Improving Academic Information Retrieval through Document Centrality and Readability

Le Zhang
Georgia Institute Of Technology
Atlanta, GA
zhang45@gatech.edu
Learning @ Scale Work in Progress

ABSTRACT
This paper highlights a proof of concept application which combines existing methodology for document centrality and readability to improve upon academic information retrieval. The application is an addendum on top of traditional search engines and ranks search results by a combination of their centrality, readability and search engine score.

Author Keywords
Search; optimize; google; centrality; readability; lecture slides

ACM Classification Keywords
Algorithm

1. INTRODUCTION

A study performed by CERLIM reports that 45% of students rely on Google as their first source of information retrieval, as opposed to using the library[1]. An interesting summary from the CERLIM report is that students in general seek out the path of least effort. Google, and other search engines thus have become the de-facto search method due to their ease of use. A 2009 study by Mali and Mahmood further confirmed that more than half of post-secondary students used web search at least once a day [2]. The study also indicated that ease of use and relevancy were the most important features users look for in a search engine. In addition, users preferred looking at the first page of results only, with more than half of the users clicking the first 10 hits. The users judged the results in the following order: title, highlighted words, description. Students rarely look into every search result to find the best source of information as it is time consuming. In fact, Brophy performed a study that indicated that students willingly sacrifice the quality of search results in return for less effort and time spent. The study found that Google is superior to libraries in terms of coverage and ease but fell behind in quality. Often, the students cannot distinguish quality results and resort to "trial and error" on results which seem promising [3]. However, this method is tedious and does not guarantee success, especially if the search parameter is inappropriate.

In addition, a fair number of students had issues with information overload and irrelevance from their search results [1]. A study performed on Middle school student indicated that without proper research training, students naturally tend to look for the answer to a problem instead of making connections of the meaning [4].Thus, students often have to revise their search parameters in iterations before finding results that are good enough.

Even in the cases where the student can't find an exact solution, Google can be used to narrow down the process of solving a problem without having an understanding of related background. In other words, students use Google to collect information in terms of segmented blocks, sometimes bypassing the foundations that would normally be required. This can be akin to regurgitation, where the student only understands how to use Google to solve a problem but lacks a wholesome understanding on a subject.

There is awareness in the academic community of the problem of these problems. Joachims and Radlinski for instance proposed a system using machine learning which tailors search results to particular user groups. Their methodology utilizes implicit feedback from counting clicks, time browsing the hit and query chains [5]. A query chain is when a user changes his query after seeing the result for the first query. The system can learn from query chains and associate pages from the query chain even if the user hasn't included it in his search. This method completely depends on user feedback to optimize search result, which does not include looking internally at the content of the result.

Another approach more literally observes the content of the results. Structural re-ranking is the coined term by Kurland and Lee which denotes re-ranking retrieved documents using language models [10]. It goes beyond Google's PageRank algorithm, which utilizes the same concept of document centrality is based off the centrality between hyperlinks [2]. The most recent publication for this effort is from 2009, where Krikon and Kurland expand on improvements to the current methodology [6]:

1. Finding inter-document similarity and re-ranking by those that shared the most similarity.
2. Splitting up documents into passages in some rank or order, and performing retrievals based on the passages and not the whole text

Their methodology is the fusion of the two techniques which ended up being not statically any better than method 1.

Another aspect of online information retrieval I explore is on readability. Online information retrieval neglects the readability of the returned results [7]. There has been research done analyzing the effectiveness of personalizing search results by reading level [8] which indicated that users prefer document of lower reading difficulty. Again, this goes back to the user's preference of the lease resistant path.

In this paper, I present a method which attempts to ameliorate the issues students face when researching: relevancy, comprehensiveness and readability. The goal of my method is to introduce a practical proof of concept application using existing tools, rather than propose an algorithm that beats out current state of the art. The goal of the tool is to demonstrate my method's ability to make academic information retrieval more efficient for students.

I utilize cosine similarity and Flesch Reading Ease as my core algorithms to determine centrality and readability, respectively. These are older and much simpler algorithm than presented by the research previously mentioned, but they have the advantage of being computationally faster. The Flesch Reading Ease algorithm is as shown below [12]:

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right).$$

2. ALGORITHM

A practical application has a few requirements:

- Ease of use, can't be more complicated than a normal search engine. Students will not use a tool that takes more effort, going back to their preference of the path of least resistance.
- Fast, needs sub 30s run time(Heroku restriction). As a prototype, this has room for improvement later.
- Offers better results than Google and other engines, in terms more relevant and comprehensive results and rank them higher in the results
- Minimizes the need to "trial and error" through a list
- Minimizes and offers solutions to incremental searches

2.1 Custom Google Search

The application first conducts the users search through Google. To focus the search result, the application applies the following filters:

- Wikipedia, users do not need google to refer them to a wiki page
- Videos (youtube,vimeo), out of the range of text analysis. Often, top lecture results are video.
- Adds "lecture" or "notes" to the search parameter, which places greater bias on academic results

With just the simple filters, the google search already returns much more relevant results.

2.2 Document Parsing and Flesch Reading Ease

Once the initial retrieval is complete, the application downloads all of the result sources and parses for the plain text. The application accepts: pdf, ppt, doc and normal html websites. Note that the lack of pptx and docx is due to Google not having any indexing on those proprietary formats. I use available python libraries to parse the raw text and then use python's NLP library to tokenize the text.

Once the text is retrieved, I use Flesch reading ease to get a score on the readability of the document. I chose this algorithm since it is a general algorithm suitable for all types of documents. The more positive the score the simpler it is to read the document. The more negative, the more difficult.

After determining the readability, I filter out common English stop words as well as everything else besides nouns and verbs. The reason being only nouns and verbs convey something unique about the document and I need to minimize the amount of text to work with to speed up my application. Finally, the text is tokenized, broken into meaningful pieces separated by punctuation and white spaces.

To aid in this, my application runs this segment of the algorithm in multiple threads.

2.3 TF_IDF Vectors and KMeans

Next, I convert the parsed document tokens into TFIDF vectors, which represent the term frequency and inverse term frequency of a token. This process produces features that are then utilizes in determining document centrality. I use python's sklearn library to generate these vectors. The vectors on their own represents each individual document.

Next, I cluster the vectors using Kmeans into two clusters. The assumption is that Google's result will have a majority of relevant results but some will not be. The relevant results gets clustered into one, the non-relevant gets clustered into the other (See figure 1). This naïve assumption is effective only when the search parameter is appropriate. A subject with a large variety of connotations will do poorly, as this algorithm will place little weight on documents outside of the main cluster. On the flip side, results that are all similar will suffer since the documents will be forcefully divided. However, the top results are guaranteed to be relevant to the search.

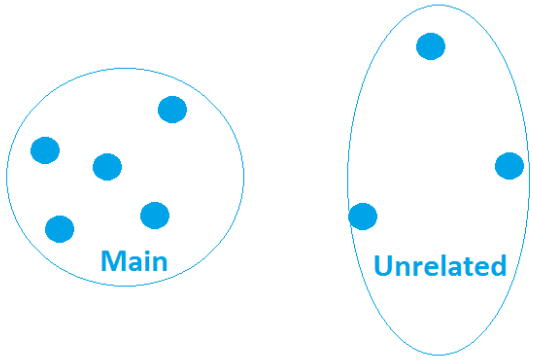

Figure 1. Example of the 2-cluster of tf-idf vectors.

2.4 Document Centrality Measure

With the main cluster determined, I use its centroid as the center point of my documents. I then iterate through each document vector and find its cosine similarity with respect to the centroid. Here I treat the centroid as a separate document tf-idf vector that represents all the documents in the main cluster.

Cosine similarity is used to determine the differences in vector orientation, not its magnitude. It is a commonly used method for document comparison [11].

After obtaining the cosine similarity for each document, I then normalize the results to prevent particular features from overshadowing others.

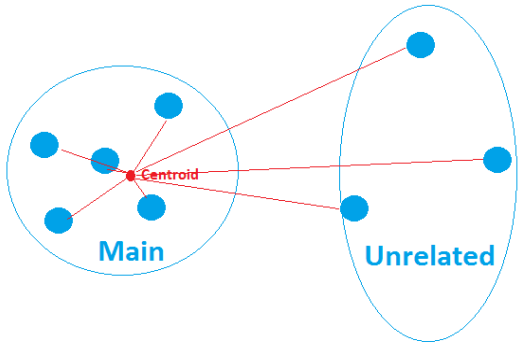

Figure 2. Centroid of main cluster (in red) is used to measure the similarity between it and each document.

2.5 Document Ranking

The final results are ranked with the following scores:

- Flesch reading ease
- Centrality score
- Google rank score

Determining the optimum balance is out of the scope of this paper, but I found success with the majority of the weight put in the centrality score, with readability and google rank as secondary. The result is then displayed on the application with each respective score as well as any errors (figure 3).

2.6 Search suggestions

As mentioned in the background discussion, users often have trouble with defining the proper search parameters. To aid with their search and minimize trial and error searches, the application provides a list of the relevant tokens that might be of interest. Google search provides something similar through their search suggestion algorithm which relies on their extensive data. For my application, only the top two document results are used to gather important words. An example is shown in Figure 4, where key words are chosen for the search "slam robotics".

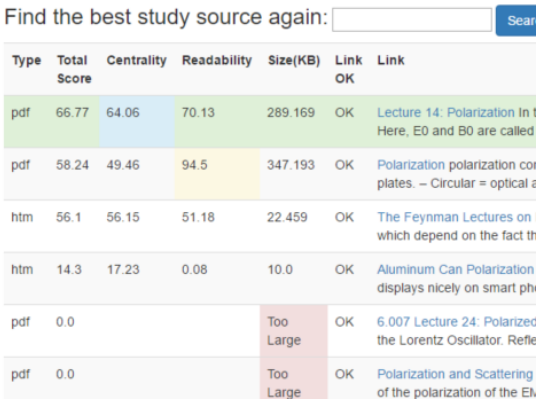

Figure 3. Displayed results with scores.

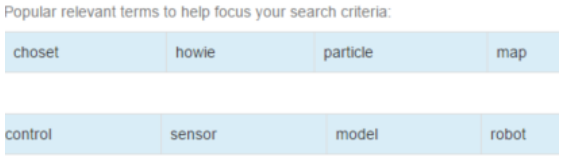

Figure 4. Notice all the key phrases are related to robotics navigation. "Howie Choset" is a professor who does research on the same subject. The application cannot distinguish names.

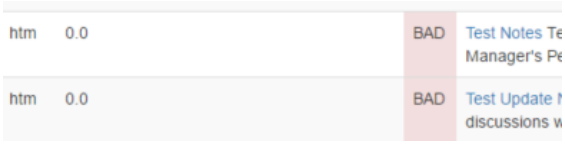

Figure 5. Notice all the key phrases are related to robotics navigation. Howie Choset is a professor who does research on the same subject. The application cannot distinguish names.

3. LIMITATIONS

The major bottleneck of this application is the pdf parser PDFMiner. It runs very slowly and does not always provide accurate results(sometimes words are not parsed correctly, or whitespaces aren't picked up). My application current runs on a free Heroku server and thus only has a single core and limited processing power. In addition, Heroku automatically exits the process if it takes more than 30 seconds. These constraints forced me to place major restrictions on pdf files. The restriction on the application is that only 3 pdf links will be analyzed in one go. This means, that even if the pdf file failed to be parse (too large, incompatible with the parser, etc) it counts against the limit. Unfortunately, there are not better native python alternatives to PDFMiner.

Aside from pdf files, some html files also are not guaranteed to be readable. A few causes may be from:

- Server prevents quick successive requests. My application queries the head before getting the page content. This process is too fast and not allowed by

some servers. However, due to time constraints, the application cannot wait.

- Some pages are ill formatted and don't conform to HTML standards. They may have improper headers and be unreadable or not fully readable.
- Links that redirects to another link are not accounted for

In such cases of failure, the system provides a minimal sort of diagnostic (see figure 5).

Furthermore, the application was not developed to handle large traffic. In its current form, concurrent searches performed by more than one person at a time will cause the Heroku site to time out.

4. PERFORMANCE AND SUMMARY

Due to the difficulty of quantifying the performance of the application, performance can only be visually observed by comparing with Google search (what the best result is in the end sometimes subjective to the viewer). In general, the search results of my application is much more lecture focused and filters out the video lectures that often populate the top of Google searches. The result two clusters work surprisingly well as it is relatively apparent the top results have some degree of relation to one another for most of test searches conducted.

The readability measure was somewhat unreliable. Sometimes the scores it provided were not all that convincing. In individual tests, results that scored highly sometimes were more difficult to read than those that scored low. This is especially the case where the lower scoring result has very few words and sentences but higher ratio of total words and syllables. Unfortunately, there were no other existing alternatives I could utilize.

5. FUTURE IMPROVEMENTS

Even when using existing optimized software packages, it is difficult to create an application that can run at satisfying speeds. The obvious solution is larger server power. However, a more prudent solution may involve more active processing from the client. However, due to the limits of JavaScript, most of the backend processes most likely cannot be dealt in the frontend. In that sense, perhaps a smartphone application might be more apt as it offers more capability at the client level.

Another point of improvement is in the parsing. The parsing currently is not reliable and sometimes a third of a search result might get thrown away from parser issues. A more robust parser would make the tool more reliable.

REFERENCES

1. Jillian R. Griffiths,Peter Brophy. Student Searching Behavior and the Web: Use of Academic Resources and Google.
2. Malik, Amara, & Mahmood, Khalid (2009). "Web search behavior of university students: a case study at University of the Punjab." Webology, 6(2), Article 70. Available at: http://www.webology.org/2009/v6n2/a70.html
3. J. Brophy and D. Bawden, 2005. "Is Google enough? Comparison of an Internet search engine with academic library resources," Aslib Proceedings: New Information Perspectives, volume 57, number 6, pp. 498–512.http://dx.doi.org/10.1108/00012530510634235
4. Bilal, D. (2000). Children's use of the Yahooligans! Web search engine: I. Cognitive, physical, and affective behaviors on fact-based search tasks. *Journal of the American Society for Information Science and Technology, 51, 646-665.*
5. T. Joachims and F. Radlinski, 2007. "Search engines that learn from implicit feedback," IEEE Computer, volume 40, number 8, pp. 34–40.http://dx.doi.org/10.1109/MC.2007.289 Jofish Kaye and Paul Dourish. 2014. Special issue on science fiction and ubiquitous computing. *Personal Ubiquitous Comput.* 18, 4 (April 2014), 765-766. http://dx.doi.org/10.1007/s00779-014-0773-4
6. Eyal Krikon, Oren Kurland, and Michael Bendersky. Utilizing inter-passage and inter-document similarities for reranking search results. ACM Trans. Inf. Syst., 29(1):3:1–3:28, December 2010.
7. Collins-Thompson, K. 2014. "Computational Assessment of Text Readability: A Survey of Current and Future Research". International Journal of Applied Linguistics. 165(2): 97– 135.
8. Collins-Thompson, K., Bennett, P.N., White, R.W., de la Chica, S., & Sontag, D. (2011). Personalizing web search results by reading level. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11). ACM, New York, NY, USA, 403–412.
9. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In Proceedings of the 7th International World Wide Web Conference, pages 107–117, 1998.
10. PageRank without hyperlinks: Structural re-ranking using links induced by language models
11. R. B. Yates and B. R. Neto. Modern Information Retrieval. ADDISON-WESLEY, New York, 1999.
12. Simplification of Flesch Reading Ease Formula.Farr, James N.; Jenkins, James J.; Paterson, Donald G. Journal of Applied Psychology, Vol 35(5), Oct 1951, 333-337.http://dx.doi.org/10.1037/h0062427