Recommendation algorithm on Academic research
Vincent 3035095779
Supervisor: Dr. Yiu
1. Introduction

Since the invention of the World Wide Web, the amount of information had entered an exponential growth. Since information on the Internet has a wide coverage on different topics, doing good research on the Internet is now an essential part in writing any essays and articles.

Several difficulties arises when writing essays. Writers often need to spend a lot of time on searching and filtering for relevant resources on their research topic. Google search and Google Scholar are very popular tools for search for academic research resources. They can significantly reduce searching time once the user had found the right keyword to search for. Despite that, time are still spent on scanning the articles and filtering useful texts. Second, while technological advancements had allow us to write and do research with the same device, the process of researching and writing had been largely separated. Integrating the research process into the writing had become a time consuming task. Tools like Zotero and Endnote exist to help manage the texts which users find and store from websites. They also provide auto-generated citations. However, these tools does not integrate the process of searching and writing as searching is still largely separate from the writing work flow. Developing a software with algorithms that can automatically understands your current writing and provide adaquate resources on the Internet can greatly reduce the time required on researching and integrating the research with the writing.

2. Background

There are several popular tools for academic research, including Google Scholar, Zotero and EndNote. Table 1 shows a summary of advantages and disadvantages of each of the tools. Google scholar is one of the most popular searching tools. It can rank research papers so that users’ searching time is greatly reduced. However, users still need to scan the whole article to determine if it is useful and locate the useful texts. Moreover, the writing and research process is separated. The general approach by most people is to use Google Scholar to search for relevant resources, open new tab for each web page, and then write their essays. This approach has a big disadvantage: writers have to spent time on revising their materials again during writing. Even if Google Scholar is combined with Zotero and EndNote, the above two problems still cannot be solved.

The deliverable of this project will be an integration of functions provided by the above mentioned tools, together with functions that recommend relevant texts and integrate writing
and searching process. Table 2 shows a comparison of the deliverable and other research tools. The core advantage of the deliverable is that it reduces essay writing time significantly by recommending most relevant resources and texts and this function is not in any of the other research tools.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Scholar</td>
<td>• Search for academic research resources quickly</td>
<td>• Lack a tool to highlight and store the relevant research resources found</td>
</tr>
<tr>
<td></td>
<td>• Search results are ranked by number of citations, helping users get the most relevant resources</td>
<td>• Lack a suggestion of text. Users need to scan the whole article to get useful text and see whether the result is appropriate</td>
</tr>
<tr>
<td>Zotero</td>
<td>• Automatically senses the content in a web browser and allows users to store the content</td>
<td>• Lack a search function to search for relevant resources</td>
</tr>
<tr>
<td>EndNote</td>
<td>• Manage PDF academic research resources</td>
<td>• Lack a method to rank resources</td>
</tr>
<tr>
<td></td>
<td>• Search research resources in PDF form in online database</td>
<td>• Lack searching of internet materials</td>
</tr>
<tr>
<td></td>
<td>• Users can look at comment made on PDF online</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Summary of current academic research tools

<table>
<thead>
<tr>
<th></th>
<th>Google/Google Scholar</th>
<th>Zotero</th>
<th>EndNote</th>
<th>Our deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for relevant website</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Search for research paper</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Search results are ranked</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-generated citations</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allow users to store useful results</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
This project will be built on top of a developing web application named Writio which provides support of highlighting and storing websites’ texts. At this stage, Writio is capable of allowing users to highlight any texts on a website. The highlighted texts are stored and can later be referred when users write essays. With the highlight data from users, it is possible to allow the searching of database to find relevant resources.

### 3. Objectives and Scope

This project is going to achieve five things.

1. Structuring a database to store users’ highlights for searching.
2. Developing an algorithm or integrating existing algorithms to search for relevant recommendations on research resources from keywords.
3. Ranking existing cited research resources by an algorithm.
4. Developing an algorithm to understand users’ essay and recommend resources accordingly automatically.
5. Generating citations in different styles.

These algorithms and functions are mainly developed for use in a web application named Writio. The integration of the algorithms and the user-interface of the extension depends on the development progress of Writio (not included in this project).

### 4. Project Methodology

#### 4.1. Database Structure

**4.1.1. Current GistNote Database**

To build a full text search engine, a different database structure is necessary. The current GistNote database is a nosql database (Firebase from Google) with a hierarchy: users have folders, and folders have their website and website contains its highlights. It is good for real time usage by users (i.e. categorized by users’ ID). (See figure 1 for a class diagram representation of the database structure). However,

<table>
<thead>
<tr>
<th>Recommend relevant text</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommend resources when writing</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2: Comparison of project deliverable and other research tools
this structure has limitation for full text search on users’ highlights:

1. Websites with same URL are duplicated in different folders.
2. Highlights are sparse in the whole database. Every highlights should be grouped by same websites.
3. Each word in highlights is not indexed for optimal search performance.

These dramatically reduce the search performance. To search for all highlights in a specific websites, or to search for all highlights containing a target word, the whole database is needed to be scanned. This takes 1 minute for 1GB of data (retrieved on 3\textsuperscript{rd} March, 2017). This is too long for a search and hence a different server with a different database structure is deemed to be used for improved efficiency.

---

![Figure 1 Diagram of GistNote’s Firebase Database Structure](image-url)
4.1.2. Database Structure for search engine (Full-text search not supported)

The schema for the search database can be illustrated by the E-R diagram in figure 2. In this schema, same websites’ information is merged into one table and highlights are grouped under the same websites such that it is efficient to find all highlights related to a specific website. Users and folders information are not stored because they are unrelated to the full text search. This is the version where there is no indexing of words. Therefore, full text search is not yet supported. See next point for the database with indexing.

![E-R diagram of search database (without indexing)](image)

Figure 2 E-R diagram of search database (without indexing)

4.2 Indexing for full text search

4.2.1 Schema for full text search

In order to perform efficient full text search, every words have to be indexed. In this project, the focus is on English word search. Here, indexing a word means creating an inverted table for the word. The indexed words are hashed as keys for more efficient search. Inside the inverted table there contains information about which websites and highlights contain the word, and the corresponding reference for linking to the related components. Figure 3 shows the E-R diagram of the database putting all together.
To obtain more precise search results, words in different fields are tokenized and indexed. The fields are highlighted text, websites’ content and title. As a result, when searching a specific word, the websites containing the word in its title, content, or highlights will be returned.

4.2.1. Improving indexing

4.2.2.1. Stop words Filtering
When doing full text search, some stopwords should be filtered because they are relatively unimportant. The list of stopwords can be found in Appendix 1.

4.2.2.2 Stemming before indexing
In English, a single word can have different states. For example, a verb (e.g. explore) have present tense(explor), past tense(explored) and present continuous tense(exploring). By stemming these states can be stemmed into common word (explor) so that the searching can become more general. In this project Porter Stemming Algorithm is used as the stemming algorithm. The parts stemmed are
suitable to English in this algorithm. This algorithm is very common when stemming words. The algorithm can be found in Appendix 2.

4.3. Tools

4.3.1. Apache Solr – server for indexing and searching
In this project, Apache Solr is used as the tool that accomplishes the above criteria. Solr is standalone enterprise search server. It supports indexing, searching with a REST-like API. At the beginning of the project, Apache Lucene was considered. Apache Lucene is an indexing tools so that the above features can be achieved. However, this requires some implementation. As a matter of fact, Apache Solr is a server implementing Apache Lucene. It is under Apache 2.0 license and hence is free to use. This matches GistNote’s commercial requirement. Moreover, it is well developed and hence more effort can be put on other function instead of setting up the server manually. The server can be accessed via http://fyp16028s1:8080/solr.

4.3.2. Apache Nutch – Website crawling tool
Since contents in websites are also necessary in the server, Apache Nutch is required for retrieving websites’ inner content. Apache Nutch can parse website provided in a seed of URLs. In this project, all the URLs highlighted are put into a seed list for Nutch crawler. Before retrieving contents, useless and inaccessible URLs should be filtered. Originally, there are 102806 URLs in GistNote’s database. After filtering out some useless URLs including dead link and some non-websites (e.g. URLs starts with chrome://), 53355 websites were stored. Another reason for choosing Apache Nutch is that it can integrate with Apache Solr easily. The parsed content can be upload to Solr server without difficulty. The time for crawling and indexing is roughly 4 hours 12 minutes (as for data in 7th March, 2017).

4.4. Data management

4.4.1. Data Import workflow

1. Parse GistNote’s Firebase data
GistNote’s data needed to be processed before indexing to Solr. All URLs with related highlights are needed. In addition, the URLs retrieved is used for website crawling by Nutch.

On 7th March, 2017, the Firebase data has size around 1GB (1056188KB). The memory of a normal machine may not be enough to parse the whole JSON data and store it in memory. Therefore, a parser that parses the JSON data line by line was developed. It is written up by JAVA and uses Jackson API. This takes 30 seconds to
parse the 1GB data. We believe that this parser will still be efficient enough for GistNote because the increase in data is around 100MB per month.

2. Website parsing by Nutch and indexed to Solr
The URLs retrieved from Firebase data can be injected to Nutch for crawling. Filtering on URLs is performed. The data containing contents and titles of websites is then indexed to Solr. This process takes 4 hours for 1GB data.

3. Import from GistNote’s Firebase to Solr
GistNote’s Firebase data are in JSON format while Solr accepts JSON input. However, GistNote’s data needed to be processed before indexing to Solr. The data is reprocessed in to schema specified in figure 2 before indexing using the parser developed. This process takes 30 seconds. With above three steps, around 4 hours was used and the total size of data stored is 1.11GB in Solr (with 1GB Firebase data). A more comprehensive analysis on the performance and data storage will be discussed in next topic. The whole process is illustrated in figure 4.
4.4.2. Incremental import of GistNote’s firebase data
Incremental update of data from Firebase to Solr has not yet been supported in current project. This is because the efficiency currently is still satisfactory. It is estimated that the data size of GistNote whole database data will not exceed 10GB with a current static 10% increase in size. Moreover, websites must be fetched and parsed for the search engine within a certain time interval. The time of fetching and parsing of website is much longer than the parsing of Firebase data (4 hours vs. 30seconds). Hence, it is concluded that incremental import of data is not necessary.

4.5. Querying
This section focuses on the query of specific search keywords in Solr database.

4.5.1. Stopwords filtering
Same as indexing, stopwords filtering is needed. Look at section 4.2.2.1 for reference

4.5.2. Stemming
Same as indexing, stemming is needed. Look at section 4.2.2.2 for reference

4.6. Ranking
Ranking is based on term frequency – inverse document frequency (TF-IDF) principles. In this project, BM25 algorithm is used for ranking in TF-IDF. In original TF-IDF principle, TF is just the frequency of a word, and IDF is calculated by:

\[
\log \left( \frac{\text{numDocs}}{\text{docFreq} + 1} \right) + 1
\]

In classic TF-IDF principle, length of document is also considered. If the length is small, there should be less probability of appearance of a word. As a result, the length of document is transformed to field norm:

\[
\frac{1}{\sqrt{\text{length}}}
\]

The overall formula:

\[
\log\left(\frac{\text{numDocs}}{(\text{docFreq} + 1)}\right) \times \sqrt{\text{tf}} \times \left(\frac{1}{\sqrt{\text{length}}}\right)
\]

For BM25 algorithm, the overall formula (take k=1.2, b=0.75)

\[
\text{IDF} \times \frac{(k + 1) \times \text{tf}}{(k \times (1.0 - b + b \times (\text{length/avgDocLength}) + \text{tf})}
\]

The basic principle of this formula is that term frequency, inverse document frequency and document’s length should have saturation of importance when they grow larger and larger.

In this project, BM25 is chosen because it is more intuitive and widely used.

4.6.1 Importance of different fields
In this project, there are three fields to be searched i.e. title, website content and highlights. There should be a score multiplier for these three fields.

4.7 Topic extraction
This section is completed by Erik Chan. Look at Erik’s report for detailed information.

4.8 Automatic citation generation
The critical part of automatic generation of citations and bibliography is to obtain the formatting of the target citation style. A cloud-based platform that store different citation styles is preferred. Based on this criteria, Citation Style Language (CSL) is chosen as the platform for the source of citation style. CSL is a open XML-based language to describe the formatting of citations and bibliography. It has many advantages. Firstly, It maintains over 8000 citation styles in the GitHub repository so that addition and modification of the formatting are convenient. The most popular citation style can be obtained easily and this already fits the scope of the project. Secondly, CSL is currently used by many other research tools, in particular, Zotero. Zotero even developed its own repository to manage the CSL file. It is seen that CSL is reliable to be used.

There already exists open-source precessors that implement CSL. Since a web-based text editor is to be developed in GistNote, a javascript version (citeproc-js) is chosen. It is already well-developed with powerful features like dynamic changing of citation styles. It reduces the effort in the development of citation generation feature in this project. The basic logic of generating citations and bibliographies is finished. The engine can generate citations with appropriate inputs, which are specified in the CSL files.

This feature has already been integrated into GistNote’s word editor. It currently supports 3 citation styles (APA, MLA, Chicago).

5. Results

5.1. Data storage efficiency
As on 7th March, 2017, the GistNote’s Firebase data has size 1GB. The data stored on Solr was 1.11GB. Intuitively with whole websites’ content stored, the data size should be much larger than the original data. With an 11% increase in data size, it is concluded the data storage for search engine is efficient.

5.1.1 Reason for efficient data storage
First, folders and users information is not stored. This helps to avoid useless data. Second, although there are 102800 URLs in original GistNote data, after filtering, around 53000 websites’ information are stored in Solr. This cuts around half of storage. Finally, the Solr indexing avoid redundant words because all words are indexed into corresponding table. Same word appeared in different highlights will not be stored again.

5.2 Importance of different fields

In correspondence to section 4.6.1, we choose the ranking formula to be

\[
\text{rank} = \text{BM25(title)} \times 1 + \text{BM25(highlights)} \times 2 + \text{BM25(content)} \times 2
\]

The use of BM25 can be referred to section 4.6. Here highlights and content are as important to each other and title is less important. We put score of highlights higher because the highlights are the most valuable data in GistNotes database. However the highlights score should not exceed content too much to remove noise or unrelated highlights’ influence on results. Contents in websites are still important to determine whether a website matches the search keyword or not.

5.3 Search page

A search page is developed at

http://i.cs.hku.hk/fyp/2016/fyp16028/FYP_searchPage/search.html

Search results pop up with keywords. There are important text (users highlights) under each website.

Clustering of search result is supported:
With keyword extraction algorithm, the keyword obtained from users’ highlights can be passed to the server for searching automatically. Hence, user can get search results without manually typing keywords.

7. Conclusion and Future Works

In conclusion, the search engine has been built to accomplish the objectives. From a search-unfriendly GistNote database structure to Solr for full text search, the workflow and effort for this transfer is designed to be efficient currently. This search function will certainly be useful to GistNote and its users who want to search for relevant resources for there research.

For future works, a recommendation engine is planning to be developed so that we can, by some machine learning algorithm, catch users’ research topic based on their highlights data and return approximate websites in the database.

8. Appendices

I.

a
about
above
after
again
against
ourselves
out
over
own
same
shan’t
all  she
am  she'd
an  she'll
and  she's
any  should
are  shouldn't
aren't  so
as  some
at  such
be  than
because  that
been  that's
before  the
being  their
below  theirs
between  them
both  themselves
but  then
by  there
can't  there's
cannot  these
could  they
couldn't  they'd
did  they'll
didn't  they're
do  they've
does  this
doesn't  those
doing  through
don't  to
down  too
during  under
each  until
few  up
for  very
from  was
further  wasn't
had  we
mustn’t
my
myself
no
nor
not
of
off
on
once
only
or
other
ought
our
ours

II

http://snowball.tartarus.org/algorithms/english/stemmer.html

Reference

References
https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf
(2) GitHub [Internet]. San Francisco, California, U.S.: GitHub Inc., 2016 [retrieved 25 Oct 2016]. Available from:
https://github.com/search?o=desc&q=keywords+extraction&ref=searchresults&s=stars&type=Repositories&utf8=%E2%9C%93
(3) CitationStyles.org [Internet]. CitationStyles.org, 2016 [retrieved 30 Oct 2016]. Available from: http://citationstyles.org/
(4) GitHub [Internet]. San Francisco, California, U.S.: GitHub Inc., 2016 [retrieved 30 Oct 2016]. Available from:
https://github.com/juris-m/citeproc-js
(5) SnowBall http://snowball.tartarus.org/algorithms/english/stemmer.html
(6) BM25
http://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-releva

(7) APACHE Solr guide
https://cwiki.apache.org/confluence/display/solr/Apache+Solr+Reference+Guide