Boogle – The Book Finder

Anandhi Solaimuthu
Drexel University, Philadelphia, PA
as5326@drexel.edu

Astha Jain
Drexel University, Philadelphia, PA
aj887@drexel.edu

Anju Tyagi
Drexel University, Philadelphia, PA
at3358@drexel.edu

Harshita Guddad
Drexel University, Philadelphia, PA
hg449@drexel.edu

Manasa J
Drexel University, Philadelphia, PA
mj658@drexel.edu

ABSTRACT

The project pursued in this paper is to develop Books search engine. We have set up full-text customized search application, our example app will provide a UI to search the complete texts of 1000+ technical Books.

This article covers the implementation of custom search engine using Elastic search and custom similarity. We are taking around 1000+ technical books from google book's API [5] and indexing them into Elastic Search [4]. Later, we are re-indexing documents in elastic search using custom similarity like DFR and BM25.

CCS Concepts

Information retrieval systems →Search engine architectures and scalability;

Keywords

Custom similarity; Document retrieval

1. INTRODUCTION

Let us think how does the Google search the entire internet with whatever we type in the search bar.? And it gives the relevant result for the search even if we misspell it. Similarly, not only Google even the Wikipedia, how does the Wikipedia sort through 5+ million articles to find the most relevant one with our search keywords.?

To understand the basic background process of these questions, we have set up full-text customized search application (obviously not a high level as Google), our example app will provide a User Interface to search the complete texts of 1000+ technical Books. Along with this we can also preview the book and get more information about it.

Generally when we try to search the book the query will be searched with in a trillions of book entries in the internet, but our project helps in customized search engine in which the relevant books are found in a relatable field in our case it will be technical field.

From our home page, when we search for a particular book and hit the search button the results are displayed in the new page, in which we can find a list of relevant books according to search keyword along with this we can also find a preview link to view the e-book and info link gives us many details about the author, bibliographic information and some tags.

2. EXPERIMENTS

Boogle is organized into three separate components: Data Collection, Data Indexing and Data Presentation. They are described in detail below.

2.1 Data Collection

Based on the list of technical books provided and requests module of python, we are fetching around 1000+ books from Google book's API. Google API allows access to the metadata of a books such as title, description, subtitle, publisher, country, language, pagecount, categories etc. via an http request which returns a JSON file with keys. [5] The response is in JSON format so, after processing the response we take out Title, Description, Authors, preview link and info link of the books.

2.2 Data Indexing and Searching

2.2.1 Data Indexing

For book search, the extracted data for each Python package is considered a document and indexed. All fields in a document are indexed, but the following fields are first analyzed using the "standard" analyzer for authors, info link, preview link and id. The "standard" analyzer divides text into terms on word boundaries, as defined by the Unicode Text Segmentation algorithm. It removes most punctuation, lowercases terms, and supports removing stop words. The "English" analyzer is used on description, which aims at analyzing English language-specific texts.

2.2.2 Data Searching

We define our search query according to ES Query DSL to look for matches in the index for user queries. ES provides many techniques like a coordination factor, field length normalization and term or query clause boosting to search a keyword. We have used BM25 similarity, which is applied on title and author, but we have tweaked its parameter values. Default value of parameter k1 in BM25 is 1.2,

K1 Controls non-linear term frequency normalization (saturation). We have changed k1 value to 2.0. Also, parameter b controls to what degree document length normalizes tf values. The default value is 0.75 but we have changed it to 1.0 [2]

DFR is used on description. We have configured the DFR similarity so it can be referenced as custom_dfr, DFR Similarity implements the divergence from randomness framework. We have taken basic_model (Possible values: g, if, in and ine.) as "g", Geometric as limiting form of the Bose-Einstein model. The formula used in Lucene differs slightly from the one in the original paper: F is increased by 1 and N is increased by F. We have set after_effect (Possible values: b and l.) parameter as "l",Model of the information gain based on Laplace's law of succession.
Parameter normalization - Normalization model in which the term frequency is inversely related to the length. We have taken value as H2. While this model is parameter less in the original article, the thesis introduces the parameterized variant. The default value for the c parameter is 1.

Creates NormalizationH2 with the supplied parameter c.

Parameters:

c - hyper-parameter that controls the term frequency normalization with respect to the document length. [3]

2.3 Data Presentation

The browser interface features a home page with a simple text box for queries

Figure 2.3.1: Boogle - The Book Finder

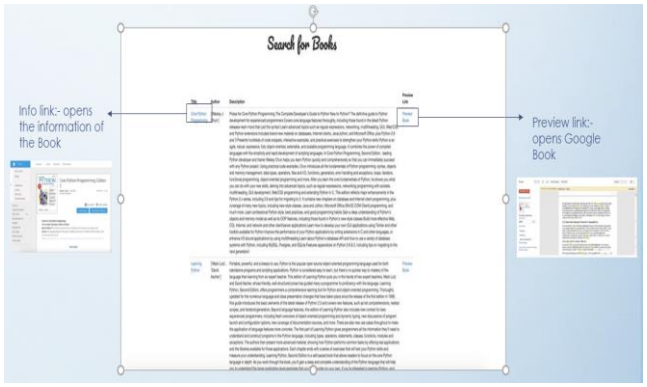

Figure 2.3.2: The Output page

In the search (Figure 2.3.2) the Elastic Search returns a list of books in the output page.

The info link redirects to Google book webpage where we can see many details about the book such as the cover page, author, reviews and ratings also an option to view the e-book. The page also displays the related books so the user can have a quick look at them, some details about the author, bibliographic information and some tags (common terms and phrases) added to book.

Preview Book link that displays the e-book with the page that opens up the e-book highlighting search query words.

3. IMPLEMENTATION

In this section we present the software/hardware requirements and the overall architecture to implement the proposed system.

Figure 3.1: System Implementation

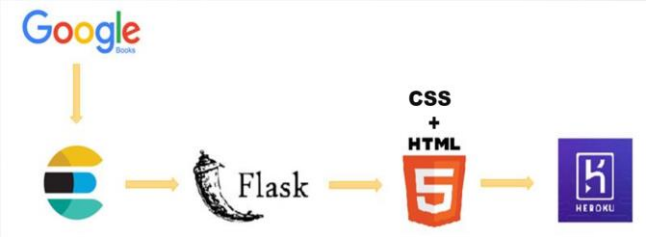

Figure 3.2: System Workflow

3.1 Software/Hardware Requirements

The web application is deployed and hosted on Heroku [6], a cloud platform as a service (PaaS) solution. Heroku allows a quick deployment, building and managing the application while bypassing infrastructure requirement to host a web application. Heroku supports applications written in programming languages such as Python. We used GitHub as the development platform and utilized the version control system. The codes uploaded in GitHub repository is linked with Heroku. Usually we need the server machine with internet connectivity so that the Boogle web app can be called on. Currently the application is hosted on Heroku's cloud server and can be accessible through internet. Added advantage is, our application works in all browsers and has mobile support.

3.2 System Architecture

Here, we present the overall architecture of the proposed web application for the book finder which uses custom similarity

components such as BM25 and DFR. In our System Implementation, (Figure 3.1) there are three main phases which include the backend (Flask), front-end (HTML5 and CSS), and deployment (Heroku). To create a web application, we utilized Flask, a micro web framework provides with tools, libraries and technologies that allows to build a web application. We styled the page with CSS to get some better UI experience. Finally, we deployed the models on Heroku. Figure 3.2 represents the outline of system workflow

4. EVALUATION

To evaluate the search engine, we consider precision and nDCG evaluation metrics. As we can not conclude the search engine performance only by considering single query precision value so, we are considering average of precision for few different queries as below,

Query1: "Python books"

'The information needed by this query is to find most relevant Python Books from our index'

Precision = 1 and nDCG = 1

Query2: "Machine learning by shai"

'The information needed by this query is to find most relevant Machine learning Books from author Shai '

Precision = 0.4 (low precision is because only 2 Machine learning books were written by this author in our index) and nDCG = 1

Query3: " By William"

' The information needed by this query is to find all the books written by author William'

Precision = 1 and nDCG = 1

for these sample of queries, we are calculating the Average precision:

Avg Precision = (1+0.4+1)/3 = 0.8

Similarly, we have evaluated several sample queries and we got good precision and nDCG values.

5. CONCLUSION AND FUTURE WORKS

Boogle is designed as a scalable search engine for technical books, powered by Elastic Search. The main goal is to replace traditional search engines so that this application can return high quality results for technical users. It is a complete system that acquires, analyses and indexes a large number of books enabling fast and semantic searches.

Currently, we are just taking technical books. As a future work we can add more non-technical Books. Also, we are presently taking books from Google API only, later we can hit other books related API to gather more relevant information about any book.

6. ACKNOWLEDGMENTS

This work was supported in part by the Drexel University College of Computer and Informatics under the course INFO- 624-003 for academic project.

7. REFERENCES

[1] https://lucene.apache.org/core/8_6_0/core/org/apache/lucene/search/similarities/DFRSimilarity.html

[2] https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html

[3] https://cloud.elastic.co/registration?elektra=downloads-overview&storm=elasticsearch%20

[4] https://www.elastic.co/what-is/elasticsearch

[5] https://developers.google.com/books

[6] https://dashboard.heroku.com/