EECS-767 Information Retrieval Project

Final Report

Prof. Bo Luo

Submitted by

Sundeep Kumar G-2844394

Jyothi Prasad P S-2853401

Nazma Sulthana K-2828884

CONTENTS

Part-A

EECS-767 Information Retrieval Project Final Report

Sundeep Kumar G - 2844394
Jyothi Prasad P S - 2853401
Nazma Sulthana K - 2828884

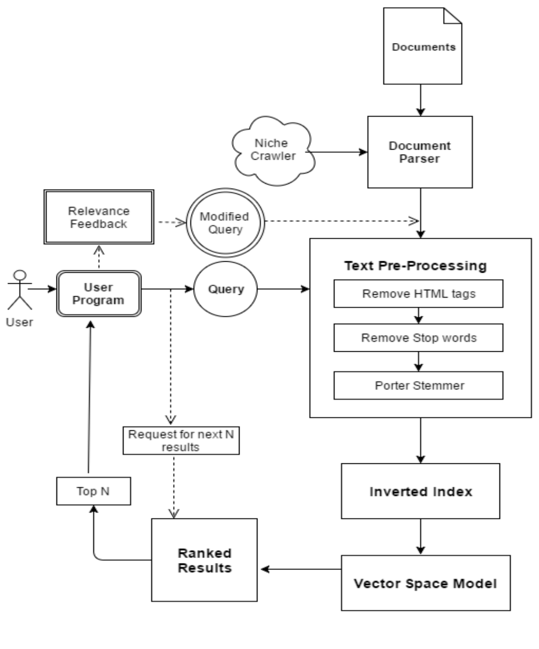
***Abstract*—The purpose of this document is to give an overview of academic project on information retrieval (IR) of unstructured content. Unstructured content typically includes text, music, video, or still images. This evaluation focuses on the retrieval of unstructured text. This project targets at ranking and retrieval of documents mainly based on the similarities between Query Vector and Document Vector.**

I. DELIVERABLES

The initial goal is to crawl through a static set of documents, retrieve and display the documents based on calculated page rank. The next step is to implement a Niche Crawler which crawls through the desired web pages and further evaluate the page rank based on our ranking criteria. To enhance the performance of the system we will include relevance feedback to our application. The selected criteria for evaluation of the final project includes, Precision Recall Values, Response Time, User Effort, Form of Presentation and Content Coverage.

II. PROJECT PLAN

A. Basic Architecture



The basic architecture of the targeted project on information retrieval domain is shown in the given block diagram. The entire project can be implemented in the following steps.
Step-1: Collecting the documents and parsing them using a document parser.
Step-2: Pre-Processing the text. This includes removal of HTML tags, stop words and implementing stemming (Porter Stemmer) algorithm.
Step-3: Creating an Inverted Index. The output includes, terms, term frequencies, file names for each term (Doc Id) and document frequencies.
Step-4: Designing Vector Space Model where Tf-Idf weights and document vectors for each document are calculated and the results are written into an external file.
Step-5: Constructing a user program which takes in the required query from user and parse it.
Step-6: Processing the user query, generating the query vector, calculating the cosine similarities for each document and query vector, ranking the documents and displaying the ranked results.
Step-7: Designing and developing Web crawling for a specific domain of interest.
Step-8: Enhancing the search by implementing relevance feedback(Rochhio Algorithm) and Term Proximity.

B. Programming Platform

We chose Java as our programming platform for a number of reasons.It's Object Oriented Programming helped us to built modular programs and reusable source code.Platform independent feature of Java made this application to run on any machine.Other very important reason to choose Java is it's programmer friendly API. For instance, we used an API called Files API which is introduced from Java version 1.7, is a very efficient and simple API to parse through files in the given file path. The Files API was further enhanced in Java Version 1.8 by adding additional methods for better File parsing. (Eg:Walk)

For a local Server we chose open source and light weighted Apache tomcat 7,an HTTP server for running Java applications that can serve Java Servlets and JSP.

III. Flow of Project Implementation

1) We started the application development with the parsing of document repository and extracting tokens.
2) We further concentrated on complete pre-processing which includes removal of HTML Tags, stop words and stemming.
3) The next task was to build an inverted index to calculate the term and document frequencies for which we used the data structure from Java's collection API in java.util.package.
4) Completed construction and implementation of Vector-Space model. We have constructed the required skeleton and methods to calculate the weights of the terms and integrated with inverted index model. We had built the index and tested the integrated vector space model for its correctness.
5) Web Crawling functionality is incorporated using multi-threading.
6) In addition, we also developed an user interface to fetch the user query and display the ranked results.
7) Worked on design and implementation of Relevance Feedback using Rocchio algorithm.
8) Implemented Term Proximity concept by making necessary changes to the required data structures.Hence the links retrieved after query processing will be based both on Cosine Similarity and Term Proximity

A. Document Parsing

For parsing through a set of documents, we tried to incorporate a new method representation present in the latest version of Files Package(java.nio.file.Files). This package contributes a method called "Walk" which returns a Stream by walking through the file tree rooted at a given starting file,which was introduced in JRE version 1.8. The file tree is traversed depth-first. For scanning through the contents of individual files, we used a traditional approach of invoking Scanner Class.

Method Representation:

$$walk(start, Integer.MAX\_VALUE, options);$$

$$public\ static\ Stream < Path > \ walk(\ Pathstart,\ FileVisitOption..options)\ throws\ IOException;$$

B. Text Pre-processing

1. As first step of pre-processing, we used pattern matching technique to remove all HTML tags while parsing through the document.

Method Representation:

$$String < Variable >= < Object > .toString().replaceAll(\ "Pattern\_Matching...");$$

2. The next step of pre-processing is to filter the stop- words from the available tokens. For this, we used "exude" library from GIT Hub which parses through the tokens and identifies the predefined set of stop words and eliminates them.

Sample invocation of Stop words API in our application is as below:

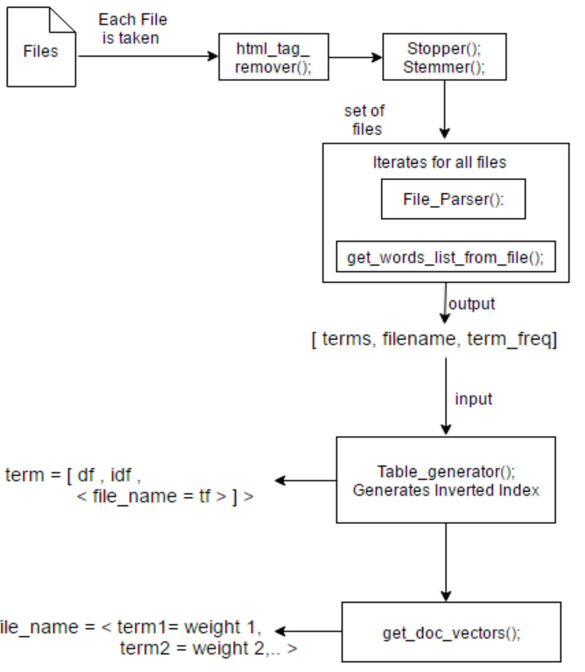
$$scrawlWords.filterStopingWord(word.trim(), filteredWords);$$

3. After Stop words removal, the next step of pre-processing is Stemming. For this we used Porters Algorithm where the stemming flow is done in five different stages. The implemented Stemmer Class transforms a word into its root form based on predefined rules.

C. Inverted Index

The construction of inverted index involves usage of basic data structures like HashMaps, ArrayLists from Java's Collection API in "java.util" package. We have also used HashSets to remove duplicates from the ArrayLists, and TreeMaps to sort the HashMaps.

Flow Diagram:


Files
Each File is taken
html_tag_ remover();
Stopper(); Stemmer();
set of files
Iterates for all files
File_Parser();
get_words_list_from_file();
output
[terms, filename, term_freq]
input
Table_generator(); Generates Inverted Index
< term = [df , idf , < file_name = tf >] >
get_doc_vectors();
<file_name = < term1= weight 1, term2 = weight 2,.. >

The above flow diagram explains the process of generating inverted index. After document parsing and performing text pre-processing, we get a set of processed files which are fed to a method $'file\_parser()'$. By iterating through all the files, $file\_parser()$ using another method $'get\_words\_list\_from\_file()'$ produces the output in the form of a list $[terms, filename, term\_freq]$.

This list is passed as input to an important method $'Table\_generator()'$. This finally generates the required inverted index.The data structure of the inverted index is as mentioned below:

Data Structure:

```
HashMap < String, List > {
    String term = ArrayList [
    Integer doc_freq,
    Doubleidf,
    HashMap < String, Integer > {
        String filename = Integer tf
        }
    ]
}
```

This inverted index is given to a method $'get\_doc\_vectors()'$ which further generates the document vectors for all the documents that are parsed.The format of document vector is shown in the above flow diagram.The generated inverted index and the document vectors of all the files are written into external files respectively. In this process we could finally implement inverted index.

Formula for calculation of IDF

$$idf_t = \log_{10} \frac{N}{df_t}$$

Formula for calculation of Weights

$$w_{t,d} = tf_{t,d} * idf_t = tf_{t,d} * \log_{10} \frac{N}{df_t}$$

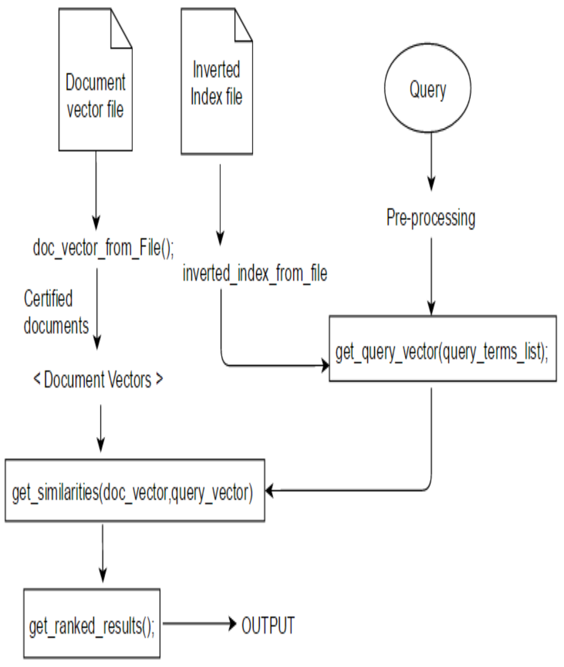
Here N is number of documents, df is the document frequency and tf is the term frequency.

D. Vector Space Model

In this module, for the construction of Vector space model, we used Array Lists Data Structure to construct Document and Query vectors.

The above flow diagram illustrates the design of Vector Space Model and Query Processing.When the query is entered by the user, it is also pre-processed by implementing stemming and stopping, similar to that of documents. This processed query long with the stored inverted index file is given to a method $'get\_query\_vector()'$. Here from the entire inverted index file only those indexes that belong to the terms in the query are picked and finally $get\_query\_vector()$ produces query vector as the output.

Flow Diagram:



The above flow diagram illustrates the design of Vector Space Model and Query Processing.When the query is entered by the user, it is also pre-processed by implementing stemming and stopping, similar to that of documents. This processed query long with the stored inverted index file is given to a method $'get\_query\_vector()'$. Here from the entire inverted index file only those indexes that belong to the terms in the query are picked and finally $get\_query\_vector()$ produces query vector as the output.

Next the produced query vector and the document vectors of only certified documents are given to a method $'get\_similarities(doc\_vector, query\_vector)'$. Here Cosine Similarities for the query vector and all the certified document vectors are calculated. The calculated similarities are fed to another method $'get\_ranked\_results()'$ where based on the similarities, ranking of the certified documents is calculated and this output information is given to a method calling the user interface thus finally displaying the ranked results to the users.

Formula for calculation of Cosine similarities:

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \bullet \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

Data Structure:

The Data Structure of Vector Space model is as mentioned below:

$HashMap < String, HashMap > \{$
$String \quad file\_name,$
$HashMap < String, Double > \quad \{$
$String\ term,$
$Double\ tf\_idf\_weight,$
$\}$

$\}$

E. Web Crawling

In order to implement web crawler functionality we started with a single spider which crawls through pre-defined number of web pages and stores the web page contents in our local directory which are further fed into our inverted index module. To incorporate the parallel processing functionality to our web crawler, we utilized multi-threading concepts of Java and enhanced our web crawler to an threaded environment to run multiple spiders simultaneously.

We used HastSet and ArrayList to store the crawled and to be crawled URL data.

Definition:

$List < String > urlLinks = new\ LinkedList < String > ();$
$Set < String > uniqueLinks = new\ HashSet < String > ();$

Method Representation:

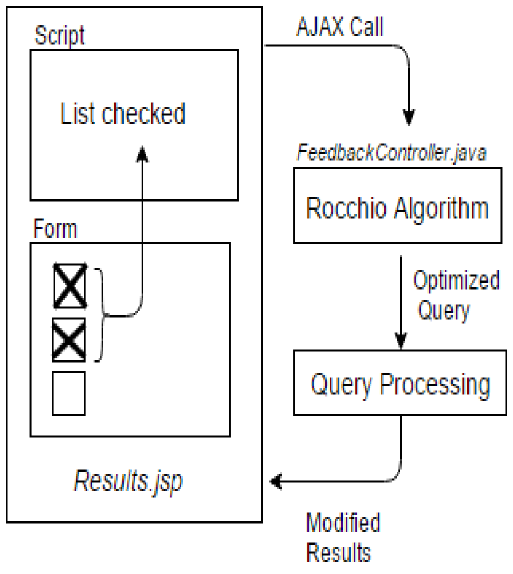
$Elements < Variable >=$
$document.select("a[href]");$

F. Search Engine User Interface

We developed an user interface for taking the user query and passing it to the Information Retrieval system.The User Interface further displays the retrieved ranked results based on the query input and relevance feedback. Front-end interface is developed using JavaScript.

G. Relevance Feedback

In order to improve our search engine performance we implemented Relevance Feedback concept.Now user will have a chance to select relevant documents which are in-turn used to refine query processing. An important consideration here is that relevancy selection is entirely dependent on the specific user. The results might change from user to user based on the relevancy selection. The flow diagram of Relevance Feedback is shown below.



We have added a check box for every retrieved document that is displayed to the user. After viewing all the retrieved documents, user can now select few documents as relevant and hit on submit feedback button. Internally,the selected documents are added to a list in the script part of .jsp file. From here an Ajax call is made to $'FeedBackController.Java'$. Here we implemented the Rocchio algorithm and calculated an optimized query vector. This is again passed to the method $'get\_similarities(doc\_vector, query\_vector)'$. Cosine similarities are again calculated for this optimized query and all the certified documents to get new ranks which are used to sort the documents to be displayed.In this way Relevance Feedback is designed and implemented.

H. Term Proximity

The main motto of the Term Proximity module is to retrieve the documents based on the highest proximities of the query terms. To start with the logic behind our algorithm, we tokenized the query string and chose the two important terms from the query string. The importance of the terms was assessed using their respective IDF values. Further, we retrieved all the documents which contains those terms and calculated the results based on the highest proximities of the query terms. For example, let us consider that there are two query terms 'kansas' and 'university' which are present in three documents D1, D2, D3 and say their corresponding positions in those documents are D1:[4,8],D2:[6,7],D3:[9,15].Then the term proximity is calculated as the inverse of their distances.So the term proximity values for the three documents are calculated as: (D1, 1/4) , (D2,1/1), (D3,1/6). Final weightages were assigned for cosine similarity and term proximity as 0.75 and 0.25 respectively and documents are ranked and retrieved accordingly. This is explained by the following example.

	Cosine similarity	Term Proximity
D1	0.1	0.25

Then the final weightage for D1:
W1 = (0.1 * 0.75) + (0.25 * 0.25).

Similarly the calculations are done for other documents and ranked in descending order of final weights. In this process the performance of our search engine is term proximity is improved by Term Proximity.

Data Structure:

The Data Structure for Term Proximity is as below:

$$HashMap < String, Double > \{ \\ String \;\; file\_name, \\ Double \;\; proximity \\ \}$$

IV. Observations

Stop word removal implementation:

Let's consider the sample input text-"HBO has released a new March Madness promo for Game of Thrones Season 6"
The output after removing the stop words would be "HBO released new March Madness promo Game Thrones season 6"

Porter Stemmer implementation:

The Porter stemmer for a sample input text "HBO released new March Madness promo Game Thrones season 6"
yields the following result "hbo releas new march mad promo game throne season 6"

Complexities:

Time Complexity for file parsing and fetching the term frequency is $O(n)$, for fetching the document frequency and the list of files in which the term is present is $O(n^2)$.

Web Crawling:

The time taken to crawl 554 URL links and write their contents in our local directory by two threaded web crawler is one minute ten seconds.

V. Challenges Encountered

1. Initially, we constructed the Inverted index without taking the Term-Proximity concepts in-to consideration. Later, when we realised that we had to include positions of the terms in the Index, we had to rewrite the data structure to incorporate the positions of the terms.

2. During the construction of Web Crawler, we had to take at most care while crawling web pages as we faced with numerous exceptions and server access concerns.Later we explicitly included our browser name and content type to avoid exceptions.For the crawler to be polite, we had to check multiple scenarios of Sleep Time and search engine output trade-offs into consideration.

3. In Relevance Feedback, while redirecting the User Feedback in the form of $Relevant\_Documents\_List$ to the $Feedback\_Controller\_Servlet$, we had a hard stop of passing the List to the Controller. Then, we figured out that AJAX Call will resolve the standing issue.

VI. Conclusion

To conclude, we developed a search engine application as part of our academic project on Information Retrieval. In this project the search engine retrieves only unstructured text. The modules which we developed to successfully display the results include, HTML parser, Inverted Index, Vector-Space Model and Niche Crawler. Further, to improve the performance of the application, we implemented the features, Relevance-Feedback and Term Proximity. There is always scope for improving the performance of any application. We tried to enhance the performance by minimizing the number of loops in a method and by following MVC Architecture. We hope not to leave this learning curve here with this project and continue to enhance the application and further apply the concepts in future projects. This subject was a great learning experience for the team in both academically and professional contexts

References

[1] Introduction to Information Retrieval,Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. . Cambridge University Press. 2008.

[2] Java^{TM} Platform Standard Ed. 7 documentation

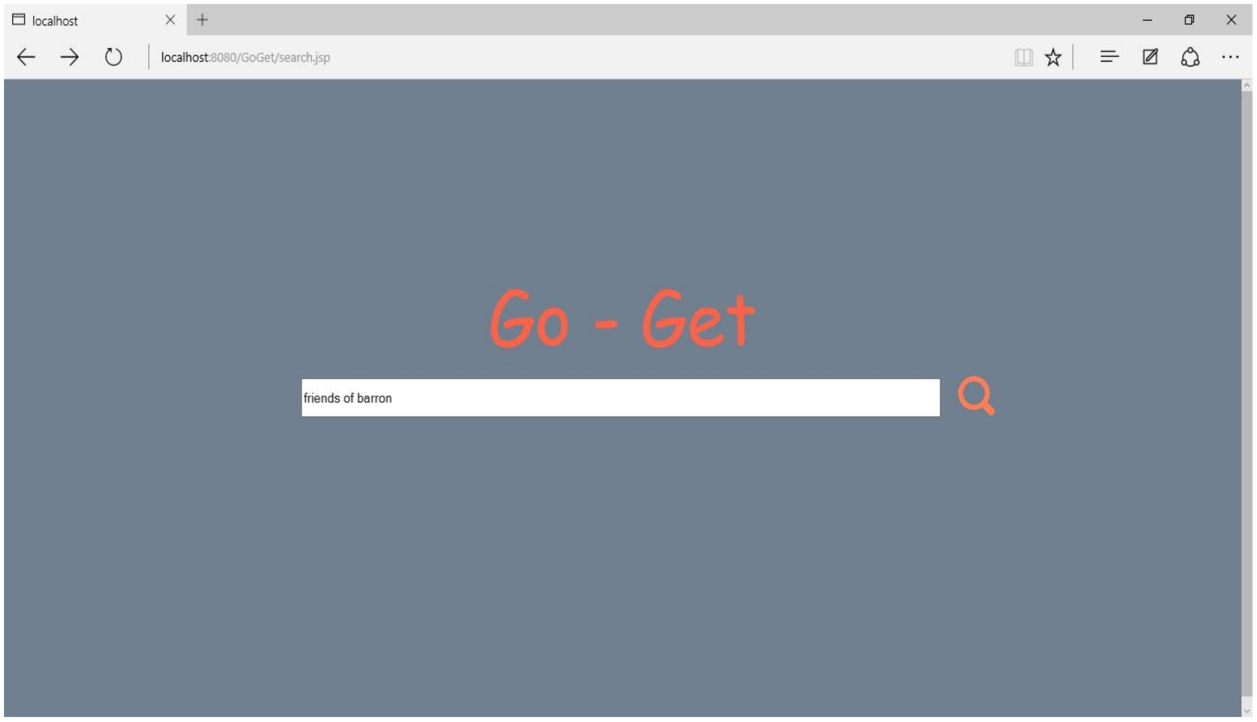
[3] Porter stemmer algorithm-Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,no. 3, pp 130-137

[4] Stack Overflow- user contributions licensed under cc by-sa 3.0 with attribution required rev 2016.5.13.3573 [*http://stackoverflow.com/*].

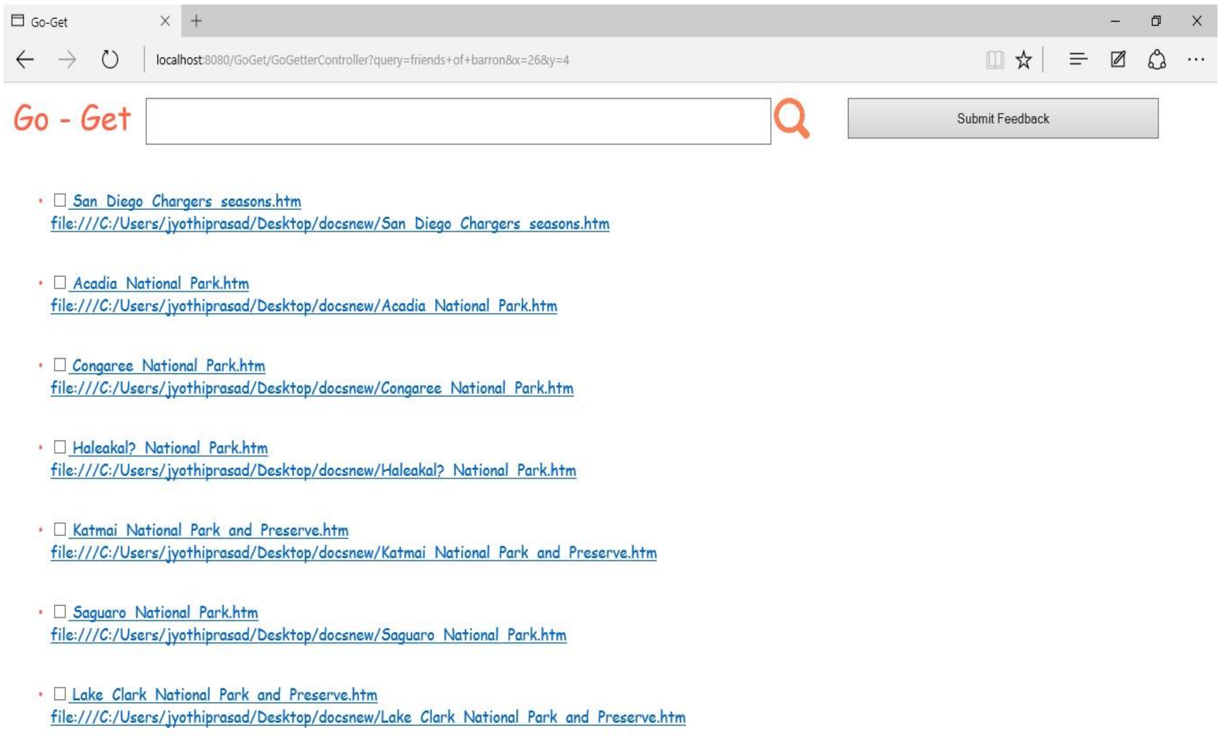
[5] Jsoup HTML parser,Copyright (c) 2009-2016 [*https://jsoup.org/*].

Screen-Shots

I. Go-Get Search Engine's User Interface:

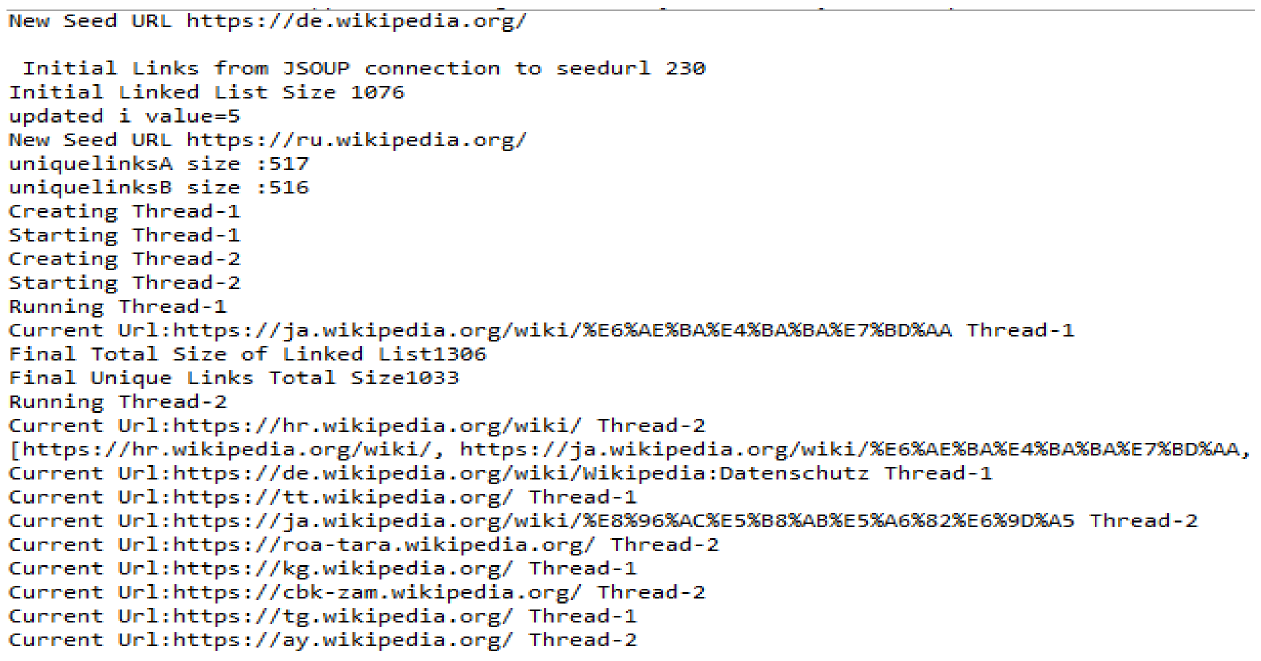


II. Retrieval of documents after query processing:


Go-Get
localhost:8080/GoGet/GoGetterController?query=friends+of+barron&x=26&y=4
Go - Get
Submit Feedback

- ☐ San_Diego_Chargers_seasons.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/San_Diego_Chargers_seasons.htm
- ☐ Acadia_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Acadia_National_Park.htm
- ☐ Congaree_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Congaree_National_Park.htm
- ☐ Haleakal?_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Haleakal?_National_Park.htm
- ☐ Katmai_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Katmai_National_Park_and_Preserve.htm
- ☐ Saguaro_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Saguaro_National_Park.htm
- ☐ Lake_Clark_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Lake_Clark_National_Park_and_Preserve.htm

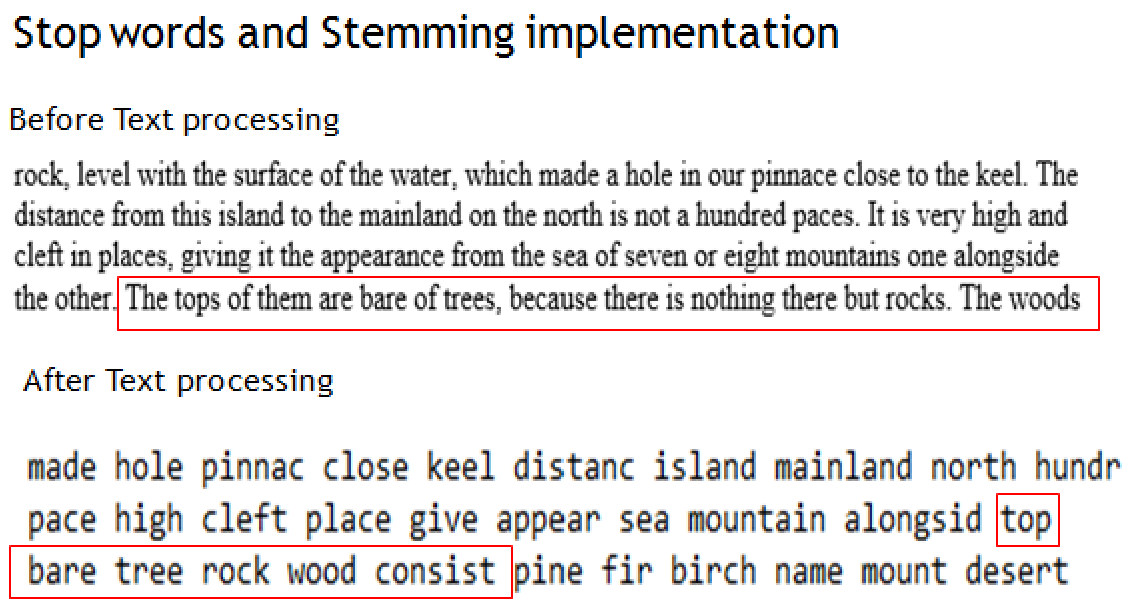
III. Multi-Threading in Web Crawling:



```
New Seed URL https://de.wikipedia.org/

 Initial Links from JSOUP connection to seedurl 230
Initial Linked List Size 1076
updated i value=5
New Seed URL https://ru.wikipedia.org/
uniquelinksA size :517
uniquelinksB size :516
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Current Url:https://ja.wikipedia.org/wiki/%E6%AE%BA%E4%BA%BA%E7%BD%AA Thread-1
Final Total Size of Linked List1306
Final Unique Links Total Size1033
Running Thread-2
Current Url:https://hr.wikipedia.org/wiki/ Thread-2
[https://hr.wikipedia.org/wiki/, https://ja.wikipedia.org/wiki/%E6%AE%BA%E4%BA%BA%E7%BD%AA,
Current Url:https://de.wikipedia.org/wiki/Wikipedia:Datenschutz Thread-1
Current Url:https://tt.wikipedia.org/ Thread-1
Current Url:https://ja.wikipedia.org/wiki/%E8%96%AC%E5%B8%AB%E5%A6%82%E6%9D%A5 Thread-2
Current Url:https://roa-tara.wikipedia.org/ Thread-2
Current Url:https://kg.wikipedia.org/ Thread-1
Current Url:https://cbk-zam.wikipedia.org/ Thread-2
Current Url:https://tg.wikipedia.org/ Thread-1
Current Url:https://ay.wikipedia.org/ Thread-2
```

IV. Text Pre-Processing:



Stop words and Stemming implementation

Before Text processing

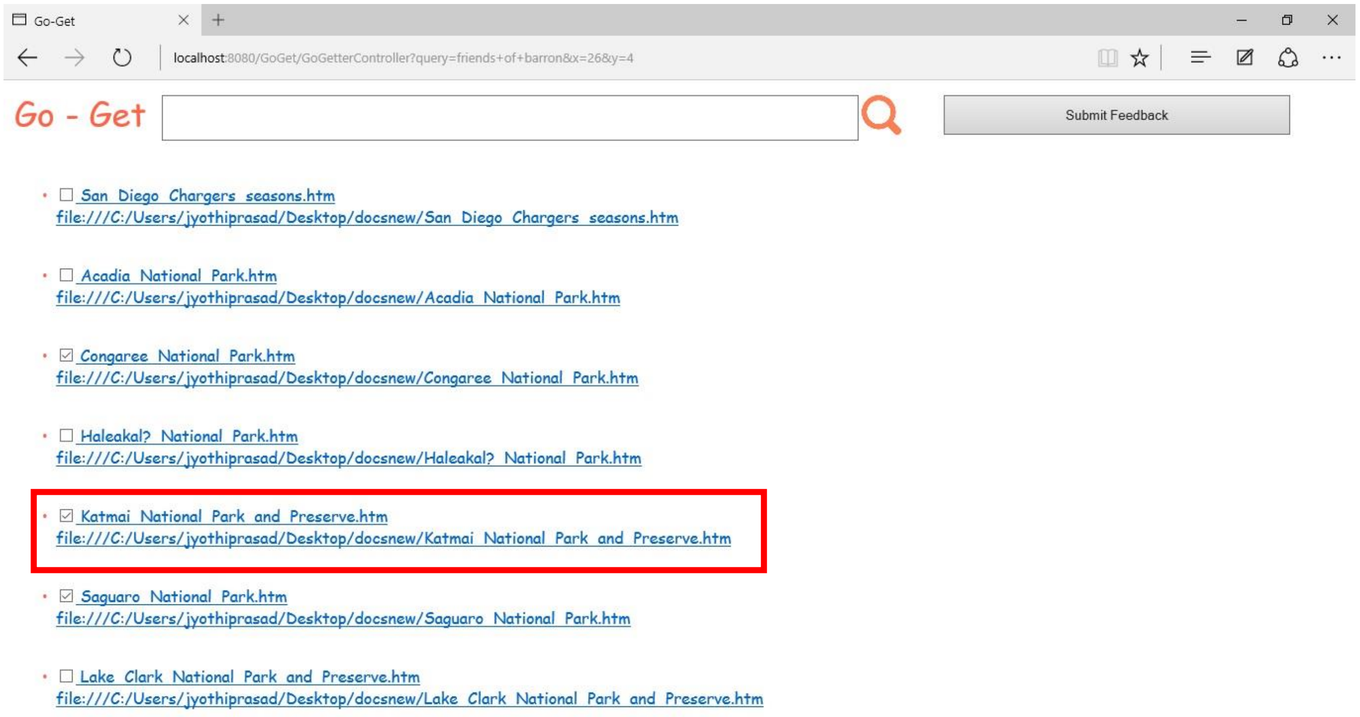
rock, level with the surface of the water, which made a hole in our pinnace close to the keel. The distance from this island to the mainland on the north is not a hundred paces. It is very high and cleft in places, giving it the appearance from the sea of seven or eight mountains one alongside the other. The tops of them are bare of trees, because there is nothing there but rocks. The woods

After Text processing

made hole pinnac close keel distanc island mainland north hundr pace high cleft place give appear sea mountain alongsid top bare tree rock wood consist pine fir birch name mount desert

V. Relevance Feedback:

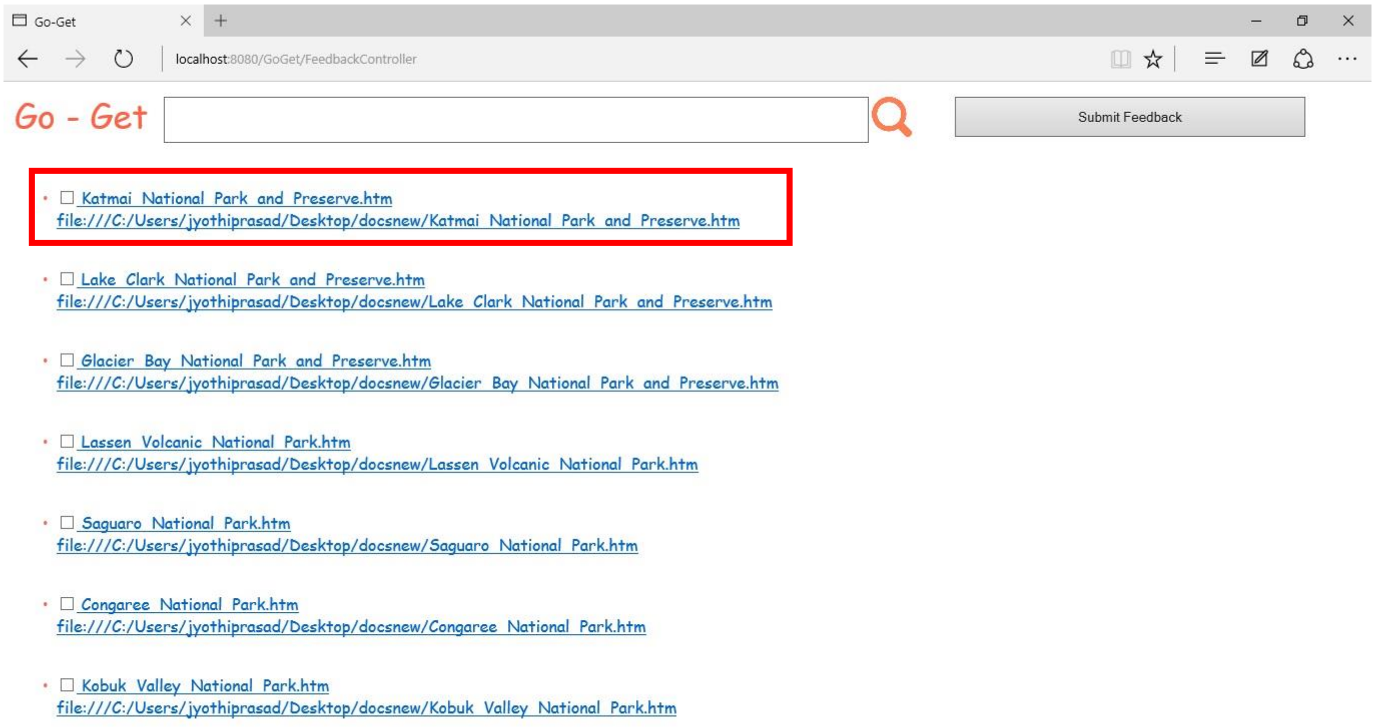
End-User checking the relevant documents:



localhost:8080/GoGet/GoGetterController?query=friends+of+barron&x=26&y=4

- ☐ San_Diego_Chargers_seasons.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/San_Diego_Chargers_seasons.htm
- ☐ Acadia_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Acadia_National_Park.htm
- ☑ Congaree_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Congaree_National_Park.htm
- ☐ Haleakal?_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Haleakal?_National_Park.htm
- ☑ Katmai_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Katmai_National_Park_and_Preserve.htm
- ☑ Saguaro_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Saguaro_National_Park.htm
- ☐ Lake_Clark_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Lake_Clark_National_Park_and_Preserve.htm

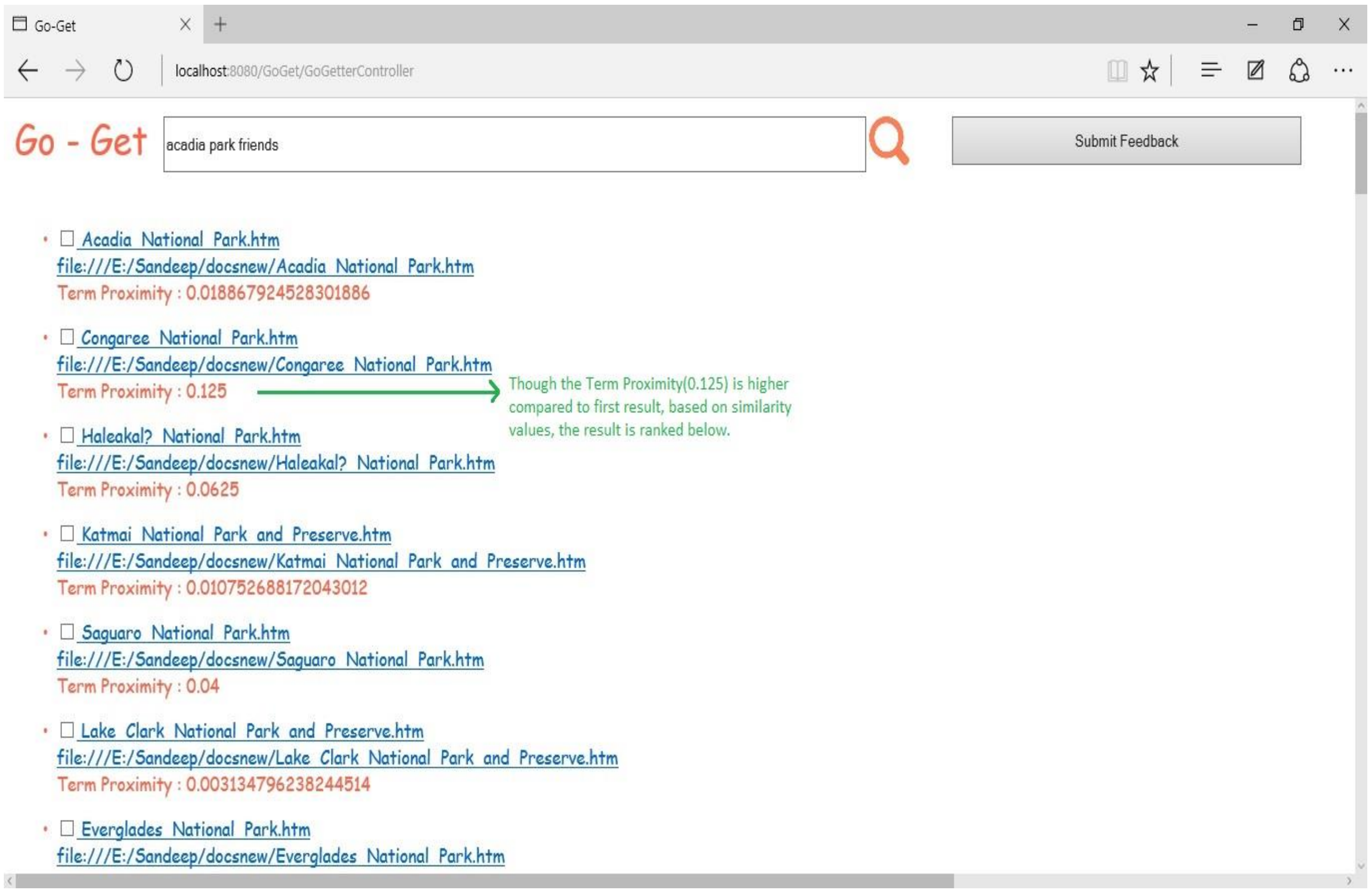
Re-Ranked documents based on relevance feedback:



localhost:8080/GoGet/FeedbackController

- ☐ Katmai_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Katmai_National_Park_and_Preserve.htm
- ☐ Lake_Clark_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Lake_Clark_National_Park_and_Preserve.htm
- ☐ Glacier_Bay_National_Park_and_Preserve.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Glacier_Bay_National_Park_and_Preserve.htm
- ☐ Lassen_Volcanic_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Lassen_Volcanic_National_Park.htm
- ☐ Saguaro_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Saguaro_National_Park.htm
- ☐ Congaree_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Congaree_National_Park.htm
- ☐ Kobuk_Valley_National_Park.htm
  file:///C:/Users/jyothiprasad/Desktop/docsnew/Kobuk_Valley_National_Park.htm

VI. Displaying Term Proximity values for retrieved documents



- ☐ Acadia_National_Park.htm
  file:///E:/Sandeep/docsnew/Acadia_National_Park.htm
  Term Proximity : 0.018867924528301886
- ☐ Congaree_National_Park.htm
  file:///E:/Sandeep/docsnew/Congaree_National_Park.htm
  Term Proximity : 0.125
- ☐ Haleakal?_National_Park.htm
  file:///E:/Sandeep/docsnew/Haleakal?_National_Park.htm
  Term Proximity : 0.0625
- ☐ Katmai_National_Park_and_Preserve.htm
  file:///E:/Sandeep/docsnew/Katmai_National_Park_and_Preserve.htm
  Term Proximity : 0.010752688172043012
- ☐ Saguaro_National_Park.htm
  file:///E:/Sandeep/docsnew/Saguaro_National_Park.htm
  Term Proximity : 0.04
- ☐ Lake_Clark_National_Park_and_Preserve.htm
  file:///E:/Sandeep/docsnew/Lake_Clark_National_Park_and_Preserve.htm
  Term Proximity : 0.003134796238244514
- ☐ Everglades_National_Park.htm
  file:///E:/Sandeep/docsnew/Everglades_National_Park.htm

Though the Term Proximity(0.125) is higher compared to first result, based on similarity values, the result is ranked below.