retrieved;**(4)** With ID, one can search specific paper and its metadata. As an example, with ID 3658, a paper written by Youngjin Kim and Jue Chen is retrieved.

- **Processing IDs locally:** It's necessary to process these ID lists (can be one or more). Here two searching logic modes are provided for use. First is **"and"** logic: one can search papers that contain all words of *["example", "searching", "keywords"]*. In this situation, only papers containing all words are retrieved and program will compute the intersection of ID list. The other one is **"or"** logic: one can search paper containing any word of *["example", "searching", "keywords"]*. In this situation, all papers with any word are retrieved and program will compute the union of ID lists (duplication is removed at last). In addition, in this logic mode, papers with most searched keywords will be thought most related to the searching query and presented in the first. As an example, papers containing all words of *["example", "searching", "keywords"]* will be presented first and the one containing any two of them will be next. In this step, one desired ID list is obtained and sent to next step.
- **Search all metadata by IDs:** Each ID in the ID list can be a key for searching paper metadata. As shown in **Section 2.3** and **Figure 1**, metadata of one paper are stored as one JSON object. Therefore, in this step, a list of JSON objects are output and sent to next.
- **Show in Website:** Received object list contains all information about queried papers and will be presented in user interface. Details of this part will be in **Section 4: User Interface**.

3.2 Faceted Searching

Faceted searching is very like keyword searching except that faceted searching uses specific facet values. Also there are two logic modes:

- **"and" mode:** Only papers that satisfy all faceted requirements are retrieved. Example: while faceted requirements are 1. should be a research article;2. was published in 2017, only research articles published in 2017 can be shown as results.
- **"or" mode:** Papers satisfying any of faceted requirements are retrieved. Example: while faceted requirements are 1. should be a research article;2. was published in 2017, all research articles or papers published in 2017 are shown as results.

One can set multiple values for one facet like searching papers published in 2014 or 2015. There is only "or" mode for requirements in one facet since no paper can published in both 2014 and 2015 which means "and" logic mode will lead to empty result.

In addition, there is an "and" logic between keyword and faceted searching. Only those papers containing queried keywords and satisfying the faceted requirements can be show in website.

3.3 Specific conditions

Specific conditions are concluded in this part:

- **With only keywords (logic "and"):** Search papers containing all queried keywords.
- **With only keywords (logic "or"):** Search papers containing any one of queried keywords.
- **With only facets (logic "and"):** Search papers satisfying all faceted requirements. Like research papers published in 2017 only.
- **With only facets (logic "or"):** Search paper satisfying any of faceted requirements. Like research papers or all kinds of papers published in 2017.
- **With keywords and facets (logic "or" + "and"):** Search paper containing any one of queried keywords and satisfying all faceted requirements in the meantime.
- **Without keywords or facets (empty query):** Randomly give 10 papers in the database.

4. USER INTERFACE

In this program, we use ReactJS, which is based on node.js, to build our UI. User interface is composed of four different parts:

4.1 Sidebar

Sidebar is responsible for setting limitation. For example, users could find articles written by certain authors by choosing authors in sidebar. And they could also find articles written by certain author and written in certain year by choosing limitations. What is more, sidebar also would present the number of the result satisfying limitations chosen by user. See **Figure 2**.

4.2 Keywords Input

Users could input their keywords in to search input to find certain articles. Our search engine support search articles using multiple keywords. See **Figure 3**.

4.3 Logic Controller

Logic controller could control different logic. There are two controllers, for each controller, there are two options: or and and. For Keyword Controller, if the key words are k1 and k2, using or logic, the result would be articles which contains either k1 or k2. Using and logic, the result would be articles which contains both k1 and k2. For limitation Controller, if there are two limitations: written in 2014 and written by Tom, if the logic is or, the result would be articles whose author is Tom or articles written in 2014. If the logic is and the result would be articles whose author is Tom and written in 2014. See **Figure 4**.

4.4 Result Area

Result Area is the area to present search result. Each result would be in a card (as **Figure 5** Left). Each metadata of this article would be presenting in this card, like category, published date, publisher and abstract. Download button could redirect to the page showing pdf, Learn More Button would show full text of this article. See **Figure 5 Right**.

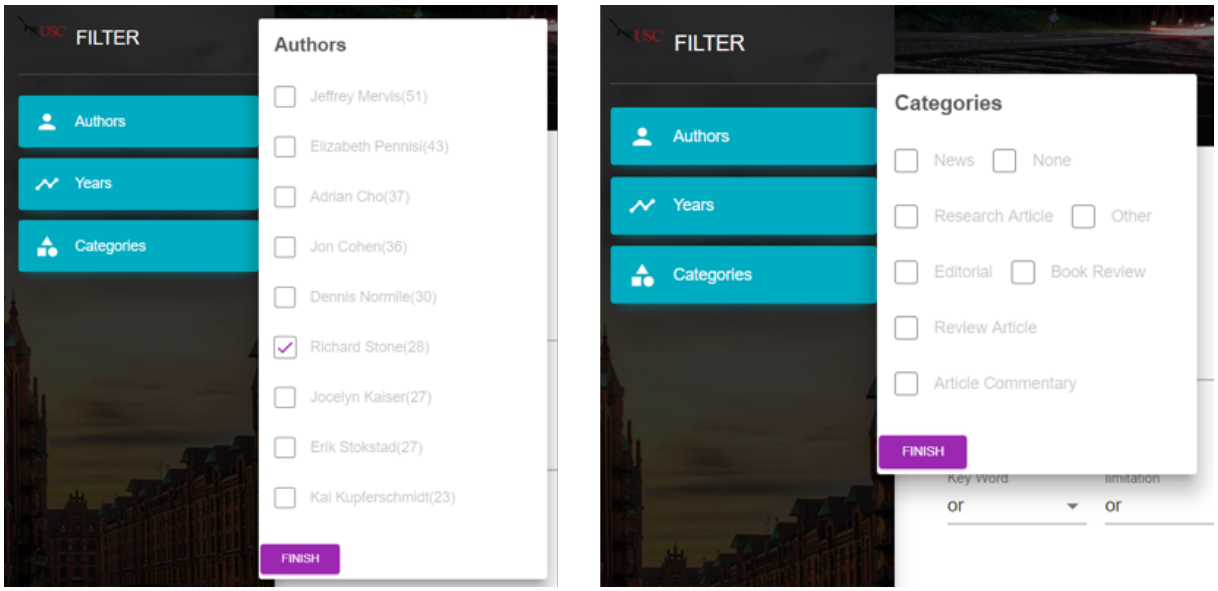

Figure 2. Sidebar

Figure 3. Keywords Input

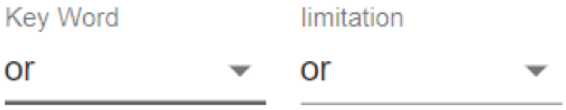

Figure 4. Logic Controller

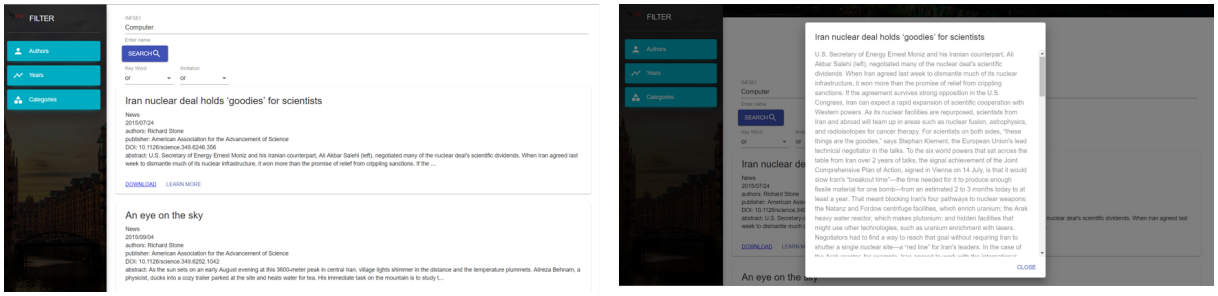

Figure 5. Result card (left) and full text area (right)

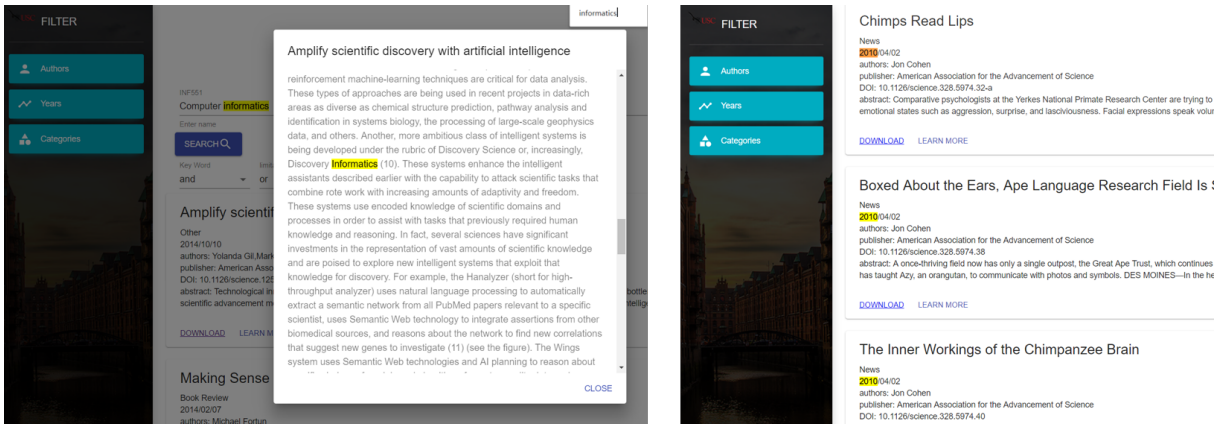

Figure 6. Keyword Search (left) and Faceted Search (right)

5. DEMO

5.1 Keyword Search

Typing keywords in the search bar, clicking search, the search engine would return articles which contains this keyword, for example, if the keywords are computer and informatics, the engine would return articles which contains computer and informatics, as **Figure 6 Left**.

5.2 Faceted Search

For example, if users want to find all articles written by John Cohen and written in 2010, users could set two limitations to search the result, as **Figure 6 Right**.

ACKNOWLEDGMENTS

It's the final report of group project in INF 551: Foundations of Data Management supervised by Professor Wensheng Wu. In the project, Wang Yuhan is responsible for back-end program and metadata preprocessing (**Section 2,3**) while Yan Juefei is responsible for user interface (**Section 4**).

Thanks to Professor Wensheng Wu and TA Weijun Deng for their suggestion and help during the project implementation.