An Extensible System for Restricted-Domain Search Engine Summarization and Clustering

Abstract

Search engines have revolutionized the process of finding information online, but they still suffer from issues that can prevent users from finding the content they are searching for in a timely fashion. Even when searching within a specified domain, such as a website or set of documents, such issues can persist. One promising improvement over vanilla search is search result clustering. In this project, I develop an extensible system that adds summarization and clustering capabilities to a search engine for a restricted document set. These capabilities were first added to AAN’s existing search bar. The second part of the project involved creating a more generic search engine solution that has summarization and clustering capabilities and can be easily set up to index and search within the domain of one website through form input. Both systems were designed with a Yapsy-based plugin system for clustering, cluster summarization (to produce cluster descriptors), and cluster ranking, such that new algorithms for any of these purposes can easily be added. Performance for DBSCAN clustering, RAKE summarization, and maximum document score cluster-ranking was tested with a user survey in which three queries and both clustered and not clustered results were presented to the participant. While participants generally showed a preference for clustered results, it was not statistically significant and depended heavily on the query. There were also commonly identified issues with cluster summaries being repetitive, ungrammatical, or too specific. These results suggest further work should be done to improve the clustered search results output, perhaps through new plugins for these purposes or different processing and extraction of text from HTML web pages.

Introduction

Search engines have revolutionized the process of finding information. They are used every day by millions of users all around the world to answer questions on topics ranging from rocket science and nuclear physics to sports news and housing services (Agichtein et. al). However, usability studies have shown that--despite their significant progress--search engines still suffer
from issues that prevent many users from finding what they’re looking for in a timely manner. Users--especially less tech savvy users--often enter queries that are too general or short to precisely convey their intent (Ferragina et al.). For instance, a user may enter a query like “Italy” with the intent of gaining information about numerous distinct subtopics--where to go traveling in Italy, where Italy is located, cultural norms in Italy, or even just what Italy is. While many search engines attempt to guess user intent based on factors like user search history, clickthrough rates, trending topics, and more, it is a difficult task to do well consistently (Ferragina et al.; . Furthermore, user studies show that most users only attend to the first 10 returned results of a searched query (Ferragina et. al; Agichtein et. al). Particularly if there are a large number of results to sort through and users are not familiar with ways to optimize queries to find exactly what they’re looking for, navigating long results lists can be very time consuming. Examining full web pages after looking over the title and snippet in the search results becomes even more so.

An interesting subcase of this general internet-level search problem is the search of individual websites or domains, such as “nhsofnewhaven.org.” While sites are generally designed with the intent of being easy to navigate, this can be a difficult task to realize. Small or underfunded organizations--schools, local governments, and non-profits come to mind--are notorious for having websites that are poorly designed and difficult to navigate (Barrett & Greene; Castro). As such, even if a user knows what site the information they seek is located in, it can be difficult to find the precise page or document. Site-specific search engines can be used to facilitate this navigation process, and often are. Major search engines like Google offer site-specific search options (e.g., a site:site_name filter), and other frameworks exist to build custom ones as well. However, especially when used in websites with large domains, similar problems arise in this restricted-domain search problem as do in general search.

One potential improvement to search is result clustering, which attempts to group together related search results. Clustering has been shown in some user studies to decrease the time spent to find a desired web page as users can quickly narrow results to only examine the pages returned in the cluster(s) that are relevant to their query’s actual intent (Cobos et al.; Ferragina et al.). Note that this sort of clustering requires an additional step past the grouping to generate intelligible descriptions of each cluster to facilitate the user’s search process. The original aim of this project was to create a system that can cluster web pages and other documents in a limited search space (e.g., one set of documents or one website) as compared to the entirety of the internet. This goal was eventually narrowed to create a system that focused on the single website or domain case.
Previous Work

Clustering has been explored as an element of search engine applications in prior works. Search engines like Carrot2 and SnakeT currently exist that provide search clustering capabilities (Osiński & Weiss; Ferragina et al.). However, most of the literature and live search engines focus their clustering capabilities on the more general problem of searching the entire web. While the two problems of searching the web and a single website are analogous in many ways, the smaller size and (likely) more restrained topic area make the single site search problem distinct. For instance, it can be prohibitively computationally intensive to download all web pages on the internet and compute their similarities. As such, many clustering search engines are meta search engines that use the much more limited information from the results of a search from, say, Google, to cluster results (Ferragina et al.). However, it is a computationally feasible task to download and compute similarities for all pages in (most) single websites, which provides richer information than just search snippets. Additionally, computing cluster descriptors for groups of web pages that are already all somewhat similar simply by virtue of being part of the same website may require some modified algorithms to ensure cluster labels are sufficiently distinct to be useful.

General Approach

This project aims to add summarization and clustering capabilities to a search engine for a restricted document set. It will first involve adding these capabilities to the AAN’s existing search bar as a proof-of-concept. The second part of the project will involve creating a more generic search engine solution that has summarization and clustering capabilities that can be quickly set up to search within the domain of one website. Another focus of this project was to make the domain-specified search engine easily extensible such that it can be modified to make use of alternate clustering and summarization algorithms past the ones implemented in the scope of this project, particularly as these are currently popular and fast-progressing research areas.

AAN Search Engine

Functionality:
The preexisting AAN website has a search bar on the front page which can be used to find papers and/or other resources that are relevant to the entered search terms. I implemented a new “Search and Cluster Resources” feature. When clicked, the search engine retrieves the top relevant resources for the query and then clusters them. The clusters of resources, ranked by
relevance, are displayed to the user as the final result along with a brief summary of each cluster, currently in the form of a list of keyphrases.

Figure 1.1. Modified search bar on homepage with “Search & Cluster Resources” option.

Figure 1.2. Snapshot of part of the returned clustered search results for the query “neural networks.”

Architecture:
The logic for the AAN website is coded in PHP with CodeIgniter. However, there is relatively little support in PHP libraries for clustering, summarization, or other such machine learning
methods. I therefore decided to use Python instead for the logic of the clustering, summarization, and cluster ranking methods.

I created a REST interface to facilitate this information sharing. The PHP code exposes a SearchAPI (see the SearchAPI controller) which returns a JSON of the relevant resources’ ids mapped to their SOLR search scores. This uses the same algorithm as the Resources search functionality, to keep results more consistent aside from the clustering process. The Python code exposes a ClusterAPI which retrieves the relevant resources for a query from the PHP-backed SearchAPI. The ClusterAPI then clusters, summarizes, and ranks the clusters for that set of resources, and returns a JSON in which each cluster is mapped to a list of associated resource ids as well as its summary string.

The PHP code which helps display the results of a clustered search is the Cluster controller. It queries the Python-backed ClusterAPI to determine the cluster information and then displays it in a table, similar to the results of a regular Resource search, but including cluster assignments and cluster summaries. Note that the clustering, summarization, and cluster ranking algorithms are specified in the query from the Cluster controller to the Python-backed ClusterAPI, which currently defaults to DBSCAN, RAKE, and max_score, but can easily be changed. These algorithms will be discussed in more depth in a following section of this report.

![Diagram](image)

*Figure 1.3. Architecture for the “Search & Cluster Resources” functionality in AAN.*
Domain-Specified Search Engine

Functionality:
This search engine has two major functions. It can crawl and index the pages on a given domain, readying them for search, once given a domain and associated web url (e.g., “makehaven.org” and “https://makehaven.org”) along with some information including where to store scraped web pages and the backing SQL and SOLR instances that will help store indexed data.

Once the domain is indexed, it can be searched. The engine allows users to choose which of the indexed domains to search as well as which clustering, summarization, and cluster ranking algorithms to use. Once this information and a query is provided, the engine displays clustered search results. Each cluster has an associated summary and a list of associated search results, which are displayed as a title page with a link and snippets of relevant text with search terms highlighted. There is also the ability to conduct vanilla, non-clustered searches which was used for testing purposes.

Figure 2.1. Form to enter required information to begin indexing domain.  
Figure 2.2. Form to enter a search query.
Architectur: This application was created using the Django framework. To index a new website, a user simply specifies a website domain and some storage information. The application then scrapes the website’s html pages using Scrapy to crawl and download them. It saves the file locally (for quicker access by further processing) and also posts the HTML contents, titles, and the extracted plaintext (approximated to be the text enclosed in title, h1-h6, or p tags) to the specified SOLR instance. Note that the SOLR instance should be preconfigured with the cluster_configs schema that was created for this application (it is included in the code in Appendix). The application then computes the similarity scores using TF-IDF and cosine similarity for every pair of pages’ titles, urls, HTML contents, and extracted plaintext to the MySQL table specified. The application also enables semantic similarity searches for the extracted plaintext by posting the results to SOLR as an additional document field, “blurred_plaintext,” if the formatting is acceptable (see Berryman’s article in the References section).

To search an indexed domain, the application asks the user to select the web domain to search (out of a list of the domains indexed), the search query, and optionally one of a selection of the implemented clustering, summarization, and ranking algorithms (the defaults are DBSCAN, RAKE, and max_score). The application then conducts a SOLR search using the given query. It retrieves the precomputed cosine similarity scores in MySQL for the returned document pairs, and uses those to compute and populate a distance matrix. It passes this matrix into the specified clustering algorithm that clusters the documents. These clusters are then passed into a cluster...
summarizing function that returns descriptors for each cluster. These clusters and their descriptors are then ranked, and then returned and displayed for the user.

![Figure 2.4](image1)  
**Figure 2.4. Architecture for the indexing process in the domain-specified search engine.**

![Figure 2.5](image2)  
**Figure 2.5. Architecture for the clustered search process in the domain-specified search engine.**

**Clustering, Summarization, and Cluster Ranking**

**Plugin System:**
One of the major goals of this project was to create a system that could easily be used to add clustering capabilities to websites and thus fit a wide range of needs. As such, making the codebase modular and extensible was one priority of the project. A significant way this was accomplished was by using Yapsy to create a plugin system for clustering, cluster summarization, and cluster ranking methods.

These three types (or “categories” in Yapsy) of plugins are saved in subdirectories of the project folder. They inherit from the IClusterPlugin, ISummaryPlugin and IRankPlugin classes I created, which themselves are derived from Yapsy’s IPlugin class. To add a new method of clustering, cluster summarization, or cluster ranking, one must simply create a new class that inherits the corresponding parent and implements the relevant function(s) (e.g., cluster_results(), summarize_clusters(), rank_clusters()) as well as create a “.yapsy-plugin” extension of the same name with a brief description of the plugin.

Note that this structure is replicated both in my implementation for AAN and my implementation for the domain-constrained search engine, and the plugins by the same name have the same ultimate functionality and algorithms, but take input and output in slightly different formats due to the differing nature of these two systems.

**Clustering Methods:**
After reading the literature on search engine clustering, I became interested in query-specific clustering methods. I thus decided to apply clustering algorithms to the results of a regular search for a query, instead of doing clustering on the entire corpus before any queries were run.

For each query, all the relevant resource pairs’ cosine similarity values are retrieved, converted to distance values, weighted and added together (normalized to 1), and the sum is put into a distance matrix. This distance matrix is then used as the input to a clustering algorithm. I created plugins that—generally using scipy helper functions—implement DBSCAN (Density-Based Spatial Clustering of Applications with Noise), HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise), Affinity Propagation, single linkage clustering, ward linkage clustering, median linkage clustering, centroid linkage clustering, weighted linkage clustering, and average linkage clustering.

**Cluster Summarization Methods:**
Compared to general single document summarization methods, I found relatively few multi-document summarization methods that have existing Python libraries or were not prohibitively time-consuming to code from scratch based on the constraints and other requirements of this project. While I considered simply appending all the documents’ text in a cluster to create one aggregated document and then summarizing it, this presented issues in
intensive decision with to I begin three these, many more algorithms also more cluster were much computationally complex ranking, considered its or While some documents respectively score median its 3) the documents from these, 2) highest sum scores the for scoring highest of the document its from cluster document output, plugins, three of I Each and max_score Cluster for large the cases tradeoff documents most. use documents very by high the large performance incurred few extremely costs.

Cluster Ranking Methods:
I implemented three cluster ranking plugins: max_score, sum_score, and median_score. Each of these use document scores for the resources in each cluster to output a list of clusters ordered from highest to lowest score for 1) its highest-scoring document, 2) the sum of the scores of all its documents, or 3) the median score of its documents respectively. While I considered some more complex cluster ranking algorithms, many were also computationally much more intensive, so I decided to begin with these three.
Evaluation Methodology

Clustering and summarization are particularly difficult to evaluate as there are often no “ground truth” results that the algorithm results can be compared to in order to authoritatively identify them as correct or not. As such, many evaluations of these algorithms especially with relation to search engines rely on or at least incorporate user studies (Ferragina et al.; Cobos et al.; Leung et al.). I followed this trend.

Due to time and resource constraints, I was not able to conduct an exhaustive user survey comparing all the plugin options. Additionally, a major goal of this project was to create an easily extensible framework to add clustering and summarization capabilities to a search system rather than focus on optimizing the capabilities of a particular algorithm. Regardless, I did want to gain a general sense of the utility and areas of improvement for the application. I therefore chose one set of a clustering, summarization, and ranking algorithms that seemed to give relatively good results compared to the others on some exploratory queries I conducted. I chose DBSCAN, RAKE, and max_score. I then conducted a survey on user experiences for my domain-constrained search engine with the clustered and summarized results versus the non-clustered results. I conducted the search on the domain-constrained search engine instead of AAN because many of my subjects were not familiar with Computer Science or artificial intelligence, and thus may have had difficulty assessing the results.

Participants were given three queries with the domain the search was performed upon, as well as the general goal for the search. They were provided with both the clustered and unclustered search results, and asked to answer some questions evaluating whether the clustering and summarization were helpful and how so. A link to the survey and sample results is provided in the Appendix.

Results

The full survey and sample results are linked to in the appendix, but for context, these were the 3 queries.

Query 1: Searched makehaven.org for "workshop." We want to find out what workshops they have.
Query 2: Searched nhsofnewhaven.org (Neighborhood Housing Services of New Haven) for "help on mortgage." We want to find information on help paying for our mortgage.
Query 3: Searched yalecode4good.wordpress.com for "how to help." We want to learn about how to help out with our tech skills.

| Did you think the clustered or not clustered results were more useful? |
|---------------------------------|-----------------------------|
|                                | Clustered | Not Clustered |
| Query 1                        | 8          | 15            |
| Query 2                        | 16         | 7             |
| Query 3                        | 13         | 10            |
| Total                          | 37         | 32            |

*Figure 3.1. Number of participant responses to the question about clustered versus not clustered results usefulness, broken down by query and response.*

| Did you think the cluster keywords/summaries were useful? |
|---------------------------------|----------------|
|                                | Yes | No | Other |
| Query 1                        | 9   | 12 | 2     |
| Query 2                        | 16  | 5  | 2     |
| Query 3                        | 10  | 11 | 2     |
| Total                          | 35  | 28 | 6     |

*Figure 3.2. Number of participant responses to the question about clustered summary usefulness, broken down by query and response.*

**Discussion**

In general, users preferred clusters and found cluster summaries useful, but this was highly dependent on the search conducted and not statistically significant. For instance, there was a noted preference for clusters and cluster summaries on Query 2, but the opposite (but less strong in valence) preference with Query 1. Query 3 was more evenly split. One hypothesis is that certain queries and websites are more suited for clustering than others—for instance, if a website has all information on a topic confined to a single page, it is unlikely clustering will be very helpful. Some users also generally seemed to like clustering more than others, which suggests that making it an option on search engines but not the only option might improve the
search experience. Some users also commented in the survey that the clusters’ formatting was a little cluttered, so making changes to that may also increase the preference for clustered results. However, further testing is needed to explore these possible conclusions.

One significant issue that many of the freeform comments by participants highlighted was that the cluster keyphrases were sometimes repetitive and nonsensical (one example is “first blog post blog blog first blog post”) or overly specific and not representative of the cluster (sometimes, isolated names were picked out as a keyphrase). These phrases often seem to have been taken from sections of a web page where there are nested section headings or a heading followed by the related paragraph, and the first header phrase is repeated. Trying alternate summarization algorithms, conducting more post-processing on the returned keyphrases in RAKE, or retrieving the extracted plaintext from HTML pages are some possible ways this can be improved in the future.

This issue was even more pronounced in the AAN cluster summaries. The links to the text for AAN resources were sometimes non-existent, broken files, or had formatting errors which made it difficult or not possible to parse, and which may have caused some clusters to have particularly nonsensical and repetitive keyphrases.

**Conclusion & Future Work**

This search engine with query-specific clustering, summarization, and cluster ranking plugin support can enable search of a domain with custom clustering features by just filling out a form. For real-world applications, the demo can be adapted to be included as a website search engine after being used to index the site’s pages. While the pre-processing and indexing portion can take some time, that only needs to be done fully once (updates are discussed in next section), and makes the searching process—which in most cases will be done many more times than the indexing—faster. Especially for organizations that do not have much technical expertise or are severely time crunched--such as smaller nonprofits or public agencies--a easy-to-deploy tool such as this could be a useful to augment the navigation and search experience of website visitors.

However, the current clustering algorithms seem to provide somewhat mixed results based on the user survey that was conducted. As such, the system could benefit from new plugins for clustering, summarization, and ranking, further tweaking of parameters, or additional pre/post processing of web pages and resources (in the case of AAN particularly). The application currently precomputes and stores the cosine similarity values for document pairs in a table such
that they can be easily looked up to decrease computation time during the actual search process. However, being constrained to using those values made it more difficult/impossible to use some clustering algorithms that use other features about documents (such as k-means).

Experimenting with ways of saving the TF-IDF vector itself or other document representations, and making new plugin classes that can take in that information and efficiently process it, could expand the clustering options and power available to the application. The application also currently only parses HTML document. Expanding this to other content types like pdf’s, docx files, and more would provide a richer search experience. Furthermore, adding customizations to parse titles and such from all these document types, and utilizing the document structure to provide information on, for instance, the importance of certain terms, could possibly improve the search and clustering capabilities as well.

Appendix

The code for part 1 of this project, the additions to AAN, is available on this Github Repository (note that it is private to users outside the AAN group): https://github.com/emdat/aan.

The code for part 2 of this project, the domain-specified search engine, is available on this public Github Repository: https://github.com/emdat/search-cluster-summarize.

The evaluation survey with links to the sample results can be accessed here:
https://docs.google.com/forms/d/e/1FAIpQLScdp3aWxMiAzzQ22-SoCQpwqeauwGfzSxhWo0oi-0AAMOLcwA/viewform?usp=sf_link
References


