Search Engine for Academia

Amey Kulkarni
18110016
amey.kulkarni@iitgn.ac.in

Chris Francis
18110041
chris.francis@iitgn.ac.in

Nishikant Parmar
18110108
nishikant.parmar@iitgn.ac.in

Abstract

Students often find the need to search for professors based on various criteria such as name, university, research topics of interest, top citated papers and rank them based on things such as their citations and h-index. The existing tools for searching for professors do not provide enough specificity, that is often needed in Academia. A simple Google search may not allow you to first shortlist professors based on whether they do research in "adversarial machine learning" and then rank them according to the number of citations that they have. We have come up with a search engine powerful enough to search based on three different querying retrieval techniques as per user opt. We have built a lightweight, but powerful search engine that provides users with plenty of options to find the right professor out of the available 13285.

1 Introduction

blah blah blah

2 High Level Architectural Design

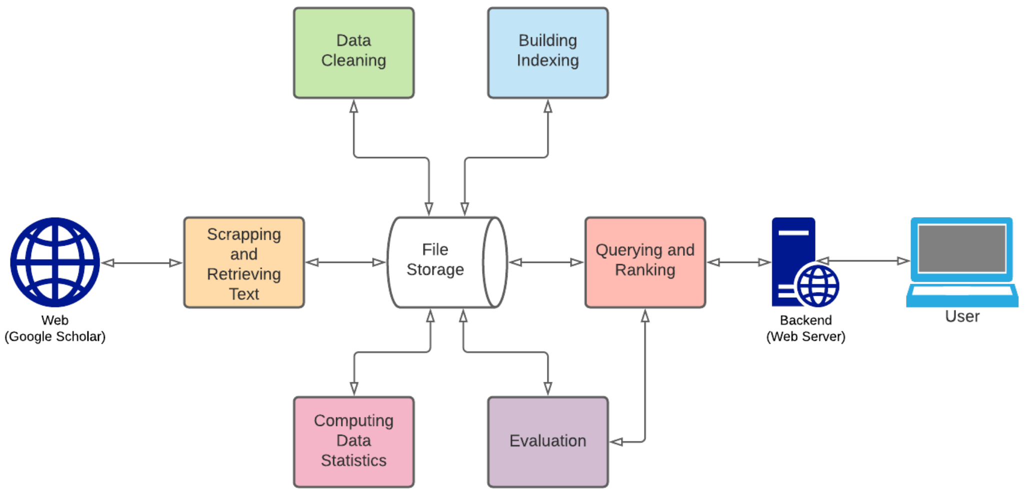

Figure 1: High Level Architecture

2.1 Scraping and Retrieving Text

This module takes as input the list of scholar ids splitted across 10 files, scrape the data from Google Scholar web page of the professors and stores them in CSV format.

2.2 Cleaning Data

This module takes as input the output of previous module, cleans it up, removes redundant data and stores them in CSV format.

2.3 Building Index

This module receives data in the correct format from the "Cleaning Data" module, builds two inverted indices (first for name and affiliation, second for topics and paper titles) and stores it in a JSON format.

2.4 Query Processing and Ranking

This module receives query information from the back end, processes it and returns an ordered list of web pages to the user depending upon the specifications provided by the user.

2.5 Backend (Web Server)

This module forwards the user's query request to the "Query Processing and Ranking" module and act as a intermediate to user's browser.

2.6 User

The user types a query, specifies the retrieval method(Boolean, phrase or TF-IDF) and also the context in which he wants the results(names and affiliations or topics and paper titles).

2.7 Computing Data Statistics

This module is not part of the main pipeline of search engine but acts as a different component that takes cleaned data generated from "Cleaning Data" module as input, computes various statistics and stores them in the form of a plots.

2.8 Evaluation

It generates its own queries, runs the queries using "Querying and Ranking" module, evaluates search results and stores the statistics obtain in the form of plot. This module is also not part of the main pipeline of search engine.

3 Data Collection and Pre-processing

In order to obtain data for enough professors, we used the following Github repository: `https://github.com/emeryberger/CSrankings`. Here we found 20000 professors along with their university and Google Scholar id page. However some of the professors did not have a Google Scholar page, some who had the Google Scholar id but page was not available (Error 404).

3.1 Web Scraping

We planned to extract key information for each professor from their respective google scholar pages. Upon analysing the web pages for various professors, we identified the HTML tags which carried important information. For this task we used Python's Beautiful Soup, an HTML parser. We then wrote this data onto a csv files.

For each professor we extracted these information:

- Name
- Affiliation
- Profile Image URL
- Verified Email at
- Personal Homepage URL
- Research Topics List
- Citation
- H-Index
- I-Index
- Citation (Past 5 Years)
- H-Index (Past 5 Years)
- I-Index (Past 5 Years)
- Citation List Yearwise
- List of Titles of Top 100 Citated Papers
- List of URL of Top 100 Citated Papers

For very few Professors some fields out the above mentioned were absent, and we represented that in our data using empty string/list, or -1 (if integer).

3.2 Challenges

Scraping Google Scholar pages using a brute force approach lead us to getting blocked. We circumvented this problem by scraping using different IPs. Also, we added a small delay of 1, 2 or 3 seconds to avoid the same problem. It took a combined time of approximately 30 hours to scrape all the data from the Google Scholar web pages. However, since we parallelised this process through the use of different IPs, we were able to do this in around 10 hours.

3.3 Data Cleaning

After scraping the data we realized that some pages on Google Scholar were inherently problematic, that is, they did not follow the same HTML conventions as the rest of the pages.

Such entries would make our stored data corrupt. To handle this, we wrote a cleaning script and with the help of Regex we removed such entries.

Also, we realized after scraping that their were redundant entries in the Github repository mentioned above which we used to take the Google scholar ids of Professors. Such entries were also removed in the data cleaning process.

Finally, out of 20000 Google Scholar ids mentioned in Github repository, the Google Scholar page actually existed for 16580 Professors and after removing redundant entries and cleaning the corrupted part, finally we had data for 13285 Professors.

4 Data Statistics

Some basic statistics about the data collected is presented in Table 1

Statistics related to number of citations, h-index and i10-index are presented in Figures 1, 2 and 3.

Figures 4, 5 and 6 show that most professors have mentioned their affiliated institution, verified their e-mail address and provided the url to their homepage.

5 Indexing

We created two indexes: one for name and affiliation and one for research topics and paper titles of the Professors.

Table 1: A basic description of the data collected

Quantity	Value
Number of professors	13285
Number of institutions	8716
Number of publications	893575
Average number of publications per professor	79.80
Size of entire Professors' data (After cleaning)	232 MB

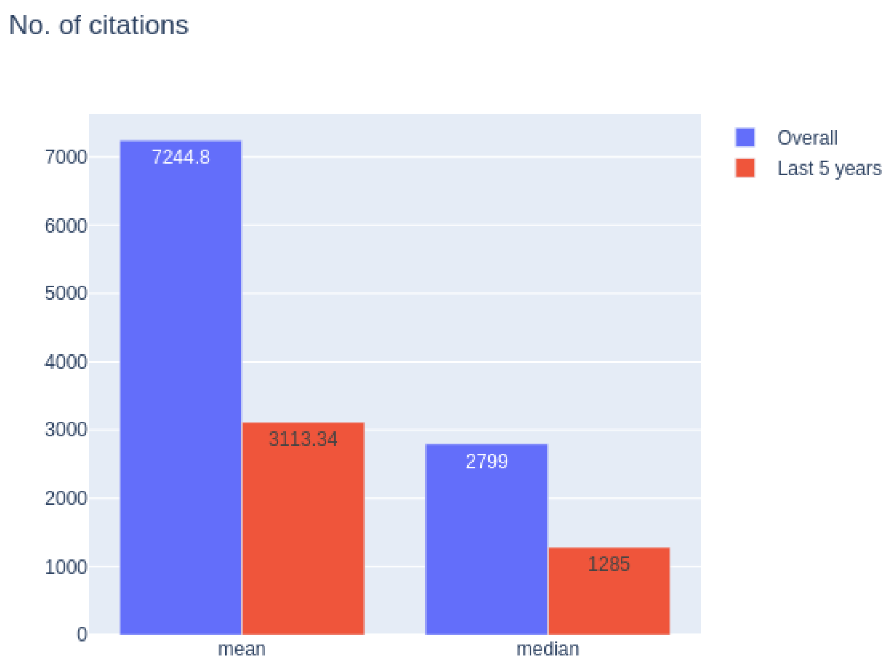

Figure 2: Number of citations

- Name and Affiliation
- Research Topics and Paper Titles

5.1 Pre-Processing

Typical search engines are case-insensitive, hence we convert all words to the lower case. We apply stemming and lemmatisation to make the search engine more flexible. Now, it will not only be able to search if the exact query word appears on a page, but also if a similar word appears.

5.2 Building Index

In order to be able to create a search engine, we had to first create an inverted index. We followed the given format: For any given word, we stored a list of two-dimensional tuples, with the first entry being the docID or the row at which it appeared in our CSV, and the second denoting which word it was in the given row, from the left. Our inverted index could also be thought of as a list of postings list for all the words that appeared in our dictionary. (A postings list is the set of all documents where a given word can be found.) This inverted index would form the base of all our ranking and filtering operations.

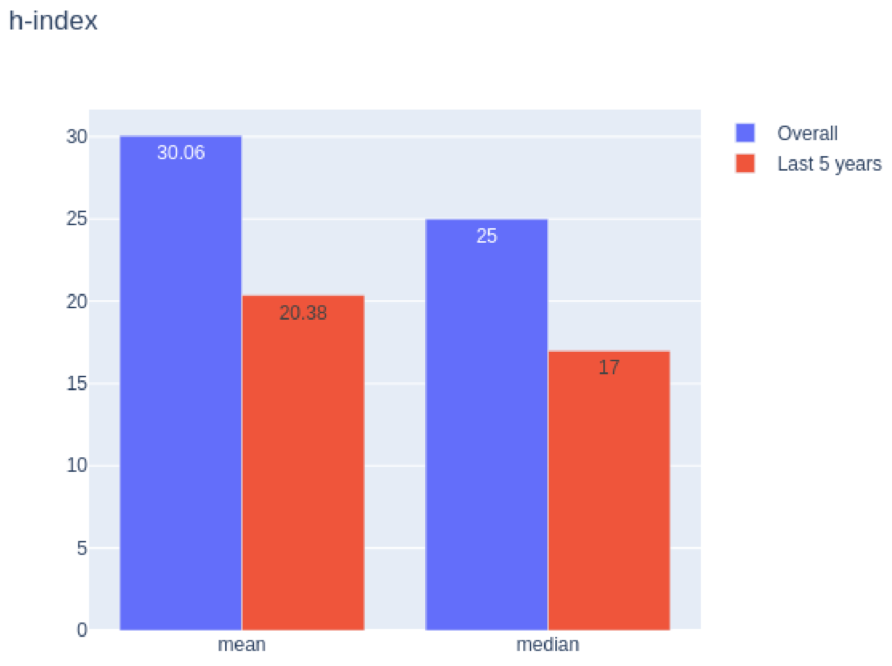

Figure 3: h-index

6 Querying and Ranking

6.1 Boolean Retrieval

Having organised our data, we now began accepting queries. Based on these queries, we filtered (or shortlisted) only certain professors who satisfied the criteria specified in the query. We implemented the following variants of the Boolean retrieval task-

6.2 AND

Only those professors would be shortlisted who had all the words from the search query on their scholar page. To do this, we simply compared the postings lists of all the words in the search query pairwise, and stored the intersection of all of them.

6.3 Optimisation

Since we are only interested in the final postings list, that is, the list containing all the words in the phrase query, the order in which we build this does not matter. Hence, to speed up the process, we sort the postings lists in the increasing order of length. Since we use the intersection of two postings lists in the next iteration, this ensures the fewest number of computations overall.

6.4 OR

Here we return the pages of all the professors who even contain one word from the search query. However, we sort matches according to the number of matches they have, that is, if a professor's page matches 4 out of 5 words from the search query, it will be displayed higher than a professor's page which matches 3 out of 5 words from the query. In case two pages match the same number of words from a search query, higher precedence will be given to the page that matches all the words in the

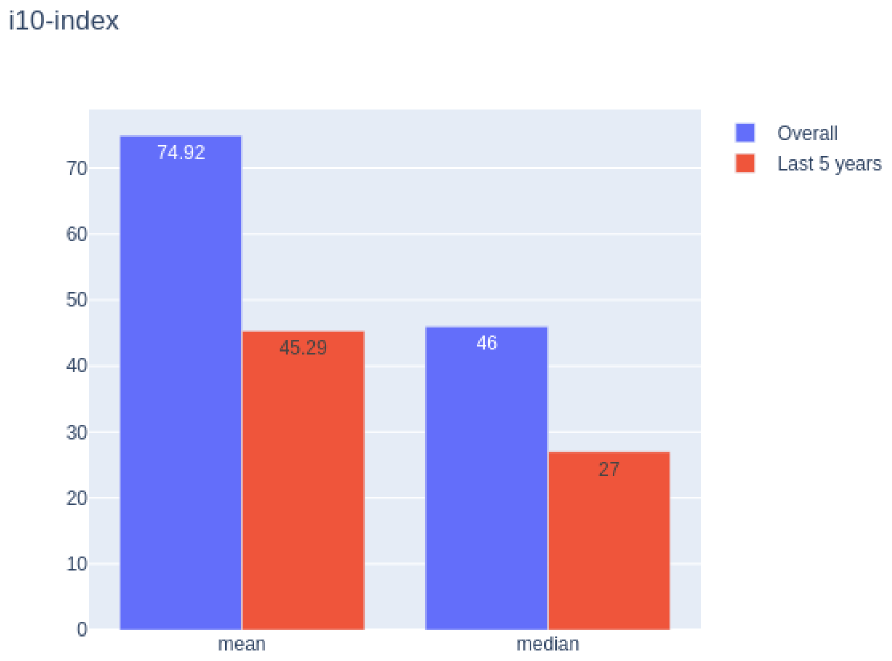

Figure 4: i10-index

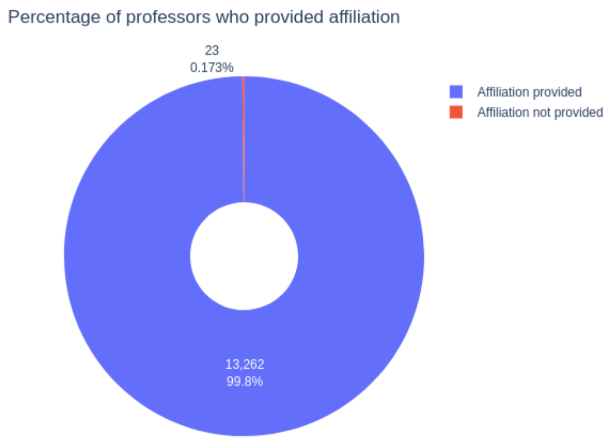

Figure 5:

query more times. For example, the word "science" may appear on a page more than once, this will increase the tiebreaker’s score.

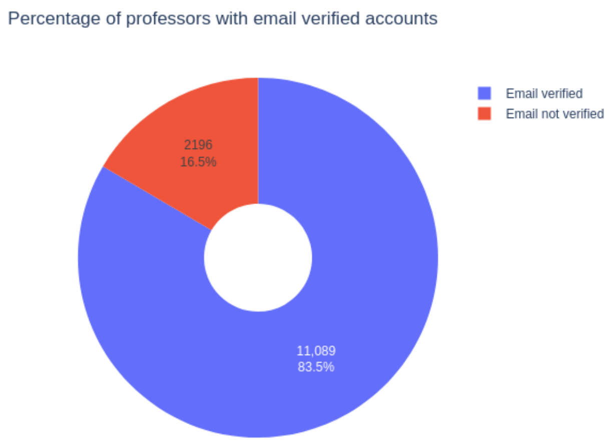

Figure 6:

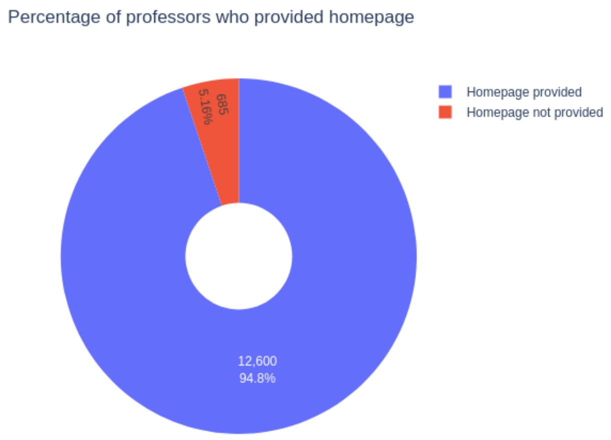

Figure 7:

6.5 Phrase Retrieval

We also added an extra feature, allowing a user to search for exactly the phrase that they have typed. This is done by storing a new variable called distance in our ordinary Boolean retrieval, which denotes the distance at which the two words appear in the search query. This is the same distance at which they must appear in the postings lists. Remember, the second parameter in the every element of our postings list denotes the position the word appears in the given web page. So once we know that the two words belong to the same page, we can check if they are at the required distance from each other.

6.6 TF-IDF Scores

blah blah blah

7 Web Server

The web server is an HTTP server built on Flask Framework in Python. It serves following purposes -

- It provides a convenient interface for users to actually use the search engine. The server hosts an HTML web page where user can enter their search query, select the query retrieval method out of Boolean Retrieval, Phrase Retrieval and TF-IDF, and then select search context i.e. name and affiliation or research topics and paper title.
- This information is then sent to the server using POST HTTP method, the search results are then computed based on query method and search context using "Querying and Ranking" module and the response of search results is rendered into the browser.
- The user can now see all the information of Professors in search results like name, affiliation, image, research topics, list of top 100 citated papers. User can follow the url to visit homepage and google scholar page of the Professor. User can also click on any paper of any Professor and see entire detail of that paper.
- Users can also see citations, h-index, i-index - past 5 years and overall both. Through animated vizual histogram they can see the number of citation yearwise for each Professor.

8 Evaluation Methods

8.1 Median Rank

Consider a user who has a particular professor in his mind, he using some information of that professor like name, affiliation, or title of a paper of that professor, and by choosing appropriate querying method (mentioned in Querying and Ranking) searches his query and gets results. Now, we define the rank as the position where the professor he had in mind shows up in the search results. Here, the professor in his mind becomes the ground truth.

We generated random 500 professors out of the entire data set that we had. These become our ground truth. Now, for each of the professor we run queries on our search engine using professor name as search query and Boolean retrieval, Phrase retrieval as query method, affiliation as search query and Boolean retrieval, Phrase retrieval as query method, and a random paper title of the professor as search query and Boolean retrieval, Phrase retrieval and TF-IDF as query method.

Since, we already have the unique id for the ground truth professor we now find the rank at which the professor appears in search results for each of these and then take *median* over 500 such professors for each search query and method type pair.

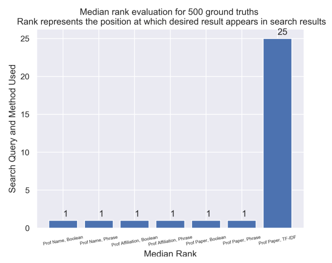

Figure 8: Median Rank

Phrase retrieval since it utilizes proximity of words appearing in query and the information present in data set performs better, followed by boolean AND. TF-IDF considers all the professors where atleast one word from the search query appears hence, the number of search results is large. As there can be multiple professors where words from paper title appear and make larger tf-idf score, hence we get slighly higher median rank for tf-idf with paper title as search query.

The definition of median rank suggests that lower the median rank, the better are search results for user. Ideally, median rank 1 means the user gets first search result which he was looking for.

8.2 Recall Rate

Along with median rank, we calculated the percentage (out of 500 professors) the ground truth appears in top X search results. This we call as Recall Rate at X. We computed Recall Rate at 5 and 10.

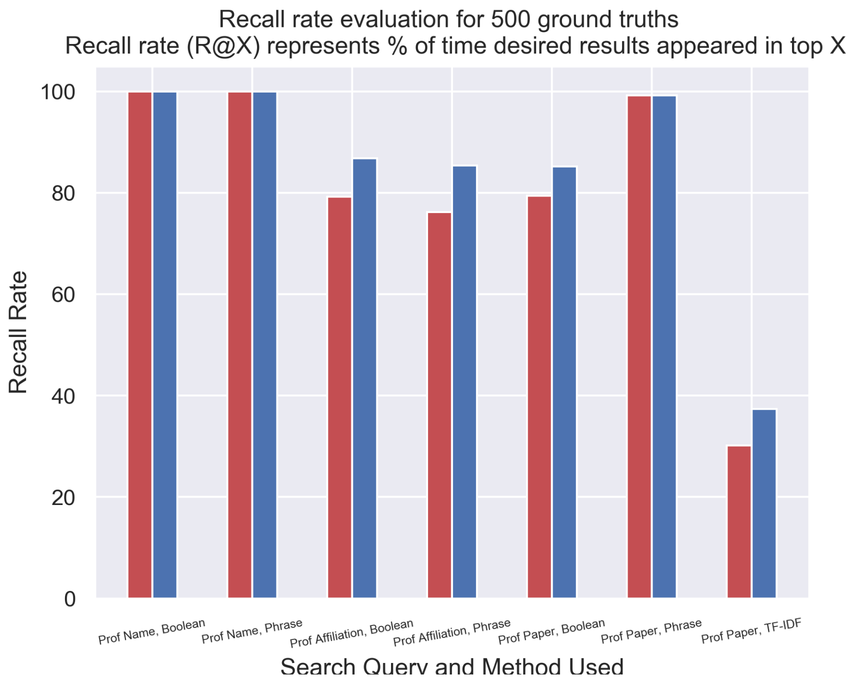

Figure 9: Recall Rate

The recall rate follows similar pattern as median rank.

9 Conclusion