IR Course Project Report

Alen Sugimoto & Mattia Monari

December 19, 2022

In this project, we created a web page in which users can search for free textbooks. The textbooks were crawled from 4 different websites.

Design and Implementation

Retrieving the Data

We first created the Scrapy spiders. We have one spider for every website we crawled. The websites we decided to crawl are

- `https://collection.bccampus.ca/search/`;
- `https://open.umn.edu/opentextbooks/`;
- `https://www.gutenberg.org/ebooks/bookshelf/`; and
- `https://openlibrary.org/search?subject=Textbooks`.

These websites were chosen for their large collection of free textbooks and their not-so-strict `robots.txt` file.

Each spider creates a JSON file containing a list of objects, each corresponding to one textbook.

There are some notable things about our process of creating the spiders:

- The BCCampus website had a button for going to the next page, instead of an anchor. So, only at most 6 textbooks were crawled for each subject in the website. For all the other websites, we were able to crawl all the textbooks.
- The Open Textbook Library website had restrictions in the `robots.txt` file on how quickly a spider could crawl the website. For all the data to be crawled without any bad requests, the `CONCURRENT_REQUESTS` variable in the `settings.py` file of our Scrapy project needed to be set to one, instead of the default of 16.
- The Gutenburg website contained more than just textbooks, so we chose to crawl only the textbooks within a group of manually selected subjects. Similarly, for the Open Library website, we crawled only the books under the subject `Textbooks`.

Indexing the Data

After crawling, the data was indexed with Solr:

```
> bin/solr start -e cloud
> bin/post -c <collection-name> <project-root>/data/*
```

We have a total of 24843 textbooks in the collection. Each textbook has the following fields:

- `title : str`   : The title of the textbook.
- `author : str`   : The author(s) of the textbook.

- `subjects : List[str]`: The subjects associated with the textbook.
- `description : str` : The description/summary of the textbook.
- `url : str` : The URL of the textbook page.

Since we want every query to consider, by default, all the fields that were indexed, a new field, named `all`, and 5 new copy fields were added such that the `managed-schema.xml` file contains the following lines:

```
...
<field name="all" type="text_general" uninvertible="true"
    indexed="true" stored="true"/>
...
<copyField source="author" dest="all"/>
<copyField source="description" dest="all"/>
<copyField source="subjects" dest="all"/>
<copyField source="title" dest="all"/>
<copyField source="url" dest="all"/>
...
```

Creating the UI

Finally, we created the user interface and added some features. The UI is coded with simple HTML, JavaScript, and CSS. The HTTP requests are sent to Solr using `XMLHttpRequest`. The following parameters are added to the request URL for every query:

```
["wt=json",
 "indent=on",
 "fl=url,subjects,description,author,title",
 "json.limit=100",
 "df=all"]
```

There was also a problem surrounding CORS when trying to send requests to Solr. It was solved by consulting this website: `https://laurenthinoul.com/how-to-enable-cors-in-solr/`.

We also implemented some extra features:

- Results presentation: The textbooks are presented to the user in a table. The `<table>` tag in HTML was used for the implementation.
- More Button: At most 50 textbooks are shown after a query search. Users then have the option to get another 50 textbooks using the button at the bottom of the page.
- Filtering: By default, the user's query searches the collection by all fields. They may also choose to search by either title, subjects, or author. We did this by using Solr's built in search format. For example, if the user chooses to search by author and enters the query `Mahoney`, the query string sent to the Solr server would be `author:Mahoney`.

- Results clustering: Users have the option to only see a specific cluster of textbooks after querying. The clusters are classified into subjects and users can select a cluster/subject in the results page. Also, the subjects are ordered such that the most relevant ones are displayed first.

User Evaluation

We got 4 users to evaluate our system. From the SUS feedback form given to the users after evaluation, the positive feedbacks were

- our system was simple and easy to use;
- there were no inconsistencies in the system; and
- the UI was useful in finding free textbooks.

However, we saw some issues in the system and received some suggestions from the users:

- When sending the query to Solr, the string did not replace any characters with escape sequences, resulting in bad requests to the server.
- The search bar became too small after results were presented.
- There was already a default query in search bar, which the users did not like. They preferred to have the search bar empty.

All the above issues were fixed after the user evaluation.

Other

We also decided to normalize subjects of the retrived texbooks using this python code:

```
import json
from cdifflib import CSequenceMatcher
import difflib

textbook_categories = ['Anthropology', 'Humanities', .... ,
                       'Technology', 'Zoology']
new_subj = []

def refac_json(filename):
    global new_subj
    with open('data/'+filename+'.json') as file:
        data = json.load(file)
        for e in data:
            for subject in e["subjects"]:
                tmp = difflib.get_close_matches(subject, textbook_categories, 1
                if len(tmp) > 0 and tmp[0] not in new_subj and subject:
                    new_subj.append(tmp[0])
                if len(new_subj) > 0:
                    e["subjects"] = new_subj
            new_subj = []

        with open('data/'+filename+'refactored.json', 'w') as new:
            json.dump(data, new, indent=2)


refac_json("bccampus")
refac_json("gutenberg")
refac_json("open_umn")
refac_json("openlibrarydata")
```

This was done because we had to many similar/insignificant subjects. This code, thanks to the help of the difflib package, replace the subjects of a textbooks with the closest one matched from a predefined array containing standard subjects.