Search engine for efficient data discovery of NYC Open Data

Chaitra Hegde
New York University
USA

Zahra Kadkhodaie
New York University
USA

Aakash Kaku
New York University
USA

ABSTRACT

An efficient search engine is built to facilitate data discovery of NYC Open Data set repository based on a key word. Each dataset is summarize in a manner that leads to results which are relevant and also leads to an efficient search. To perform an efficient and relevant search, each dataset is characterize by set of features that helps us to find relevant datasets pertaining to a particular keyword. The search engine also implements a similar word search to increase the scope of user's search. For finding similar words, state-of-art word2vec model is used to find the nearest words. The search engine also enables the user to perform more specific and complex search based on user defined criteria.

KEYWORDS

Search Engine, Data Browser, Data discovery, Data Summaries, Data Profiling

ACM Reference format:
Chaitra Hegde, Zahra Kadkhodaie, and Aakash Kaku. 2016. Search engine for efficient data discovery of NYC Open Data. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the advent of massive datasets available on open data portal like NYC Open Data, Data.Gov, efficient search of repository in real time plays a very crucial role. In this project, we built an efficient and highly flexible search engine for NYC Open data which contains almost 1500 datasets. The search engine is capable of searching for datasets based on meta data keywords, table content keywords, similar words as well as complex combinations conditions pertaining to certain user specified need. Our search engine outperforms the existing NYC open data API in terms of efficiency as well as query flexibility.

2 PROBLEM FORMULATION

Currently, the NYC open data API can be used only for a single type of search, that is searching for one keyword. The goal of this project was to overcome this limitation by introducing a variety of search templates which could be employed by the user to obtain a narrow range of desired datasets in a short period of time. For example, a user might be interested in datasets which contain the keyword "1988" in at least 1000 of their rows. Or, she might be interested in all the tables which contain either "school" or "college" or some similar words to these keywords. Another user might be looking for all the datasets where "taxi" is a field name. The purpose of this project was to build an interactive search engine to help the users conduct a number of specific and flexible searches and obtain relevant datasets from a corpus of dataset.

A practical formulation of this problem requires implementation of an exhaustive, yet limited, number of search templates. To this end, we came up with 10 search templates each of which could be used for a common type of search. These 10 templates are as follows

(1) Search keywords in a) only metadata, b) in meta data and data
   - Example: Datasets which contain "nutrition" in only the meta data
   - Example: Datasets which contain "nutrition" either the meta data or the content data
(2) Search for a keyword with threshold on number of times it occurs
   - Example: Datasets which contain "address" at least 2000 times in the entire content of the table
(3) Search for a keyword with threshold on number of rows in the dataset
   - Example: Datasets which contain "1988" in at least 1000 of the rows.
(4) Search for datasets belonging to certain category
   - Example: Datasets which contain "district" and belong to the Education category
(5) Search for datasets with certain number of rows and columns
   - Example: All the datasets with more than 100 columns and 1000 rows.
(6) Search for multiple keywords in OR fashion
   - Example: Datasets which contain either "college" or "university" or **similar words** to these keywords
(7) Search for multiple keywords in AND fashion
   - Example: Datasets which contain both "parent" and "teacher"
(8) Search for files with certain columns in it
   - Example: Datasets where "state" is a column name
(9) Search for files with a certain column and value
   - Example: Datasets with a column whose name contains "address", and keyword "building" appears in that column
(10) See the Schema of Dataset

As noted under query (6), an effective algorithm should search for similar keywords in addition to the target keyword. This helps to widen user's search and increases the chances of user finding a relevant dataset. The similar words are determined using a state-of-the-art word2vec embeddings pre-trained on a large corpus [5].

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

3 PREVIOUS WORKS AND REFERENCES

There is extensive work done on building dataset browsers that performs simple search tasks like finding a set of datasets with a certain keyword and also, advanced search tasks like finding a set of datasets that has a field with very high resemblance to the a certain user input field. One of the example of such browser is *Bellman* browser [1]. These browsers also make use of data summaries that helps in doing quick and crude exploratory analysis of the data. Additionally, many approximate matching algorithms are developed that helps to performs joins based on string distances[2–4]. We find such approximate matching algorithms based on string distances useful for us to find relevant datasets from the corpus. Additionally, data summaries suggested in [1], like the number of tuples in each table,the number of distinct and the number of null values of each field.etc can also be useful to perform an initial quick search to give a list of relevant datasets that satisfies user specified conditions.

4 METHODS, ARCHITECTURE AND DESIGN

As mentioned in [1], the data discovery task becomes extremely easy if a data summary or data profile is built and cached in the memory. Since the data profile has a much smaller memory footprint than the data itself, it is very efficient to query and cache the data profiles. We have approached the problem of building a search engine for datasets on the similar lines.

But one has to ensure that data profile, that is built, captures essential information of the dataset. The data profiling helps to reduce data to query but it also results in losing information about the dataset. The loss of information will result into incorrect results. For example, there are many datasets in the NYC Open Data that contains the word 'Uber' but when performing the actual search no dataset is returned. This is because, the search engine of NYC open data has pruned a lot of essential data in order to make the search efficient. Therefore, there is an inherent trade off between speed and accuracy.

The trade off between speed and accuracy can be well answered if the features that summarizes the data are very informative and well suited to the task. For this task, simple features like number of distinct values and its counts, number of rows and number of null values, etc. can work quite well [1]. But we wanted some additional features related to the dataset which were more informative and suggestive about the dataset. These features are described in detail in the Method section.

4.1 Method

The method was as follows:

(1) Meta-data discovery
(2) Extraction of meta-data as data summaries
(3) Performing search over the data-summaries

4.1.1 Meta-data discovery. The meta-data discovery stage is the most important stage of the dataset searching process. During this stage, relevant features that can informatively characterize a dataset are chosen or built. Hence, more richer are the features, more relevant is the search result.

For the NYC Open data, each dataset is accompanied with a meta-data section. The meta-data section has a wealth of information about the dataset. This acted a good starting point to find informative features about a dataset.

From the meta-data section following features were extracted for each dataset:

(1) Attribution Link: It gives link to access the dataset
(2) Category: The category of the dataset e.g "Education", "Crime", etc.
(3) Description: The description briefly describes the dataset and the values in it.
(4) id: Unique id to identify the dataset
(5) name: Name of the dataset
(6) tags: Tags associated with each dataset e.g. For a public safety dataset, the tags were 'public', 'safety', 'crime', '2015-16', etc.
(7) Download count and View count: Gives information about the popularity of the dataset
(8) Columns: It describes each column of the dataset with attributes like column name, top words in the column, maximum value in the column, minimum value in the column, average value of the column, etc.
(9) num_rows: Number of rows in the dataset.
(10) num_cols: Number of columns in the dataset.

As it can be seen, the features like category, tags and description are very informative since it describes the dataset in a similar way as a human would describe it. Hence, it greatly eases the task of finding relevant datasets.

We also wanted to characterize each column of the each dataset in the repository. We used following features to characterize the columns of dataset:

(1) col_name: It gives the name of the column
(2) file_name: It gives the dataset name
(3) maximum: It gives the maximum value in a particular column
(4) minimum: It gives the minimum value in a particular column
(5) nulls: It gives the number of Nulls in a particular column
(6) type: It gives the type of data contained in the column
(7) position: It gives the position of the column in the given dataset (eg. If the column is second in the table, the value would be 2)
(8) Description: It gives the description of the column

As it can be seen, the features mentioned above are very rich in terms of capturing information regarding the columns present in the dataset.

A third summary table contained all the information regarding the words present in the column of a dataset and its count. This summary is called 'Word Count Summary' and it has following features:

(1) file_name: It gives the name of the dataset
(2) col_name: It gives the name of the column
(3) word: The word
(4) count: The count for the word

For creating the above table, each column of each dataset was processed as follows:

(1) All the text columns in the entire repository were identified. Some identifiers like '%','numb','number','#','ave',etc. were used to filter out the columns with numerical entries. We performed the search using the column names as the column names gave a good enough information on the type of entries in the column. We only performed word-count processing of the columns with text entries like 'Address', 'Name of Building', etc. We were left with 6999 text columns on which word-count processing was done
(2) The entry in each text column was cleaned and normalized. For cleaning, all the punctuations, all the null entries were filtered out and white spaces were removed. For normalizing, all the text was converted to lower case, longer sequence of text was broken into individual words and all the common stop words were filtered out..
(3) After tokenizing the long text sequence in to individual words, the word count was performed and all the words with length greater than three characters were kept. We performed an analysis of number of words to retain, and it was seen that top 200 unique words seemed to capture all the required information without loosing essential information. Below are few charts that showcases that as we reach 200th word, the count tends towards 1 which signifies that top 200 words are sufficient to capture the essential information of a column.

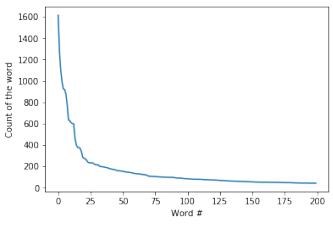


Figure 1: Distribution of count v/s word for one of the sample text column

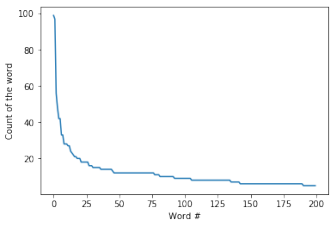


Figure 2: Distribution of count v/s word for one of the sample text column

(4) Steps 2 and 3 are repeated for each of 6999 text columns.

We, therefore build three data summaries that are linked either by dataset name or column name. The schema for three summaries and fields connecting them can be seen below.

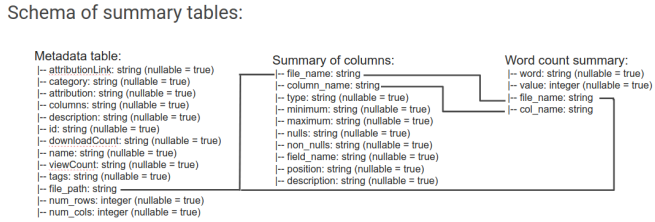


Figure 3: Schema of Summary Tables

4.1.2 Extraction of meta-data as data summaries. The extraction of meta-data is relatively straight forward if the meta-data to extract are well defined. In our case, all the meta-data are well defined and majority of them are readily available in the meta-data section of the dataset. But some meta-data like distinct value and its counts, number of rows, number of null values needs to be extracted from the dataset. A naive implementation of SQL queries to extract these values would be imprudent and non-beneficial. Since, the naive implementation of distinct values and its count can result in words with punctuation, inconsistent case type (some in lower case and some in upper case) and extra white spaces. Additionally, it would be inefficient to naively implement an SQL query to extract these values for a large corpus. We, therefore, cleaned the values of each column (field) of the dataset by removing extra white spaces, removing punctuations, standardizing all entries to lower case, filtering out empty rows and also filtering out common stop words. Once this cleaning is done, a word count can be performed that returns distinct values and its count for each column (field) in the dataset.

Once all the meta-data is extracted, it will be cached as data summaries for future use. The data summary of each dataset are joined to obtain a collection of data summaries. This will help us to avoid the future computation of the values of the data summaries. One important point to note here is that this method of using data summaries to search for a dataset would be efficient only if the velocity of the data is relatively moderate. If the velocity of data is very high like Twitter's tweet data, the data summaries would quickly be outdated and would no longer be reliable to fetch relevant results. But if the data doesn't have such high velocity, the data summaries can be updated in a timely manner to ensure they are relevant for the task of fetching relevant datasets.

4.1.3 Performing search over the data-summaries. If first two steps are performed well, then the last step is straight-forward. We just need an SQL query to fetch relevant results from the data summaries. An elegant part of this solution is that we can tailor the SQL query to perform very complex searches like finding a set of datasets that has value "1988" in a table with at least 1000 rows very efficiently and easily.

4.2 Design and Architecture

As a part of Design and Architecture, we determine different types of queries that can be performed by the user to find a dataset. Some search would be simple like finding a dataset with a particular keyword whereas some search can be complex like finding a set of dataset with columns name containing 'taxi' and with at least 1000

rows of data. Therefore, we came up a set of predetermined queries (query templates) that can interactively be used by the user via a command line interface. The set of query templates are as follows:

(1) Query to find a set of dataset with a particular keyword
(2) Query to find a set of dataset with a particular keyword with a threshold on number of times it occurs in the dataset
(3) Query to find a set of dataset with a particular keyword with a threshold on number of rows in the dataset
(4) Query to find a set of dataset belonging to a certain category like 'education', 'crime','public', 'transportation' etc.
(5) Query to find a set of dataset having certain number of rows and columns.
(6) Query to find a set of dataset having for multiple keywords in OR fashion
(7) Query to find a set of dataset having for multiple keywords in AND fashion
(8) Query to find a set of dataset with a certain key word in the column name. eg: Find all the datasets that has 'address' as one of its column.
(9) Query to find a set of dataset with a certain key word (kw1) in the column name and kw2 in that column. eg. Find all the datasets that contain 'address' as one of the column and 'building' word under the 'address' column.
(10) Query to print the schema of a particular dataset.

As it can be seen, there can be many such queries that can be performed by the user. But to showcase how useful data summaries are to fetch datasets, we have decided the above-mentioned queries as query templates for this project.

The users can nevertheless implement their own queries on the data summaries to find relevant information they need.

Result for some of the queries and results to same queries by NYC Open Data search engine are displayed and discussed in the Result section.

We have also implemented user interactive command line interface which makes the searching for keyword much easier. A snapshot it can be seen here 4. Famous datasets are very frequently viewed and downloaded by people and hence, when the command line interface runs, 5 most downloaded and viewed datasets from the repository are displayed to the user. If they are looking for something else, the user can choose one of the query templates displayed to them. When the choice is made, user is asked for inputs required to execute the query and result of the query is returned to the user.

5 TECHNICAL DEPTH AND INNOVATION

5.1 Technical depth

The extraction of features to describe a dataset requires considerable technical expertise to implement algorithms in a distributed file system environment with distributed computing. Since the size of the data is 650 GBs, without distributed computing the task would take ever to complete.

The data and meta-data are present in the .json files available on HDFS. This data is very huge and creating summaries would require us to iterate through the entire data. So, that's where Pyspark comes into the picture. Rather than using a mundane SQL implementation, using spark implementation of pyspark gives advantages of distributed computing. As mentioned before, we need to create summaries like meta-data, detailed description of columns, top 200 words and corresponding count, and row column count for each of the datasets in the repository. In order to fetch these data, we iterated through the datasets and extracted the required piece of information using pyspark implementation of pyspark. The entire phase of extracting summary takes about one hour when jobs are run in parallel.

Speaking about running jobs in parallel, the task like extracting summary for each of the dataset is independent of each other. Hence, executing the code sequentially is expensive. Instead, using the concept of threading in pyspark environment, we can create separate threads for every dataset or countable few datasets and achieve higher speeds to extract the summary.

But a question may arise, what if the data repository get modified or content of dataset changes? Since, the data summary extraction takes about one hour, one can easily update the summaries as required without having a high down time. The frequency of updating summaries depends on how often the data is modified.

Finally, the extracted data summaries are stored as CSV files and made available in HDFS for further usage.

5.2 Innovation

We have implemented an innovative search functionality that helps to increase the relevancy of search for the user. The task we are aiming to optimize is keyword search in different fashion by creating search templates. As mentioned before, we have come up with 10 query templates which accept keywords as input. But one way to increase the relevancy of the search could be to search for words similar in meaning to the key word inputted by the user. We, therefore, have implemented a similar word search functionality which helps users to search for words that are similar to the input word. The similar words are determined using a Glove word2vec embeddings which was trained on a large wikipedia corpus of 6 billion tokens with a vocabulary of 400k words [5]. Each word is represented using a vector of 50 dimension. For a given input word 'w', we find the top 3 nearest neighbors to word 'w' in this 50 dimensional sub-space using the cosine similarity measure. So, rather than searching for one word throughout the summary, now we look for 3 words. Also, number of similar words to map for it self could be used as an input from the user giving them more flexibility.

This would help to increase the scope of the user. Many a times users are also not aware of the exact word hence, this would recommend them some similar words from which the user can search. This is variation that we have implemented in the current search queries.

6 CODE DESCRIPTION

The entire code and cached files can found in the git repository: https://github.com/Zahra-Kadkhodaie/Data-InQuirer

Out of 2000 datasets that are contained in the NYC data repository, almost 500 of them do not contain anything and hence names of only non-empty datasets are extracted and saved in a text file - `file_name_2.txt`.

6.1 File names that extracts meta-data (3 data summaries)

`extract_meta.py` code iterates through these datasets and extracts required meta-data from them to form one file - `meta_data_full.csv` with meta-data of all the datasets. As mentioned in section 4.1.1, only attribution link, category, Description, tags, ID, Download and View count and Columns informations are extracted as they seem to contain most relevant information about the dataset.

Similarly, `column_meta_extract.py` is the code to extract the detailed description of each column for each dataset which is,eventually, cached as `table_col_meta.csv` in the `Cached_Data` folder.

Similarly, `extract_word_count.py` is the code to extract top 200 words and its count for each text column in the repository which is,eventually, cached as `final_wc_file.csv` in the `Cached_Data` folder.

6.2 File names that performs SQL query search on the cached data

(1) `main.py` is the file rendering the command line interface which asks for user's input.
(2) `category_search.py` code takes a keyword for category and searches over metadata for dataset belonging to that category.
(3) `search_keyword.py` is the code that accepts a keyword and performs search on `meta_data_full.csv` file.
(4) `col_val_type.py` is script to perform search for column name and column value.
(5) `column_name.py` is script to return datasets with particular column name.
(6) `min_max_row_col.py` is script that returns datasets pertaining to minimum and maximum constraints on number of rows and columns in datasets.
(7) `most_popular.py` returns five most viewed and downloaded datasets in the repository which are cached as `query_most_download.out` and `query_most_viewed.out` in the `Cached_Data` folder.
(8) `search_everywhere.py` contains the python code that performs a search for a given input word in all the three data summaries. It also fetches the similar words and performs search for the similar words in all the three data summaries.

7 RESULTS AND DISCUSSION

The Appendix section contains some snapshots of different queries. Figure 5 shows the search result for "uber", which contains a number of datasets, as opposed to the NYC open data API search result which does not return any datasets. Figure 6 shows the results for "income" as a field name, and Figure 7 shows datasets with a column that contains a keyword "address" with keyword "building" in it. Figure 8 shows datasets that contains word "classroom" at least 50 times. Figure 9 shows datasets that contains "alcohol" in it. Since we used the query template (6) for this search, the algorithm returns datasets which contain either "alcohol" or its similar words, i.e. "drinking" and "marijuana".

As demonstrated by the figures in the appendix, our search engine not only outperforms the NYC open data API in the single keyword search, but also it creates a flexible environment for the user to make specialized queries and obtain the most relevant datasets under certain conditions. More importantly, by making use of metadata and summary tables, the search is performed in real-time without compromising accuracy of the search to a large extent. Given that the summary tables contain most of the critical information of the datasets, more search templates can be developed later if new queries with other conditions are needed.

8 FUTURE WORK

To enhance the capability of the search engine that we built, the following capabilities can be added

(1) Ranking the search based on relevancy: Right now, our aim was solely on the execution of the query and displaying the corresponding output. But, in reality, the results of the query should be sorted according to some relevancy metric. A naive way of achieving this would be sorting based on frequency of occurrence of the key word. But there would be more sophisticated way achieving the same which would be a great thing to look in the future.
(2) Finding similar datasets for the user input dataset : As we have seen, the data summaries that we have created are very informative and flexible and hence, they could be used to find similar datasets to the input dataset name. One of the way of achieving this could be by finding the similarities between the columns among the two datasets. There could be other ways based on words and its counts which could be helpful to find the similar datasets.

9 CONCLUSION

We built a search engine that performs a task of data discovery in NYC open data. Our search engine was found to be superior to NYC open data website search engine in terms of relevancy of search results and functionalities. The search engine built by us was able to support many advanced queries that were not yet supported by the website's search engine. We approached the problem by building data summaries which captured all the relevant information of a dataset. These data summaries are easy to build, update and query. Such data summaries can also be tailored for user specific needs and we also see potential use cases of these summaries beyond searching a dataset based on a keyword. It could be used to find related tables or related columns in a repository of datasets.

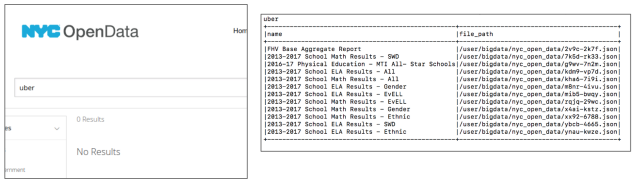
REFERENCES

[1] Dasu, Tamraparni, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. "Mining Database Structure; Or, How to Build a Data Quality Browser." Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data - SIGMOD 02, 2002. doi:10.1145/564691.564719.
[2] A.Monge and P.Elkan.An efficient domain-independent algorithm for detecting approximately duplicate database records.In Proc. SIGMOD Workshop on Research Issues in Data Mining and Know ledge Discovery,1997
[3] A.Monge.The field matching problem:Algorithms and applications.IEEE Data Engineering Bulletin, 23(4):14-20,2000
[4] L.Gravano,P.Ipeirotis,H.Jagadish,N.Koudas,S.Muthukrishnan,and D.Srivastava.Approximate string joins in a database (almost)for free.In Proc. Intl.Conf.VLDB,2001
[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

10 APPENDIX

```
If you are looking for general datasets, take a look at our famous datasets
*********************************************************************************************
5 most viewed datasets:
-----------------------
Dataset Name: Completed Percent for Art projects with artist information,Link to Dataset: http://www.nyc.gov/html/dcla/html/panyc/projects.shtm
l
Dataset Name: NYPD Motor Vehicle Collisions,Link to Dataset: null
Dataset Name: Disposition Of Discourtesy Allegations 2005 - 2009,Link to Dataset: null
Dataset Name: Inventory of New York City Greenhouse Gas Emissions - Heating And Cooling Degree Days (2004-2016),Link to Dataset: null
Dataset Name: Local Law 44 - Tax Incentive,Link to Dataset: null

                    ##############################################
5 most downloaded datasets:
---------------------------
Dataset Name: 2006 - 2012  Math Test Results - District - All Students,Link to Dataset: http://schools.nyc.gov/NR/rdonlyres/3E439063-A1AE-486E-
B019-8634A8915D71/0/DistrictMathResults20062012Public.xlsx
Dataset Name: Recreational Boating Permits,Link to Dataset: null
Dataset Name: 2006 - 2012 English Language Arts (ELA) Test Results - Charter Schools,Link to Dataset: http://schools.nyc.gov/NR/rdonlyres/D3DA5
7F8-DC92-4E89-B954-CD36A75FC8D4/0/CharterSchoolMathandELAResults20062012Public.xlsx
Dataset Name: DOP Adult Investigations by Calendar Year,Link to Dataset: null
Dataset Name: Personal Income By AGI Range,Link to Dataset: null

*********************************************************************************************


Else,
Choose one of the query types


1 Search for  keyword
2 Search for a keyword with threshold on number of times it should occur
3 Search for a keyword with threshold on number of rows the datasets should contain
4 Dataset belonging to certain category
5 Dataset with certain number of rows and columns
6 See the Schema of Dataset
7 Search for multiple keywords in OR fashion
8 Searcg for multiple keywords in AND fashion
9 Search for files with certain columns in it
10 Search for files with certain column and value
11 Search keywords in meta data and data


Enter Your Choice
```

Figure 4: A snapshot of the Command Line Interface for the search engine




```
uber
+--------------------------------------------+--------------------------------------------+
|name                                        |file_path                                   |
+--------------------------------------------+--------------------------------------------+
|FHV Base Aggregate Report                   |/user/bigdata/nyc_open_data/2v9c-2k7f.json|
|2013-2017 School Math Results - SWD         |/user/bigdata/nyc_open_data/7mSd-rk33.json|
|2016-17 Physical Education - MTI All- Star Schools|/user/bigdata/nyc_open_data/g9ev-7n2m.json|
|2013-2017 School ELA Results - All          |/user/bigdata/nyc_open_data/kd9f-uy7d.json|
|2013-2017 School Math Results - All         |/user/bigdata/nyc_open_data/khm6-719i.json|
|2013-2017 School ELA Results - Gender       |/user/bigdata/nyc_open_data/m8nr-4ivu.json|
|2013-2017 School ELA Results - EvELL        |/user/bigdata/nyc_open_data/mib8-bmqy.json|
|2013-2017 School Math Results - EvELL       |/user/bigdata/nyc_open_data/rzjq-29wc.json|
|2013-2017 School Math Results - Gender      |/user/bigdata/nyc_open_data/x4xi-kktz.json|
|2013-2017 School Math Results - Ethnic      |/user/bigdata/nyc_open_data/xx92-6788.json|
|2013-2017 School ELA Results - SWD          |/user/bigdata/nyc_open_data/ybcb-4665.json|
|2013-2017 School ELA Results - Ethnic       |/user/bigdata/nyc_open_data/ynau-kwze.json|
+--------------------------------------------+--------------------------------------------+
```

Figure 5: Comparison of Search for word "uber". NYC open data API does not find the relevant datasets as opposed to our search engine which returns multiple datasets.

```
+--------------------------------------------+--------------------------------------------+
|path                                        |name                                        |
+--------------------------------------------+--------------------------------------------+
|/user/bigdata/nyc_open_data/gffu-ps8j.json|Income By Type Of Income And AGI Range|
|/user/bigdata/nyc_open_data/nwet-nc6h.json|Tax Credits By Agi Range                    |
|/user/bigdata/nyc_open_data/3vvi-fwjs.json|Tax Liability By AGI Range                  |
|/user/bigdata/nyc_open_data/ipc3-2nbm.json|Personal Income By AGI Range                |
|/user/bigdata/nyc_open_data/9ay9-xkek.json|Local Law 44 - Unit Income Rent             |
+--------------------------------------------+--------------------------------------------+
```

Figure 6: Results for "income" as a field name

```
+--------------------------------------------+--------------------------------------------+
|path                                        |name                                        |
+--------------------------------------------+--------------------------------------------+
|/user/bigdata/nyc_open_data/9a87-6m4x.json|SBS ICAP Contract Opportunities             |
|/user/bigdata/nyc_open_data/u35m-9t32.json|DCLA Cultural Organizations                 |
|/user/bigdata/nyc_open_data/m3fi-rt3k.json|New York City Council Discretionary Funding (2009-2013)|
|/user/bigdata/nyc_open_data/un8d-rbed.json|SBS ICAP Contract Opportunities - Historical|
|/user/bigdata/nyc_open_data/ye3c-m4ga.json|Civil List                                  |
+--------------------------------------------+--------------------------------------------+
```

Figure 7: Datasets with a column that contain a keyword "address" with keyword "building" in it.

```
+--------------------------------------------+--------------------------------------------+
|file_name                                   |name                                        |
+--------------------------------------------+--------------------------------------------+
|/user/bigdata/nyc_open_data/8gpu-s594.json|Application for State Aid                    |
|/user/bigdata/nyc_open_data/tm6d-hbzd.json|Incidents Responded to by Fire Companies    |
|/user/bigdata/nyc_open_data/gshi-yqza.json|Transportable Classroom Units- Buildings & Schools|
+--------------------------------------------+--------------------------------------------+
```

Figure 8: Datasets that contains word "classroom" at least 50 times.

```
['alcohol']
['drinking', 'marijuana', 'alcohol']
+--------------------------------------------+--------------------------------------------+
|file_name                                   |name                                        |
+--------------------------------------------+--------------------------------------------+
|/user/bigdata/nyc_open_data/tm6d-hbzd.json|Incidents Responded to by Fire Companies    |
|/user/bigdata/nyc_open_data/8sdw-8vja.json|Asset Management Parks System (AMPS) - Work Orders|
|/user/bigdata/nyc_open_data/m3fi-rt3k.json|New York City Council Discretionary Funding (2009-2013)|
|/user/bigdata/nyc_open_data/6r4h-c2y6.json|Sustainability Indicators (2012)            |
|/user/bigdata/nyc_open_data/8nqg-ia7v.json|Mental Health Service Finder Data           |
+--------------------------------------------+--------------------------------------------+
```

Figure 9: Datasets that contains "alcohol". Since we used the query template (6) for this search, the algorithm returns datasets which contain either "alcohol" or its similar words, i.e. "drinking" and "marijuana".