Search Engine for Structured Data

Ksenia Saenko
MS Data Science
ks4841@nyu.edu

Stella Sun
MS Data Science
ss8955@nyu.edu

Zhiwei Yu
MS Computer Science
zy1129@nyu.edu

ABSTRACT

The search engines are essential tools locate online information for many people. For this paper, we are using NYC Open Data (https://opendata.cityofnewyork.us/) as an example to show how we implement inverted index, similarity score, etc. to make searching more efficient.

The method we used were Hadoop Mapreduce and Spark, and we wanted to enable not only table name searches, but also column search and content search, with customized filters.

CCS CONCEPTS

• **Database management system engines** • **Database query processing**

KEYWORDS

Search Engine, Data Cleaning, Database Design

ACM Reference format:

G. Gubbiotti, P. Malagò, S. Fin, S. Tacchi, L. Giovannini, D. Bisero, M. Madami, and G. Carlotti. 1997. SIG Proceedings Paper in word Format. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK'97)*, 4 pages. https://doi.org/10.1145/123 4

1 INTRODUCTION

Problem: efficiency and quality of any data project depends on the ability to accurately and timely obtain the necessary information. The current search capabilities of Open Data NYC portal are limited to search over table titles (See Fig. 1). A user cannot search contents of tables, such as specifying a column name or keyword to retrieve from the rows. A user also cannot search by table properties, such as a number of rows in a table. The current interface is inconvenient for the user and results in slower workflow and inability to find all relevant tables.

The goal of this project is to build a search engine for structured data with improved query capabilities for the NYC Open Data portal https://opendata.cityofnewyork.us/
Our search engine will allow users to query the contents of a large collection of data sets using structured queries.
Users will be able to specify their search criteria. For example, they could ask for all data sets that contain a column called "taxi", or a value "1988" from a table with at least 1000 rows, or a column "address" that contains the string "village". To accomplish that, we will provide capabilities to search by 3 levels: table title, columns, and keyword search within the columns. In addition, we will provide filters on the final output to take table size and other properties of table instances into consideration for our search engine.

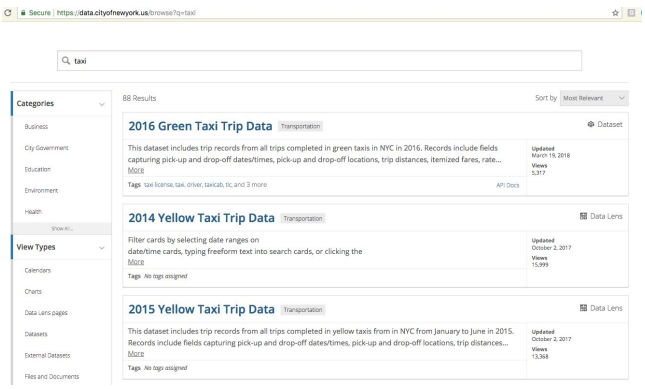

Figure 1: Search result in NYC Open Data. The platform only searches for keywords across titles.

2 ARCHITECTURE AND DESIGN

Figure 2 below described the search engine architecture. User input is translated into querying title, column, and content indices. The result is matched to document id in master index. The resulting table(s) are obtained from the HDFS directory. The output is ranked and filtered based on

user specifications and returned to the user in the form of document names.

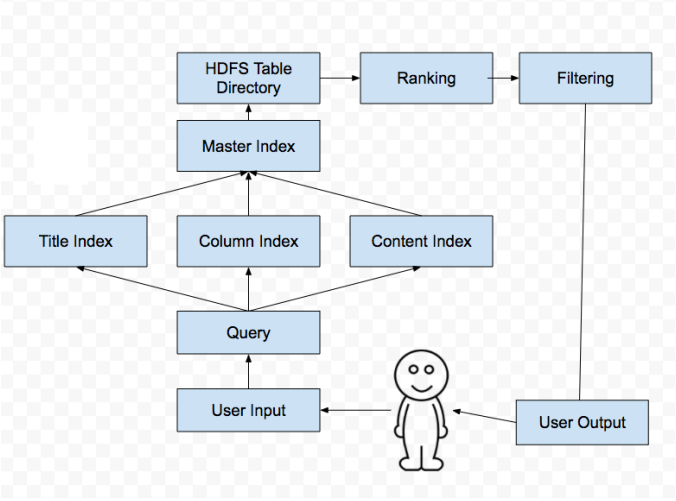

Figure 2: Architecture of the search engine.

3 METHODS

3.1 Sample Fabrication

To simulate the problem for concept testing and development we obtained a sample dataset from NYC Open Data website using Socrata API. We downloaded tables from five categories: business, education, government, environment, and health in order to have a representative sample. We also queried metadata about each table, including the number of columns and title for later processing. We used Python to create a table with table ID's matched to metadata to allow for efficient querying and filtering in later stages of the project. We uploaded fabricated data to Hadoop

3.2 Inverted Index

An inverted index is a data structure common to nearly all information retrieval systems. In our experiment, we created inverted index for three types of search: **title search** where user can search by table names; **column search** where user could search by individual columns from all tables in the system; **content search** that the search goes through all table contents.

3.2.1 Title Index.

Our first goal was to build an inverted index for title search. After initial research we decided on the following design of the index table: key will be the keyword, first column represents a number of documents containing the keyword, third column contains key-value pairs of the form (unique document ID, number of times the keyword appears in the document). This format is efficient for later search as it allows to sort the index by the keyword frequency. Hence searching for a word in the index is faster for more frequent words.

We implemented two version of creating the title index - one with Spark and one with Hadoop. This way we could determine the most efficient method.

3.2.2 Column Index.

We then built an inverted index for the column names of each table: the keyword is the specific word we will be looking for in the column names sorted by the frequency in the dataset and the corresponding index is the document ID along with its column name. This process is implemented by MapReduce, made parallel and thus efficient as the preprocessing of the user query search.

3.2.3 Content Search.

To create inverted index for content search, we used MapReduce to go through all tables and took each word as a key, with a (table_name, count) as value. The key, value pair was reduced as (key, frequency among tables, table_name) with table_name sorted by in-table frequency. The result was then sorted and printed by frequency in the descending order.

3.3 Query Design

Python functions could handle the query design by taking inputs from user, including the keyword, customized filter, etc. and returning a SQL query. Then the query would go directly to the index tables, and then points to the raw tables.

3.4 Similarity Score, Ranking

We will rank the results by relevance based on several metrics such as how many keywords in the user input are found in each result and the similarity between them. The similarity should be based on the edit distance of strings.

4. TESTING AND VALIDATION

We will test accuracy of results compared to searching in NYC Open Data website. We will also time our queries to ensure efficient performance. We will compare our Hadoop vs. Spark methods to determine which one results in the best performance in terms of time.

5 RESULTS AND DISCUSSION

5.1 Preliminary Results

We obtained sample of datasets from NYC Open Data. We created three indices: title index, column index, and content index. We have two methods for creating inverted index: using Hadoop and Spark.We will later compare performance of these approaches. We also obtained a master table with metadata for tables. Figure 3 below describes a general approach for creating inverted index using key-value pairs for MapReduce concept.

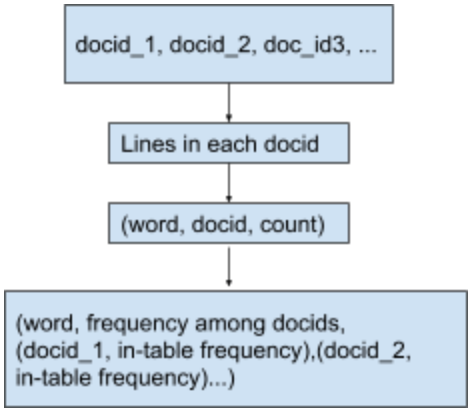

Figure 3: Work progress to make inverted index

```
('2011', 3, (('1', 1), ('2', 1), ('4', 1)))
('city', 3, (('4', 1), ('5', 1), ('8', 1)))
('york', 2, (('5', 1), ('8', 1)))
('new', 2, (('5', 1), ('8', 1)))
('results', 2, (('1', 1), ('5', 1)))
('school', 2, (('1', 1), ('2', 1)))
('by', 2, (('1', 1), ('8', 1)))
('level', 2, (('1', 1), ('2', 1)))
('and', 2, (('4', 1), ('8', 1)))
('center', 1, (('3', 1),))
('projects', 1, (('6', 1),))
('bins', 1, (('9', 1),))
('group', 1, (('8', 1),))
('test', 1, (('1', 1),))
('class', 1, (('2', 1),))
('dohmh', 1, (('5', 1),))
('of', 1, (('4', 1),))
('size', 1, (('2', 1),))
('domain', 1, (('0', 1),))
('inquiry', 1, (('3', 1),))
('call', 1, (('3', 1),))
('math', 1, (('1', 1),))
('grade', 1, (('1', 1),))
('law', 1, (('4', 1),))
('names', 1, (('8', 1),))
('recycling', 1, (('9', 1),))
('48', 1, (('4', 1),))
('detail', 1, (('2', 1),))
('2010', 1, (('2', 1),))
("mother's", 1, (('8', 1),))
```

Figure 4: Table title index example

5.2 Github Repository

https://github.com/stella0316/Search_Engine

6 CONCLUSIONS

In summary, we have developed methods in Spark and Hadoop to create inverted indices efficient for querying and ranking. These results will allow to innovate upon the current capabilities of NYC Open Data website - in particular, they will allow a user to search for keywords in columns and contents, which is currently unavailable at the website.

Our next steps will be creating queries and building ranking using similarity score. We will also explore the latest tech tools to implement, such as Elasticsearch.

A.1 Introduction

A.2 Architecture and Design

A.3 Methods

A.3.1 Sample Fabrication

A.3.2 Inverted Index

A.3.3 Query Design

A.3.4 Similarity Score, Ranking

A.4 Testing and Validation

A.5 Results and Discussion

A.5.1 Preliminary Results

A.5.2 Github Repository

A.6 Conclusion

A.7 References

REFERENCES

[1] Cafarella, M. J. et al. WebTables:Exploring the Power of Tables on the Web.

[2] Dasu, T. et al. Mining Database Structure; Or, How to Build a Data Quality Browser.

[3] Agrawal, S. et al. DBXplorer: A System for Keyword-Based Search over Relational Database.

[4] Luo, Y. (n.d.). Spark:A Keyword Search Engine on Relational Databases.

[5] Rejaraman, A., & Ullman, J. (n.d.). *Mining of Massive Datasets*.

[6] Cloud9. (n.d.). Retrieved April 13, 2018, from https://lintool.github.io/Cloud9/docs/exercises/indexing.html