**Introduction**

As described in the project proposal abstract, we will compare the performance between Generative Entity Retrieval(GENRE) and Elasticsearch-based standard full-text search methods. We will use the Python documentation as the knowledge base, documentation sections as documents, and StackOverflow-like questions as queries.

**Basic idea**

As learned in our course, the performance measurement of text information retrieval is inclined to be subjective. So, we decided to use the StackOverflow answer set as a golden ruleset with the below idea.

* The StackOverflow has a perfect answer for the question by human voting, which can be considered a golden ruleset.
* Moreover, many of the most voted answers include the external link to extend their explanation.

We paid attention to the embedded link in the answer. We found a tendency that the StackOverflow answer is a summarization of the embedded link, especially when the embedded link is the language reference documentation.

From that perspective, we had an idea that the search engines could extract the same link as the StackOverflow answer against the same StackOverflow question if they had a whole language reference set and searched them. If the search engine is powerful enough, that will extract the same language reference link as the StackOverflow answer in a high rank against the same question.

**Summary of the design and Implementation**

From that perspective, we implemented codes and solutions as below.

* Crawl data from the Python reference doc
* Crawl data from the StackOverflow
* Implement a search repository in the ElasticSearch
* Implement a search repository in the GENRE
* Query questions from the StackOverflow to the ElasticSearch and retrieve the reciprocal rank

And, below is a diagram for actual implementation. It is interesting to compare with the conceptual diagram in the project proposal stage. That shows the gap between the first estimation and actual implementation, which also implies our challenges during the project.

Diagram

Description automatically generated

**Figure 1. The actual implementation diagram**

Chart, treemap chart

Description automatically generated

**Figure 2. The conceptual diagram in the project proposal stage**

**Component explanation (Code, solution)**

**Crawl data from the Python reference doc (scraper-reference.py)**

That is the first step to obtaining the reference dataset. Technically, that is a website crawling, and we used Selenium and BeautifulSoup for web scraping.

That code is an extension of the assignment MP2.1 in our course. We reused some of the functions in that assignment, applied a similar workflow to retrieve the web page.

* Step 1: Collect links from the TOC (div class="toctree-wrapper compound") and get the URL link and title (class="reference internal")
* Step 2: Analyse the document structure in the collected links and parse anchor and header links
* Step 3: Crawl text from links and result in a JSON format that includes title, document URL, and text content object.

There is, however, a difference between that code and the assignment. The assignment focuses on the extensive page retrieval across the link, but the objective of the code is a refined dataset to enable effective search in the ElasticSearch.

From that perspective, one of our critical goals is to slice the retrieved documents based on the anchor link (e.g., <https://docs.python.org/3/library/stdtypes.html#bytearray.join>). Moreover, one of the considerations was to determine whether to remove top-level documents. (e.g., [https://docs.python.org/3/library/stdtypes.html](https://docs.python.org/3/library/stdtypes.html#bytearray.join))

It is reasonable enough to leave top-level documents. Especially, that will be mandatory in production service to display the last resort page for user convenience. But we are conducting an academic project, therefore finding the relevance is a more critical factor. In that perspective, it will be a rather more reasonable approach not to display if there is no relevant result. That is the reason we decided to remove the full top-level documents. So we decided not to include the top-level(full) documents.

**Crawl data from the StackOverflow**

<Please fill out>

**Implement a search repository in the ElasticSearch**

**Format adjustment through a separate code (ElasticSearch\_import.py)**

Although we could retrieve data from the target website, we had to take another step for format adjustment to enable bulk import to ElasticSearch. We expected that the ElasticSearch could import any type of JSON data but realized it requires a specific JSON format.

* An ‘Index’ object requires for each data entry as a predefined metadata
* The data field does not allow separation. All data fields should be in a single line.

So, we have written another code to adjust the format to enable import. The function of the code is to add index ID and consolidate data into a single line.

Below is an example of the adjustment. The source JSON file only includes text files, and each field is in a separate line as below.

{

"text": "This exception is raised when the server unexpectedly disconnects, or when an attempt is made to use the SMTP instance before connecting it to a server. ",

"title": "debugging cgi scripts",

"url": "https://docs.python.org/3/library/smtplib.html#smtplib.SMTPServerDisconnected"

},

ElasticSearch\_import.py adjusts the file to enable import to the ElasticSearch. The adjusted file includes the index field and data fields consolidation as below.

{"index": {"\_index": "reference", "\_id": 9370}}

{"text": "This exception is raised when the server unexpectedly disconnects, or when an attempt is made to use the SMTP instance before connecting it to a server. ", "title": "debugging cgi scripts", "url": "https://docs.python.org/3/library/smtplib.html#smtplib.SMTPServerDisconnected"}

**Construction of the ElasticSearch and data import**

The ElasticSearch is a product that can work as it is. But, installation and setup is another big topic separate from code development. With those considerations, decisions, and implementations, the ElasticSeach instance could be set up on Windows Azure (13.82.107.120 port 9200).

* We decided to set up a plain ElasticSearch to a single blank Linux instance in Windows Azure. Since that was installed in a single node, clustering was not considered.
* Kibana was helpful for development and testing but not mandatory to get a required reciprocal link. So we decided not to install Kibana into a production instance.
* We decided to use the default text retrieval function. We also considered the machine learning feature in Kibana, but that was a commercial function.
* Setting up a Windows Azure was another separate topic. We assume a single C class internal network and a single server with public IP. The port security was also considered to allow only port 9200, the default port for ElasticSearch access.
* But, ElasticSearch authentication was intentionally ignored because configuring a native realm did not work as expected. That is still one of the open items.

Finally, the below command starts to import the crawled and adjusted data into the ElasticSearch.

curl -X POST "13.82.107.120:9200/reference/\_bulk?pretty" -H "Content-Type: application/json" --data-binary @reference\_doc\_import.json

**Implement a search repository in the GENRE**

<Please fill out>

**Query questions from the StackOverflow to the ElasticSearch and retrieve the reciprocal rank (ElasticSearch\_query.py)**

This code aims to query the StackOverflow question to the ElasticSearch and compare the resulting URL(s) between the ElasticSearch and the most voted answer. Technically, that querying, retrieval, and comparison consist of the steps below.

* Step 1: Retrieve the collected StackOverflow data(StackOverflow-entries.json) and extract questions and answer link(s)
* Step 2: Query the extracted questions to the ElasticSearch and get the result set
* Step 3: Compare the StackOverflow answer link and ElasticSearch resulting URL, and calculate the reciprocal rank (Which rank is the StackOverflow URL in the ElasticSearch URL result set?)
* Step 4: Draw the table and historgram of reciprocal rank

Below is a short explanation of the technology and library used in the code

* To handle JSON, the JSON library is extensively used in the code.
* Moreover, the Python Elasticsearch Client library was very helpful in accessing the ElasticSearch in a simple and straightforward manner.
* Finally, the matplotlib library was used to draw the histogram of the reciprocal rank.

One consideration of the code is setting the scope of the valid reciprocal rank. We set 200 as the limit and only get up to 200 results from the ElasticSearch by setting the ‘size’ parameter to 200. We are aware that people generally focus only on less than the first 50 search results, but also tried to reflect that the experienced professionals do not fully trust the search engine ranking. They believe first ranked pages are manipulated by the optimization techniques led by commercial purposes, thus rather trust mid-rank pages. That is the reason that we set 200 results as the limit.

**The analysis of the result**

The result was pretty interesting. We tested with 1172 StackOverflow questions, and the ElasticSearch matched 320 answers, with approximately 27.3% accuracy.

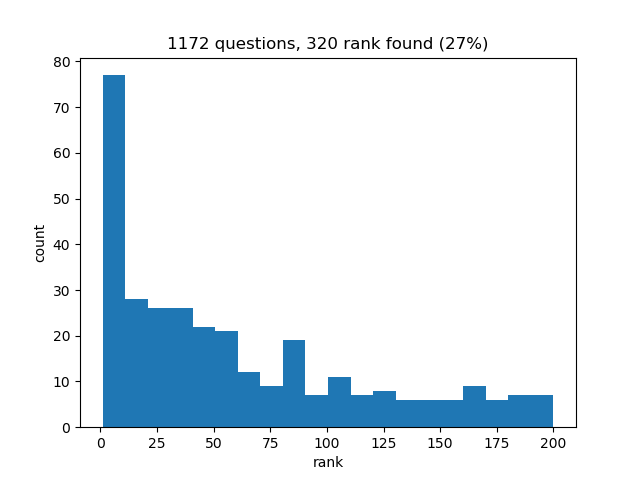
As seen in the table and histogram below,

* The distribution is extremely right-skewed.
* Rank 1-20 consists of 33% of the entire result. Then another 67% has a gentle slope.
* The accuracy means the proportion of the ranked set, and the ratio means the proportion of the entire StackOverflow question set. From the ratio perspective, rank 1-20 data consists of 9% of the whole question set.
* Considering the general public behavior(searching for less than 50-60 pages), more than 69% of the ranked data set is between 1 to 60. But, they are still 17% of the entire question set.

The result implies that the ElasticSearch offers the accurate result when discovered, but the accuracy against the entire question set is pretty low (27%).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rank** | **Count** | **Accuracy** | **Cumulative Accuracy** | **Ratio** | **Cumulative Ratio** |
| 1-20 | 105 | 33% | 33% | 9.0% | 9.0% |
| 21-40 | 52 | 16% | 49% | 4.4% | 13.4% |
| 41-60 | 43 | 13% | 63% | 3.7% | 17.1% |
| 61-80 | 21 | 7% | 69% | 1.8% | 18.9% |
| 81-100 | 26 | 8% | 77% | 2.2% | 21.1% |
| 101-120 | 18 | 6% | 83% | 1.5% | 22.6% |
| 121-140 | 14 | 4% | 87% | 1.2% | 23.8% |
| 141-160 | 12 | 4% | 91% | 1.0% | 24.8% |
| 161-180 | 15 | 5% | 96% | 1.3% | 26.1% |
| 181-200 | 14 | 4% | 100% | 1.2% | 27.3% |

**Table 1. The ElasitcSearch result of the StackOverflow question set**



**Figure 3. The resulting histogram of the ElasticSearch reciprocal rank**

<Please fill out the result of GENRE and comparison between the ElasticSearch and GENRE>

**Lesson learned, and open items**

Same as other technical projects, challenges and open items are our friends, and we could learn more through those challenges and open items.

I want to explain complexity management as a key takeaway among many challenges. It is common that the actual implementation is different from what we thought in the early stage, and our project was not an exception.

We were aware that our project would be challenging with a complicated implementation. However, we found unexpected hidden obstacles during the project, such as reformatting and finding the right libraries for development.

From that perspective, one of our primary challenges was managing complexity within a given timeline. We had to balance completeness and release on time, and some items had to be left open. But, that is also an important lesson learned that we finally achieved an acceptable result against those challenged.

**Open items**

As explained above, some items had to open during the project for complexity management. Below are some of them.

* Executable binary of the Python code: We understand the importance of executable files. If this is an enterprise or commercial IT project, they are mandatory for end users. But we decided not to offer them for those reasons.
  + We should consider multiple OS, including Windows, macOS, and Linux.
  + Python is basically an interpreter-based language. Therefore, compiling Python to an executable file is not perfect, especially in macOS and Linux environments.
  + Our target users are colleague students in this course. We expect all of us are now familiar with using Python language.
* Authentication of the ElasticSearch: That was another open item. Security is now the baseline of a modern computing environment. We enhanced the security of the Windows Azure environment that the ElasticSearch located.

However, we finally decided not to implement authentication of the ElasticsSearch. We tried the basic authentication methodology (described in <https://www.elastic.co/guide/en/elasticsearch/reference/current/native-realm.html>), but that did not work as expected. Under that obstacle, we decided that it is more important to prove our functionality than security implementation because we are in a project, not actual production. If we could have enough time, we could have an opportunity to address that topic.