Acad Search: Search Engine for Academia

Amey Kulkarni 18110016 **Chris Francis** 18110041 **Nishikant Parmar** 18110108

Abstract

Students often find the need to search for professors based on various criteria such as name, university, research topics, top cited papers and rank them based on factors like citations or h-index. 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 in the last 5 years. We have developed a search engine that can cater to the needs of students looking for professors to approach for projects, internships or jobs. The search engine allows users to search for professors based on name, university, research areas and paper titles using 3 different retrieval methods. The engine also allows users to sort the search results based on criteria like h-index, citations in the last 5 years etc. We have deployed the search engine publicly as a web application, and also evaluated its performance in terms of time and quality of results.

1 Introduction

Finding professors to apply for internships, research projects or jobs is a common and time-consuming task among students. Students may value different things in such a context and it is hard to perform this search with existing tools. We propose a simple search engine that can help students to search for professors based on various criteria such as name, university, research topics of interest, top cited papers and sort the results based on criteria such as the number of citations or h-index.

The best solution currently in use is Google Scholar, in which the Profile Search feature can search for professors. However it does not allow you to sort or filter your search results based on any criteria. However, Google Scholar does have a huge collection of data on professors, researchers and their research. We have used this data to create a tool more focused towards students who are searching for professors. We provide the user with more control over the search results with the help of multiple retrieval methods and criteria-based sorting of search results.

2 High Level Architectural Design

2.1 Scraping and Retrieving Text

This module uses the list of Google Scholar IDs of professors(from CSRankings, split across 10 files), scrapes data from their Google Scholar pages and stores it as CSV.

2.2 Cleaning Data

This module cleans the scraped data from the previous module and stores it as CSV.

2.3 Building Index

This module uses the cleaned data to build two inverted indices (first for name and affiliation, second for topics and paper titles) and stores them as JSON.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

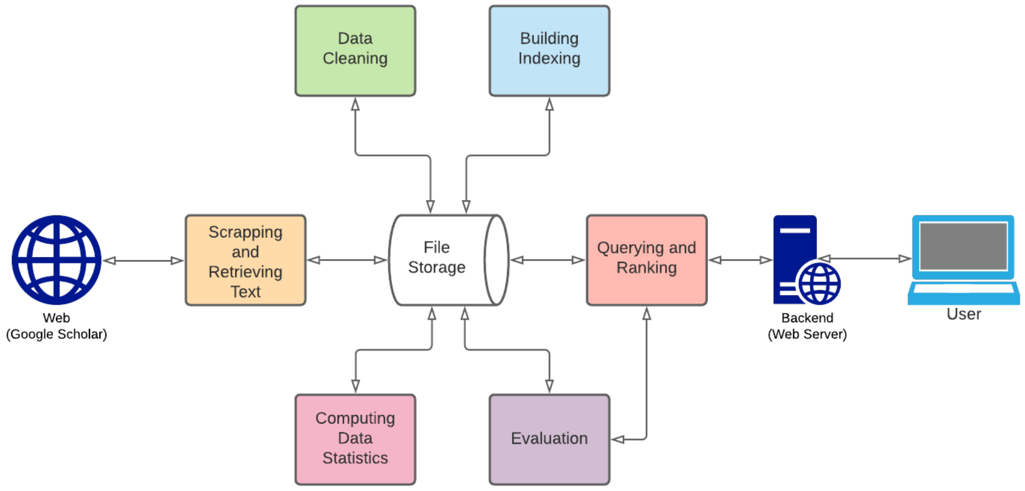

Figure 1: High Level Architecture

2.4 Query Processing and Ranking

This module receives query information from the "Backend", processes it and returns an ordered list of professors depending upon the specifications provided by the user.

2.5 Backend (Web Server)

This module forwards the user's query to the "Query Processing and Ranking" module and acts 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 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 the search engine. It computes and plots various statistics of the cleaned dataset.

2.8 Evaluation

It generates its own queries, runs the queries using "Querying and Ranking" module, evaluates search results and plots the evaluation metrics. This module is also not part of the main pipeline.

3 Data Collection and Pre-processing

We obtained a list of around 20000 professors and their Google Scholar IDs from the CSRankings repository[1]: `https://github.com/emeryberger/CSrankings`. Some of them did not have a Google Scholar ID, and some IDs were obsolete(page no longer available, resulted in error 404).

3.1 Web Scraping

We extracted key information for each professor from their respective Google Scholar pages. We used the Beautiful Soup Python library, an HTML parser. We stored the scraped data as CSV.

Information scraped for each professor:

- Name
- Affiliation/Institute
- Profile image URL
- Verified email domain
- Personal homepage URL
- Research topics list
- No. of citations
- h-index
- i10-index
- No. of citations (past 5 years)
- h-index (past 5 years)
- i10-index (past 5 years)
- Year-wise no. of citations
- List of titles of top 100 cited papers
- List of URLs of top 100 cited papers

We represented missing data using empty string/list, or -1(if integer) and handled them in the computations.

3.2 Challenges

We avoided getting blocked while scraping by using different IP addresses. We also added a small delay of 1, 2 or 3 seconds and included request headers to act like a browser. It should have taken us 30 hours for scraping, but we completed it in 10 hours by parallelising across multiple IP addresses.

3.3 Data Cleaning

Some pages on Google Scholar did not follow the same HTML conventions as the rest, leading to corruption in data. We cleaned such entries using regular expressions. We also removed repetitions of the same professor that arose due to redundancies in the CSRankings repository. Finally, out of around 20000 Google Scholar IDs obtained from the repository, 16580 professors had Google Scholar IDs and after removing redundant entries and cleaning the corruptions, finally we had data for 13285 professors.

4 Data Statistics

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

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 dataset(cleaned)	232 MB

Statistics related to number of citations, h-index and i10-index are presented in Figure 2. Figure 3 shows that most professors have mentioned their affiliated institution, verified their e-mail address and provided the URL to their homepage. Figure 4 shows that the frequency distribution follows a power law, i.e., the number of words(in the topics and paper titles index) with frequency k is C/k^n where C and n are constants.

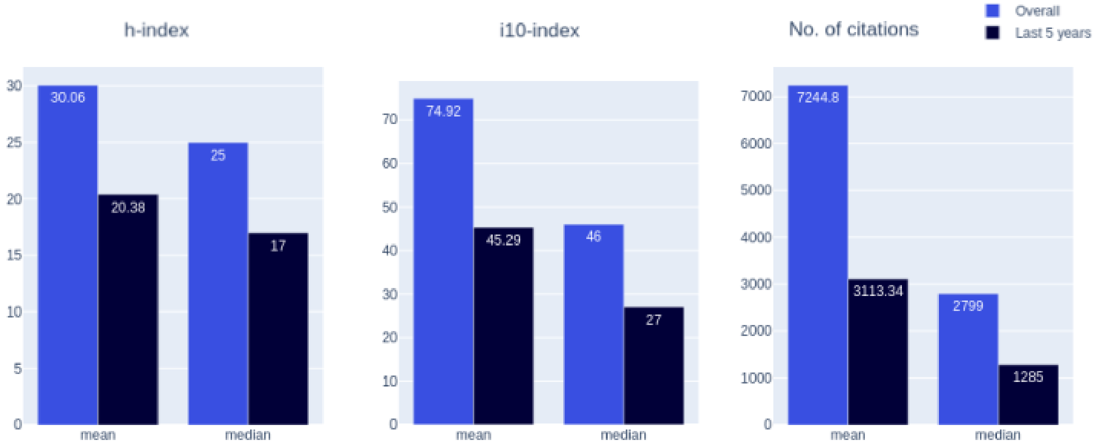

Figure 2:

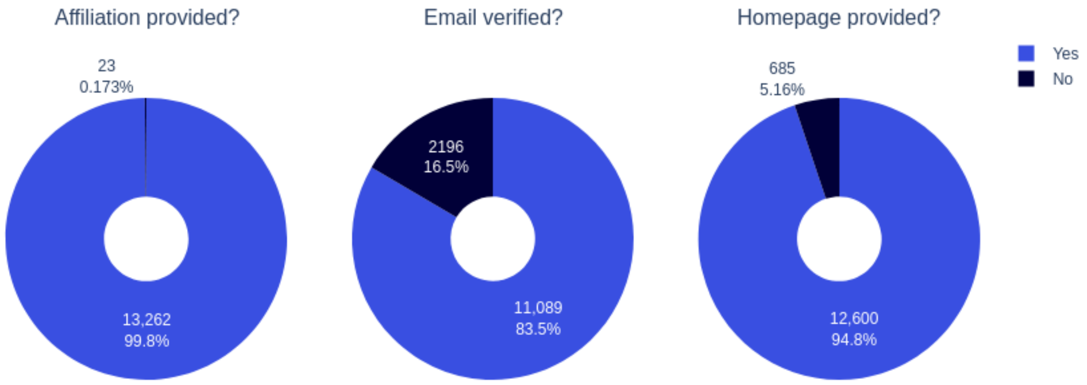

Figure 3:

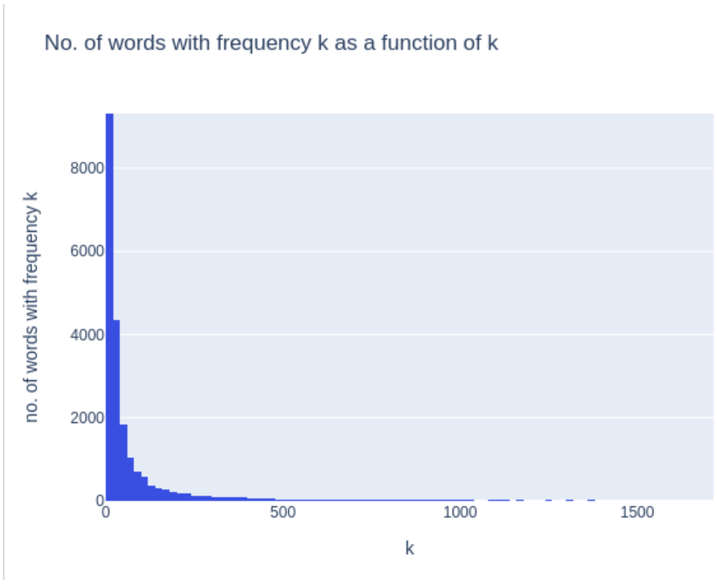

Figure 4:

5 Indexing

We created two indices: one for name and affiliation, and another for research topics and paper titles.

5.1 Pre-Processing

Typical search engines are case-insensitive, hence we converted all words to lower case. For research topics and paper titles index, we also removed stop words and applied stemming 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. We did not perform stemming and stop words removal for name and affiliation index.

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 unique professor id (where that word appears), and the second denoting the positional index at which the word appears in the information of the professor, 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. We stored the two indices as JSON files.

6 Querying and Ranking

Along with the search query, we also take the retrieval method and the search context i.e. whether they want to search in name and affiliation space or research topics and paper title space, as inputs. We then use the corresponding index and retrieval method. We implemented three different techniques for querying:

6.1 Boolean Retrieval

We implemented the following variants of Boolean Retrieval:

6.1.1 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.

6.1.2 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.

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

6.2 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. The second parameter in the every element of our postings list(inverted index) denotes the position the word appears in the given web page. So once we know that two words belong to the same page, we can check if they are at the required distance from each other.

6.3 TF-IDF Scores

Users can also get search results based on Term Frequency-Inverse Document Frequency (TF-IDF) scores [2]. It is a numerical statistic widely used in information retrieval and text mining to reflect how important a word is to a document in a collection or corpus. We consider professors as documents. Term Frequency is a measure of how frequently a word/term appears in a document, normalised by the document size.

Term Frequency of term t in a document d is, TF$(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$

where $f_{t,d}$ is the number of times the term/word t appears in document d. The denominator is the total number of words in the document. Inverse document frequency is a measure of how much information a word provides, i.e., if it's common or rare across all documents.

Inverse Document Frequency of term t is, IDF$(t, D) =$ log $\left[\frac{N}{1+|\{d \in D : t \in d\}|}\right]$

where D is the corpus of documents and N is the total number of documents in the corpus D. The denominator is 1 more than the number of documents that contain the term t. The addition of 1 is to avoid division by zero. TF-IDF is calculated as

TF-IDF$(t, d, D) =$ TF$(t, d) \times$ IDF(t, D)

We compute TF-IDF scores of all word-document pairs using the inverted index for research topics and paper titles and store the scores as a JSON file. The scores are stored in a format similar to an adjacency list rather than something similar to a matrix, to improve space and time requirements. When a user queries for TF-IDF based results, we return the documents(professors) with non-zero scores sorted according to the score.

6.4 Ranking Metric

For ranking the search results, in TF-IDF we rank results in decreasing order of their TF-IDF scores. Despite Boolean and Phrase Retrieval being quite robust in finding relevant documents, they just provide a *binary score* to a document i.e. either the document will be present in search results or not. For these two, we propose the following criterion for ranking search results:

score = $\alpha\times$ h-index $+$ $\beta\times$ i10-index $+$ $\gamma\times$ no. of citations $+$ $\delta\times$ h-index(past 5 years) $+$ $\zeta\times$ i10-index(past 5 years) $+$ $\eta\times$ no. of citations(past 5 years)

where $\alpha, \beta, \gamma, \delta, \zeta, \eta$ are weights. Ranking using this criterion is performed while showing results through web server in browser and not for evaluation methods.

7 Web Server

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

- It acts as an interface for users to use the search engine. The server hosts an HTML web page where users can enter their search query, select the query retrieval method(Boolean Retrieval, Phrase Retrieval or TF-IDF), and then select search context(names-affiliations or research topics-paper titles).

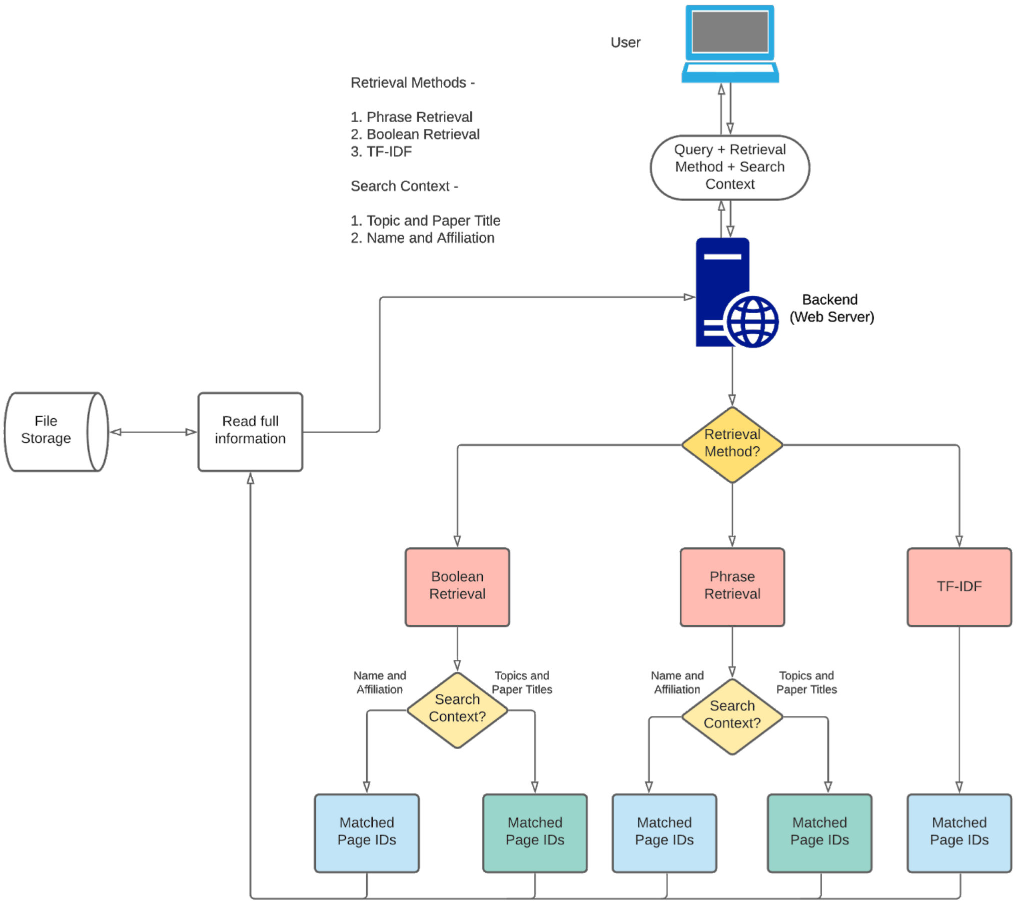

Figure 5: Flow Diagram

- 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 in 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 cited papers. User can follow the URLs to visit homepage and Google Scholar page of the professor. User can also click on any paper of any professor and view its details.
- Users can also see citations, h-index, i-index - both past 5 years and overall. Through animated visual histogram they can see the year-wise number of citations for each professor.
- Users can sort the search results using h-index, i-index, and citations - both past 5 years and overall. They can also use the default ranking provided by the search engine. In case of TF-IDF this ranking is based on TF-IDF score and for Boolean and Phrase Retrieval it is calculated using a default ranking metric as mentioned in the previous section.

8 Evaluation

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 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 is the ground truth.

We created 500 random queries and recorded the rank achieved by the ground truth among the search results. We performed this 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.

8.1 Median Rank

The *median* of the ranks obtained by the ground truth is called the *median rank*.

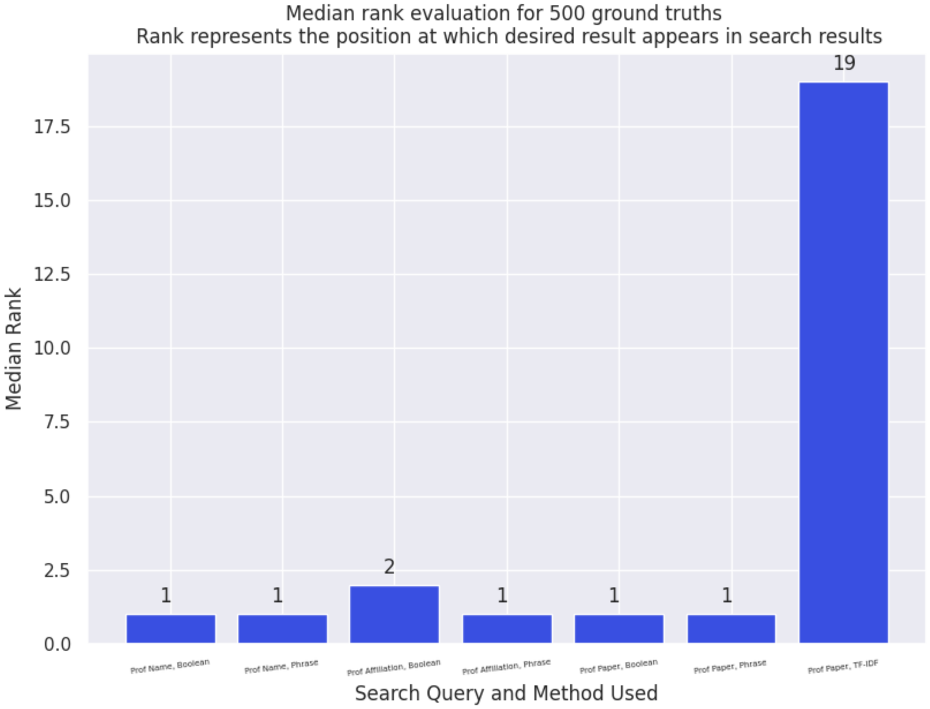

Figure 6: Median Rank

Figure 6 shows the median rank for each combination. The X-axis represents the search query and method type pair, and Y-axis represents the median rank obtained. Since phrase retrieval utilizes proximity of words appearing in query and the information present in data set, it performs much better even on paper titles. TF-IDF on the other hand considers all the professors where at least one word from the search query appears hence, the number of search results is very large. As there can be multiple professors where words from paper title could appear and could cause larger TF-IDF score. Hence we get slightly 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 the search results. A median rank of one means the user gets what they were looking for in the first position.

8.2 Recall Rate

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

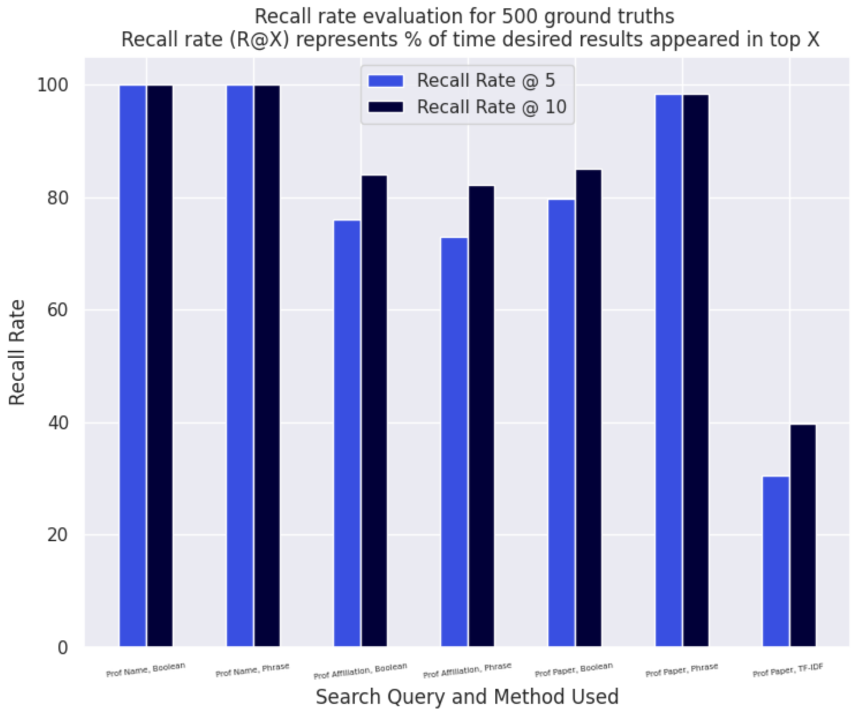

Figure 7: Recall Rate

Figure 7 shows the recall rates for each combination. The X-axis represents the search query and method type pair. The recall rate follows similar pattern as median rank. A recall rate of 100 percent would mean that user finds their desired result in top 5 or 10 *every* time they query.

8.3 Average Time Per Query

Figure 8 shows the average query time for each combination. For each search query and query method we evaluated the average time required by the algorithm to return the matched document(professor) ids.

Phrase and Boolean Retrieval perform well on search query of type professor name, affiliation and paper title. However, TF-IDF is relatively slower on paper title since it computes scores for all documents where at least one word from the search query appears.

9 Future Work

- Scraping data from homepages of professors and universities, periodically.
- Making a directed graph using citations e.g. if a professor (in one of his papers) cites another professor's paper then it can be a directed edge. This graph can then be used to implement Pagerank.
- Improving user experience by adding search history and providing suggestions based on collaborative filtering.
- Making the default ranking metric(for Phrase and Boolean Retrieval) learn-able based on user feedback on search results.
- Evaluating the search engine with real users.

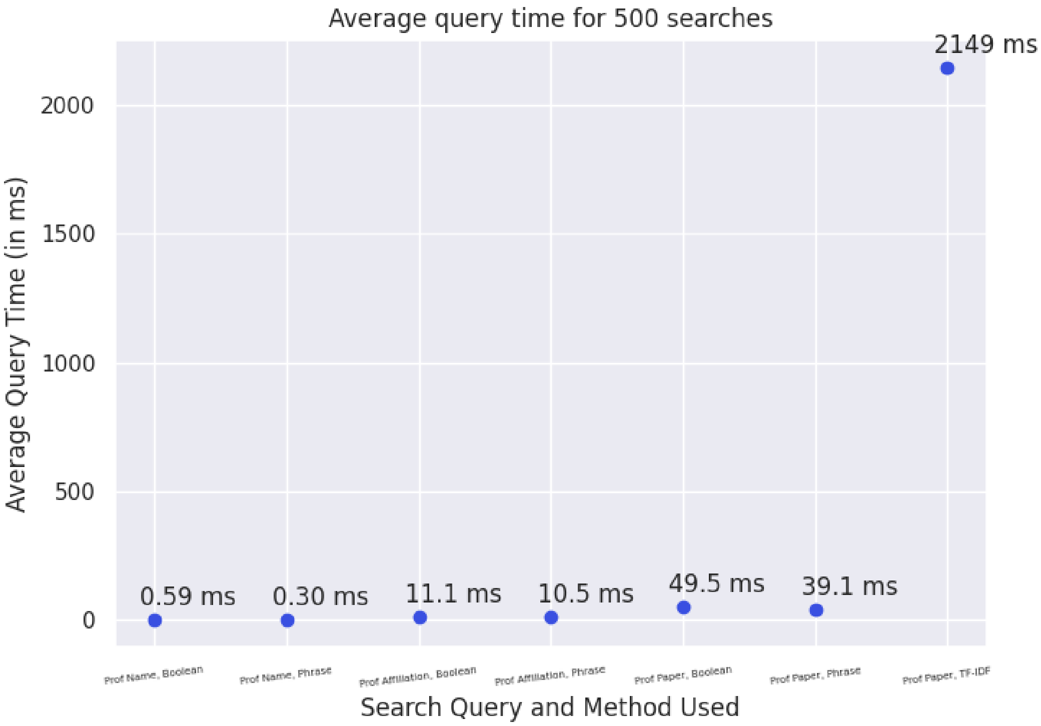

Figure 8: Average Time Per Query

10 Conclusions

In this work, we proposed a light-weight and crowd-sourced search engine for students looking for professors. We scraped data about professors from Google Scholar and created inverted indices. We also implemented TF-IDF scoring, Boolean Retrieval and Phrase Retrieval for retrieving results. We provided users with options to sort the search results based on various criteria. We also introduced a ranking metric which will be used by default to sort search results for Boolean and Phrase Retrieval. We evaluated the search engine using median rank, recall rates and timing experiments.

References

[1] Berger, E. (2017). GitHub Repository. *emeryberger/CSRankings*. `https://github.com/emeryberger/CSrankings`

[2] Rajaraman, A.; Ullman, J.D. (2011). "Data Mining" (PDF). *Mining of Massive Datasets*. pp. 1–17.