Image Content Search Engine using Lucene

CS 242: Information Retrieval and Web Search
Department of Computer Science and Engineering
Bourns College of Engineering
University of California, Riverside

Karthik Harpanahalli
kharp009@ucr.edu

Varun Sapre
vsapr002@ucr.edu

Hoora Shobani
hshob002@ucr.edu

Ameya Padole
apado003@ucr.edu

Sarthak Jain
sjain050@ucr.edu

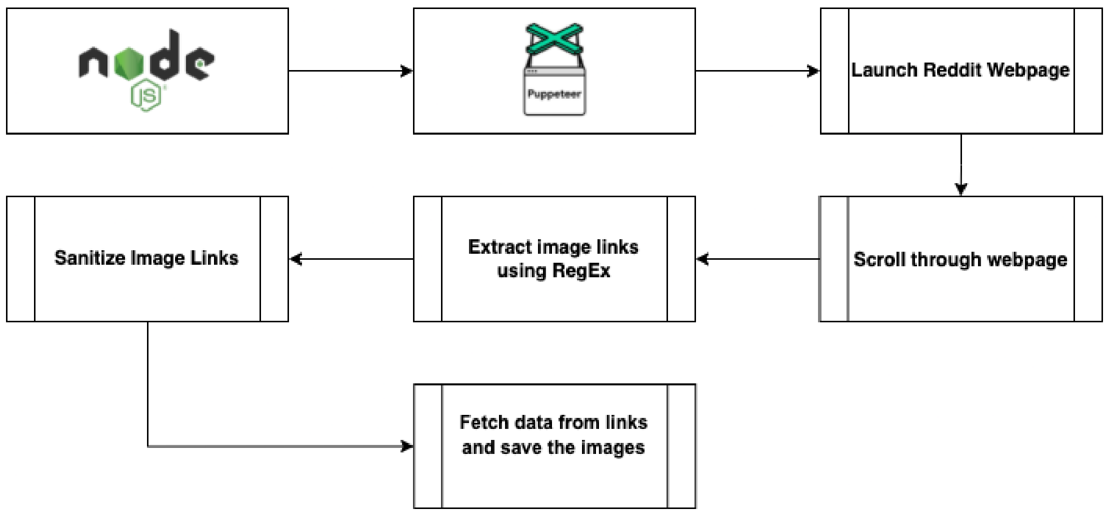
Introduction:

We designed and developed an image search engine using Lucene indexing which returns the images pertaining to the input query by the user. We employ Google Vision API to fetch the relevant objects in the image, and use the indexed keywords to match with the query. Lucene then ranks the documents using an inverted index. We aim to use this ranking to display the images according to the ranking on our front-end.

1. **Collaboration Details: Description of the contribution of each team member.**
   a. Ameya Padole - Data Cleaning, Documentation and Scripting.
   b. Hoora Sobhani - Lucene setup and fixing JSON dependency issues. Lucene Indexing and tracking performance metrics.
   c. Karthik Harpanahalli - Implementing Object Detection E2E pipeline using FiftyOne & Reddit Scraper on NodeJS using Puppeteer, Data Cleaning and Scripting.
   d. Sarthak Jain - Data Cleaning and Documentation of the entire project.
   e. Varun Sapre - Researched object detection API, dataset and tools. Setup initial foundation of the reddit image scraper in Python.

2. **Overview of the scraping system:**

   a. **Architecture.**



   b. **The Scraping Strategy.**

1. We use NodeJS to scrap images from Reddit's **r/itookapicture** subreddit for high resolution pictures.
2. We make use of Puppeteer node module. It allows use to control Chromium using the DevTools. This runs the automated code, launches the browser opens a new tab, navigates to the reddit link and performs scrolling whose value can be changed in the code. The scrolling handles the infinite scrolling of reddit by fetching the window object and scrolling to the body height.

```
// New instance of the chromimum in headless mode
const browser = await puppeteer.launch({
headless: false
});

// Create a new tab in the browser
const tab = await browser.newPage();

// Change the navigation timeout to avoid timeout
await tab.setDefaultNavigationTimeout(0);

// Navigate to reddit
await tab.goto('https://www.reddit.com/r/itookapicture/top/?t=all');

//Change the following limit to the number of times the scrolling is to be done
for (let j = 0; j < 100; j++) {
 await tab.evaluate('window.scrollTo(0, document.body.scrollHeight)');
 await tab.waitFor(1000);
 }
```

3. Using windows object we fetch the innerHTML of the body and pick up the src attribute of the <img> elements using Regular Expression `/src="https:\/\/preview.redd.it\/(.*?)"/g`

   ```
   //Regex is used to filter the entire HTML element to find links that are in the format
   //src = "https.preview.redd.it/...". There is probably a better and more efficient way
   //to do this, but should do for now for the IR project
    const regex = /src="https:\/\/preview.redd.it\/(.*?)"/g
    let matches = bodyHTML.matchAll(regex);
    let counter = 0;
   ```

4. The src attributes of the images contain other metadata that is not needed and need to be cleaned. We sanitized the links and extracted only the image link.

   ```
   //We split the matched string to remove the preview.redd.it part in the link and replace it with
   // "https://i.redd.it/e14dvx95vpg81.jpg" as the previous link throw a 403 forbidden error.
   // THERE IS PROBABLY A MUCH BETTER WAY TO DO THIS BUT IN INTEREST OF TIME, THIS SHOULD DO FOR THE PROJECT.
   let processedString = match[0].split('src="https://preview.redd.it/');
   processedString = processedString[1].split('?');
   counter++;
   imageArray.push("https://i.redd.it/" + processedString[0]);
   ```

5. We then fetch images from the links stored in the JSON file and store the images.

3. **Overview of Object Detection from images:**

The objective of this stage is to detect objects from the images scraped along with COCO dataset. We are using FiftyOne to achieve this. FiftyOne is an open-source tool for loading datasets and annotating them. We use Fifty one here to annotate images scraped from reddit and the COCO 2017 dataset export them to a JSON file. The attached .ipynb file is self descriptive and instructions to run all the commands. It also the general structure of the pipeline.

Install FiftyOne

FiftyOne is an open-source tool for loading datasets and annotating them. We use Fifty one here to annotate images scraped from COCO 2017 dataset and export them to a JSON file.

Load COCO Dataset

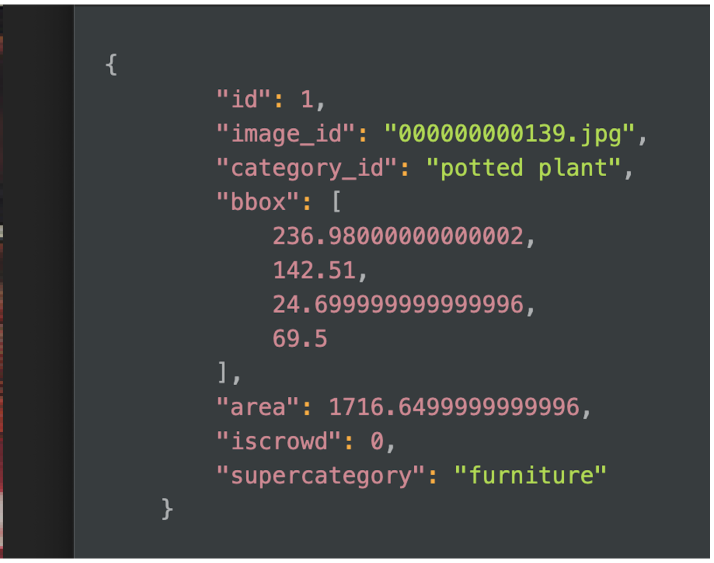
Loads necessary libraries and dataset along with images that have been scrapped by the reddit scraper implemented using NodeJS

FiftyOne UI [Optional]

To launch the UI provided by the FiftyOne to see how the tool is annotating images, run the commands below the section in .ipynb file provided as part of the code.

Generate Label

The output is a JSON file in the /path/ folder. A sample json annotation is shown below. The JSON output is described in the .ipynb file including the fields and annotation section. The JSON contains annotation for all the images that describes the content in the images.



```
{
        "id": 1,
        "image_id": "000000000139.jpg",
        "category_id": "potted plant",
        "bbox": [
            236.98000000000002,
            142.51,
            24.699999999999996,
            69.5
        ],
        "area": 1716.6499999999996,
        "iscrowd": 0,
        "supercategory": "furniture"
    }
```

4. **Overview of the Lucene indexing strategy:**

Our Lucene indexer is implemented based on the `org.apache.lucene` Java library which is robust and well-documented. The input of our indexer is the JSON file which is the output of our crawler with multiple modifications to be easy to be parsed. The main implementation of our indexer is in Indexer.java. We will elaborate on each part of the implementation of `class Indexer` in the following section:

Reading the Input

The following method reads the JSON file as an input and parses it and converts it to an array. To parse the JSON objects, we used `org.json.simple` Java library which caused dependency issues. First, we tried to build the project with Maven and add the dependencies to pom.xml, but the `mvn` was not installed, so we used `javac`, and to solve the dependencies, we manually changed the class-path every time and attached the required `jar` file (More details are in the instruction section).

```
private static JSONArray readFile(String filePath)
throws FileNotFoundException, IOException, org.json.simple.parser.ParseException {
        JSONParser jsonParser = new JSONParser();
        FileReader reader = new FileReader(filePath);
        Object file = jsonParser.parse(reader);
        JSONArray annotations = (JSONArray) file;
  return annotations;
}
```

Setup the indexer

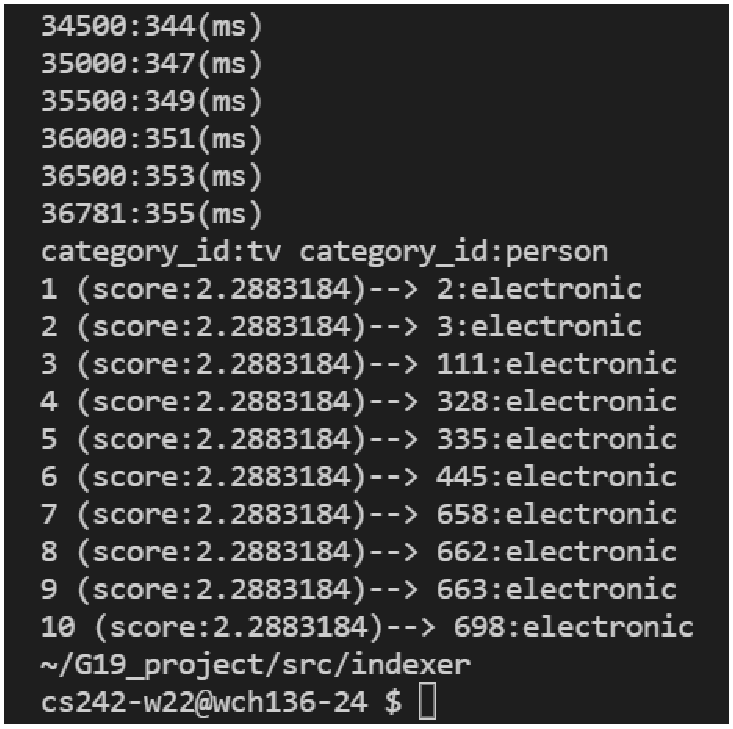
In the main method, we created the indexer. Our indexer exploits `StandardAnalyzer` which uses `StandardTokenizer`, `LoweCaseFilter`, and `StopFilter` to parse and index documents. Then, we created the directory object on the storage (not RAM) which stores the indexed data on the `data` directory which is in a parallel directory with `src`. Then, we created an `indexWriter` to index the data.

```
Analyzer analyzer = new StandardAnalyzer();
//Store the index in memory:
//Directory directory = new RAMDirectory();

//To store an index on disk, use this instead:
Directory directory = FSDirectory.open((new File("../../data/")).toPath());
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter indexWriter = new IndexWriter(directory, config);
```

Parse and index the data

In this part, we considered each annotation as a document and indexed the fields which we thought would help our designed search engine. The implementation of parsing the data and indexing them is attached to the project. To make sure that the indexer works correctly, we implemented a small search test that verifies our correct implementations. The codes of the test searcher are in the main method. You can see the search result for a “tv person” query in the following figure.



```
34500:344(ms)
35000:347(ms)
35500:349(ms)
36000:351(ms)
36500:353(ms)
36781:355(ms)
category_id:tv category_id:person
1 (score:2.2883184)--> 2:electronic
2 (score:2.2883184)--> 3:electronic
3 (score:2.2883184)--> 111:electronic
4 (score:2.2883184)--> 328:electronic
5 (score:2.2883184)--> 335:electronic
6 (score:2.2883184)--> 445:electronic
7 (score:2.2883184)--> 658:electronic
8 (score:2.2883184)--> 662:electronic
9 (score:2.2883184)--> 663:electronic
10 (score:2.2883184)--> 698:electronic
~/G19_project/src/indexer
cs242-w22@wch136-24 $ 
```

Fields in the Lucene index:

Thinking about how we want to search among the images and their annotations at the end of the project, we decided to enable searching based on the images' names, categories, and super-categories. Therefore, we indexed each annotation as a document and stored the id, image_id, category_id, and super-category related to them. A sample of each document is as follow:

```
{
        "id": 1,
        "image_id": "000000000139.jpg",
        "category_id": "potted plant",
        "bbox": [
            236.98000000000002,
            142.51,
            24.699999999999996,
            69.5
        ],
        "area": 1716.6499999999996,
        "iscrowd": 0,
        "supercategory": "furniture"
    }
```

Considering our documents, not only we can search images by their name, category, and super-category, we also can enable vertical search on super-categories, or we can add filtering by size, area, etc.

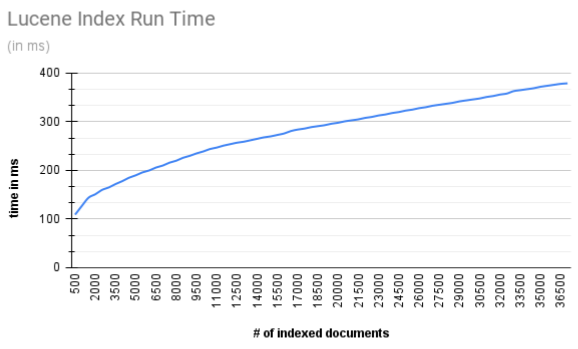
Text analyzer choices:

Firstly, we used JSONParser to analyze the data gathered from our crawler. Then, considering the mentioned fields that we indexed, we used the standard analyzer of the Lucene. The `StandardAnalyzer` uses `StandardTokenizer`, `LoweCaseFilter`, and `StopFilter` (The default list of stop words for Standard analyzer is {"a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"}).

Also, we indexed the mentioned fields as `TextField`-type so that we enabled prefix matches and token (potentially prefix) matches, and we avoided using `StringField`-type which is usually used for exact matches.

Run time of the Lucene index creation process:

As can be seen in the following figure, by increasing the number of documents, the required time for indexing increases. The entire indexing for 36781 documents took 379 ms.



5. **Limitations of the system :**

The only limitation we encountered while scraping was that we were unable to access subsequent links from the reddit webpage r/itookapicture. As a consequence our extracting of the images was limited to a single page with infinite scroll. This limitation arose primarily due the design of reddit.com, r/itookapicture subreddit's design. The webpage is designed such that all images are displayed on a single webpage.

6. **Obstacles and solutions:**

   1. Infinite Scroll problem while scraping Reddit :

      While scraping for images on Reddit, the primary challenge we faced was to make the crawler scroll to the next image so that its URL can be extracted such that it won't require further sanitization. To achieve this we scrolled to the current body of the page and extracted the URL of the images.

   2. Preprocessing of JSON to include category and super-category names

      Changed json annotation format to accommodate category. Converting our JSON input file more readable for the Indexer by replacing the object ID and category ID with the values instead of IDs.

   3. While some difficulties were faced during installation and setup of Lucene on our Local system we resolved it by utilizing Lucene on the bolt server.

   4. Since we did not have the root access, we were unable to import certain necessary libraries and jar files onto the bolt server. We resolved this dependency issue by directly mentioning the class-path during the compilation.

   5. Before indexing the data, we needed to design our desired future search engine and its behavior to know which fields should be indexed to avoid indexing unnecessary data. In this way, we expect our searcher implementation for the second phase of the project would be more strait-forward.

   6. To use the FSDirectory, we faced a little challenge because we were using a deprecated API of Lucene. By checking the version of the installed Lucene on our assigned machine, we updated our used APIs.

7. **Instruction on how to deploy the crawler.**

   a. Extract Reddit_Crawler_Sanitizer

   b. Run scraper.sh

   The script will install dependency packages and run code. It will launch chromium in headless mode and it starts scraping images. The images are stored locally.

8. **Instruction on how to build the Lucene index:**

   To develop our indexer, we exploited the Java-based Lucene. To avoid the complexity of the Lucene installation, we utilized the Lucene provided on the WCH136-24 machine. To access the platform, as mentioned in the demo session, we took the following steps:

   ```
   $ ssh netid@bolt.cs.ucr.edu

   *Then on bolt server*
   $ cs242_login
   ```

   First, to test Lucene we executed the `TestLucene` program which is located in the `G19_TestLucene` directory. Then, we developed our indexer (located in the `G19_project` directory) that receives the outputs of our crawler (`json` files) as inputs. To resolve the dependencies and to compile and execute the project, we need the following steps:

   Dependencies:

   ```
   $ indexer_dependency=/opt/home/cs242-w22/lucene-8.11.1/core/lucene-core-8.11.1.jar:
   /opt/home/cs242-w22/lucene-8.11.1/queryparser/lucene-queryparser-8.11.1.jar:
   /opt/home/cs242-w22/lucene-8.11.1/analysis/common/lucene-analyzers-common-8.11.1.jar:
   /opt/home/cs242-w22/lucene-8.11.1/demo/lucene-demo-8.11.1.jar:./json-simple-1.1.1.jar:.
   ```

   Compile:

   ```
   $ javac -cp $indexer_dependency Indexer.java
   ```

   Run:

   ```
   $ java -cp $indexer_dependency Indexer
   ```