Summalyze: Organizing Information to Aid Learning

By: Neel Patel, Nithin Tumma, Edward Grafstein, James Ruben

Our fantastic demo video

Overview

We implemented Summalyze to naturally order links about a query to facilitate learning about the query topic. Google search results only return a set of URLs, but they're not ordered in a logical manner for learning. Summalyze improves the search experience by returning everything you need to learn about a topic - in order of complexity and also in the natural order of what you need to learn about first.

When you Google a topic like "biology", the order of results does not indicate the level of depth/the natural order of the links. We want to improve search for the cases where the user is looking for a more in depth treatment of the topic, not just a single article. With Summalyze, users can choose what level of depth that they want to explore, and articles are arranged according to their natural order (prerequisites for a particular article are given before it) within a given depth range.

How it works:

A user submits a query at www.summalyze.herokuapp.com and recieves a list of links ordered by depth (with respect to the query topic, from summary to specific) and in natural order of preresiquites. The user can also choose a range of depth by which to filter.

Check out our functioning app at www.summalyze.herokuapp.com. Search for one of our cached results: 'Harvard', 'Manifold', 'Biology', or 'Communism'. If you search for any other term, it may take a few minutes for the results to be generated since we're rate limited by Google. Thus, we recommend searching for one of our sample cached results for demonstration purposes.

Planning

We initially envisioned creating textbooks on demand for a particular topic. However, we quickly found out that it was not feasible to get the quantity of links needed to span the breadth of a topic as general as biology. So we focused on generating content that covered different depths of the same topic - starting at the Wikipedia article of the topic (a high level overview) to specific research level articles on components of the topic.

Please see our PDF titled “Annotated Spec” in the “report” folder. We annotated this PDF with our actual implementation.

It took longer than we expected to get the Wiki corpus, so we had to use a set of 10 articles for our test corpus for the first week. Towards the beginning of the second week, we settled on using the Brown corpus, which ships with the nltk toolkit in python and proved to be easier.

The Google search wrapper was much simpler than we expected so we were able to finish it within the first week. We weren’t able to overcome the rate limitation that Google places on its API usage, so we were forced to cache some results for testing. It turns out that it isn’t possible to use Google to crawl the web without paying for an enterprise level license, but it seems to work for our purposes and with low volume.

All of our other milestones were completed as we had initially planned.

Design and implementation

Implementing Summalyze was difficult because we had to design the majority of algorithms that we ended up implementing from scratch. We went through a few iterations of generating heuristic scores for the depth and order of each article.

The initial algorithm involved finding a list of keywords (based on tf-idf scores) and then finding sub-keywords for each of the initial keywords. Finally, we would return the set of resulting URLs ordered by keyword density. This implementation returned an ordered set of URLs but was not as fast or accurate as we needed it to be. The first problem with this implementation was that recursively finding keywords resulted in hundreds of Google/HTML queries, which made our algorithm very slow. Additionally, the set of resultant keywords didn’t have the context of the original query, meaning that the resultant links didn’t span the original query very well. Finally, our ordering method (based on the density of keywords) was very naive - we tried to rank based on how evenly all keywords were distributed, which didn’t account for certain keywords being inherently more important. Our main takeaways from this implementation were that the tf-idf method works for determining keywords, but we needed a non-recursive approach and a smarter way of ordering based on keyword distribution.

Our next implementation of the algorithm addressed three key issues: 1) speed of algorithm, 2) more representative spanning keywords, and 3) better ordering algorithm. To solve both problems 1 and 2, we stopped recursively calculating sub-keywords. Instead, we found the 10 keywords on a topic, and found search results for each combination of these 10 keywords. Not only did this decrease the number of HTML queries (by 90%), but it also resulted in keyword combinations that directly spanned a query. We also tried searching for logical groups of keywords based on a correlation matrix of keyword associations, but we found that this decreased the breadth of the search results, and limited the span of the keyword groups.

To solve problem 3 and improve the ordering algorithm, we first improved our method for determining article complexity (summarability). Rather than just looking at the distribution of keywords in an article, we compared this distribution to the distribution of keywords in the source Wikipedia article. By making this comparison relative, we accounted for the inherent variability in keyword importance. We also realized that summarability alone was not a good metric for ordering - we also needed to take into account the natural prerequisite ordering of keywords. We solved this by calculating an orderability score for each article: we first calculated the natural ordering of keywords, and then calculated how well each article represented this natural ordering. This orderability score could then be used to determine which articles were prerequisites to others.

The end result of these improvements were that our algorithm was reasonably fast and minimized the number of search queries needed, our ordering method took into account both the complexity and prerequisite ordering of an article, and that the set of resulting URLs directly spanned information on a given query. Heuristic tests on terms like ‘biology’, ‘manifold’, ‘communism’, and ‘Harvard’ revealed that the algorithm returns relevant and interesting information in a natural order.

Reflection

We were pleasantly surprised by the tf-idf method of finding keywords in the original Wiki article. This produced words that, on heuristic level, appeared to adequately explain a topic. We were also very happy with how our summarability and orderability algorithms performed - on a heuristic level, the sorted set of results logically progresses in terms of both complexity and prerequisites.

Getting the links from Google proved to be very difficult, as Google does not support more than 100 queries a day form its search API. We looked into other options (Bing, Yahoo, etc.), but there were no free search API’s available. We implemented our algorithm with Google’s free API, caching each new result to speed up the process the next time. Our algorithms would work much faster if we purchased a commercial API license, but we thought that it was unnecessary for proof of concept (it takes about 3 minutes to complete a new search and 5 secs for a cached result).

Our decision to combine the summarability and orderability scores while sorting the links worked out well when we examined our results on our test queries. This allowed us to keep both elements that we thought were key to learning (complexity and order) when we sorted the links.

Our worst decisions were our initial decisions to recursively find sub-keywords for each of the keyword articles - this method required 1000s of HTML queries every time we ran our program, which was very slow (due to Google's rate limiting) and also not representative of the initial query (since sub-keywords didn't necessarily span knowledge of the initial query). Since this method was very slow, it made iterating and improving the algorithm difficult.

If we had more time, we would optimize the process of getting links. Currently, we are getting likes from Google using their free API, but we are limited to a 100 queries per day. If we had time, we would write our own web crawler to index links to speed this up.

If we were to redo this project from scratch, we would consider writing some of the more computationally heavy portions of the algorithm in C to speed the algorithm up. Additionally, we would attempt to cache more results than we were able to do to refine the algorithm on a larger set of queries.

Initially, Neel and James wrote the majority of the Pre-processing code and generated the file that contained all of the tf-idf values. Nithin and Eddy spent most of the first two weeks working on implementing the summaribility and orderability algorithms. During the second week, Neel and James worked on getting the initial Django app running with basic processing and using the google wrapper to get the set of URLs. During the end of the second week and the beginning of the third week, Nithin and Eddy worked on combining the summaribility score and orderability score to develop a natural ordering of the links. They also implemented the ability to filter links by depth on the web app. Towards the end of the third week, James and Eddy worked on the report, video, and UI design, while Nithin and Neel continued to tinker with the backend to improve performance.

Advice for future students

Perhaps the most important lesson that we learned was that we were most productive as a group when we worked at the same time and in the same space. While this strategy definitely led to many long nights together, solving our problems as a collective unit proved to be most effective because whenever a bug, complication, or confusion came up we were able to address it together. This led to solutions that were created much more quickly because we attacked most problems with four minds instead of just one or two. Moreover, we would definitely suggest finding a topic that you actually are interested in learning about and finding a solution to (as we thankfully did) or else you will dread the work that needs to be done rather than enjoy it.

In addition, if you plan to work on a project that focuses on NLP, or specifically on text summarization, we advise spending a lot of time initially mapping out the algorithms that you plan to use. In our project, we had to create many of the algorithms from scratch. It is important to note that it is difficult to determine if your results are “right”, as they can only be tested in a heuristic sense. Therefore, you must put a lot of time into making sure that the logic of your algorithm is sound. Figuring out which algorithms to use early on will avoid a lot of the problems we faced when we realized some of the techniques we wanted to use were not going to work.