Development of a Command Line Interface for the Analysis of Result Sets from Automated Queries to Literature Databases

Tobias J. BRIX\textsuperscript{a,1}, Malte BERENTZEN\textsuperscript{b}, Ludger BECKER\textsuperscript{b}, Michael STORCK\textsuperscript{a} and Julian VARGHESE\textsuperscript{a}

\textsuperscript{a}Institute of Medical Informatics, University of Münster, Germany
\textsuperscript{b}Institute of Computer Science, University of Münster, Germany

ORCiD ID: Tobias J. Brix https://orcid.org/0000-0002-1557-8097

Abstract. The first step of a systematic review is the identification of publications related to a research question in different literature databases. The quality of the final review is mainly influenced by finding the best search query resulting in high precision and recall. Usually, this process is iterative and requires refining the initial query and comparing the different result sets. Furthermore, result sets of different literature databases must be compared as well. Objective of this work is to develop a command line interface, which supports the automated comparison of result sets of publications from literature databases. The tool should incorporate existing application programming interfaces of literature database and should be integrable into more complex analysis scripts. We present a command line interface written in Python and available as open-source application at https://imigitlab.uni-muenster.de/published/literature-cli under MIT license. The tool calculates the intersection and differences of the result sets of multiple queries on a single literature database or of the same query on different databases. These results and their configurable metadata can be exported as CSV-files or in Research Information System format for post-processing or as starting point for a systematic review. Due to the support of inline parameters, the tool can be integrated into existing analysis scripts. Currently, the literature databases PubMed and DBLP are supported, but the tool can easily be extended to support any literature database providing a web-based application programming interface.

Keywords. Systematic Review, Literature Database, PubMed, DBLP

1. Introduction

Systematic reviews are important for the scientific community to provide an overview of the current state-of-the-art of a given topic or an entire research field. Guidelines for performing systematic reviews have been proposed to ensure reproducibility. At the time of writing, most journals request the authors to follow the PRISMA guidelines [1]. As first step of this guideline, a search string must be defined, which is used to query scientific articles from multiple literature databases. During the development of the search string, search terms are often exchanged, e.g., by synonyms, or different search

\textsuperscript{1} Corresponding Author: Tobias J. Brix, Institute of Medical Informatics, University of Münster, Albert-Schweitzer-Campus 1, Building A11, 48149 Münster, Germany; E-mail: tobias.brix@uni-muenster.de
terms are connected via logical operators like “and” or “or”. After each iteration with different numbers of results, the main task is to identify the newly found articles and articles, which were missed by the changed query. Mathematically speaking, having two article result sets from different queries, A and B, the set difference \( A \setminus B \) and \( B \setminus A \) must be calculated and analyzed. When the final search string has been determined, it is used to query different literature databases and duplicates must be filtered out, i.e., the intersection of two result sets \( A \cap B \) must be determined. These calculations involving thousands of articles are very time consuming and error prone. Even though most citation managers support the filtering of duplicates, the export of result sets and import into the manager is not feasible for rapid prototyping of queries.

Objective of this work is to provide a command line interface supporting the operations described above on result sets from search queries on literature databases. The tool should be capable of sending queries to literature databases automatically via their web-based application programming interfaces (APIs) and support the integration into existing analysis scripts via inline parameters. The tool should be flexible enough to allow the extension to any literature database providing an API.

2. Methods

Literature or academic databases are large collections of references to peer-reviewed scientific research articles. Besides the references themselves, these databases store structured metadata, like authors’ names or journal information, to support detailed search queries and allow scientifically valid citations. Most literature databases provide their own online search engine on these metadata and an API for querying articles programmatically. Usually, these literature databases are focused on specific fields of research. In this work, we will exemplarily focus on PubMed (www.pubmed.gov) and DBLP (www.dblp.org).

PubMed is a free to use literature database covering life sciences and biomedical topics [2]. In 1997, the database was first available to the public and is currently maintained by the United States National Library of Medicine (NLM) at the National Institutes of Health (NIH). At the time of writing, PubMed consists of over 34 million scientific references. Its search engine supports the search for phrases, i.e., exact order of words, and any combination of logical operations “and”, “or” and “not”, grouped by brackets. In addition, search terms can be limited to a certain metadata filed, e.g., author’s name, by using a postfix notation. The search API is well documented and free to use.

The Digital Bibliography & Library Project (DBLP) is a literature database containing conference and journal articles from the field of computer science [3]. DBLP was founded in 1993 by Michael Ley at the University of Trier. Since 2018, the database is hosted and maintained as a service by the Leibniz Center for Informatics Schloss Dagstuhl. At the time of writing, DBLP consists of over 6 million references. Its search engine does not support the search for phrases, however, by using “-” like “first-second” it can be enforced that “second” must appear anywhere after “first”. By default, each word of the search string is implicitly connected via an “and”. The logical “or” is only supported on the level of single words. The logical “not” as well as groupings and brackets is not supported and will be ignored. Similar to PubMed, each search term can be limited to a certain metadata field by using a prefix notation. Again, the search API is well documented and free to use but limited to 10000 references per query result.
3. Results

3.1. The Command Line Interface

The application is available as open-source software under the MIT license. It is written in Python 3.11 and provides a command line interface for the programmatic integration into more complex analysis scripts. The tool can be called with “py literature-cli.py <parameters>“. The parameter “-h” shows the instructions and a list of available parameters. The tool offers two modes, which support sending multiple queries to a single literature database or a single query to multiple databases. For the remainder of this section, we will focus on the second use case since both modes work similarly.

First, the query is transformed into the syntax of each supported database. Logical operations like “AND”, “OR” and “NOT” are replaced according to the requirements of each literature database, e.g., in case of DBLP the replacement of “OR” with “|” and removal of all “AND” operators, which are implicit.

Afterwards, the transformed search string is sent to the corresponding API endpoints. By default, only the metadata of the result set, containing the authors’ names, title, journal, publication year and DOI, are further processed. If a DOI is present, it is used to identify a publication in different result sets. Otherwise, the heuristic of matching title and year could be applied since both values are most consistent between different literature databases and are not affected by abbreviations. All combinations of intersections and differences of the result sets are calculated, which, for q queries sent, are $2^q - 1$ sets. These subsets can be exported as separate CSV-files containing all metadata for further post-processing or in the Research Information System (RIS) format for the upload into a citation manager as starting point of a systematic review. Furthermore, if defined in the parameter list, a Venn diagram is provided as visual feedback [4]. To provide a history of all requests and enable reproducibility, all query parameters and results are stored in a logging file as shown in Table 1.

3.2. Configuration and Extensibility

The internal structure of the tool is designed to enable easy extensibility to additional databases, as long as they provide a web-based API. For each new database, a class must be implemented inheriting from the connection base interface. The class implements the specific API endpoints, because structure and call-order are highly literature database dependent. Besides the implementation, a configuration file must be provided, which contains the exact location of metadata in the response. The associated XML-tags in the response can be defined by XPath. Furthermore, a mapping to the tool’s syntax must be provided, e.g., the handling of logical operations like “AND”, “OR” and “NOT”.

The metadata, which identifies a publication, if no DOI is provided, can be configured application-wide. In addition, all metadata, which should be exported or taken into consideration during analysis can be specified and extended as well. These changes must be replicated to the individual configuration of each connected database, i.e., the path to the corresponding XML element in the response message.

3.3. Example Workflow

The tool is most useful when processing large quantities of results that can hardly be managed manually. For comprehensibility, a minimal example is considered here. Let us
assume that a systematic review about machine learning approaches in the context of the rare Kawasaki disease should be performed [5]. Since it is an interdisciplinary topic between computer science and medicine, the literature databases DBLP and PubMed should be included in the systematic review. In the following, the process of finding the appropriate search string using the aforementioned tooling will be explained.

First, we directly experiment on the PubMed website by searching “Kawasaki disease” AND “machine learning” getting 16 results. Then, we try a word with a related meaning and replace “machine” by “deep” and receive 7 results. As shown in line 1 of Table 1, we use the tool to compare both results. There is only an overlap of 2 articles so combining both terms is beneficial. Back on the PubMed website, we combine both terms with the logical “OR” operator and get 21 results as expected. We are satisfied with the current result and want to apply it to DBLP. Unfortunately, DBLP does not support the phrase search and only supports OR for single tokens. Therefore, we restructure the query in a way that it can be interpreted correctly by DBLP. As shown in line 2, we verify by using the tool that our transformed query does not lose previous publications in PubMed. Instead, we are now getting 23 results, but keeping all previous 21 publications. Finally, we apply the query to both literature databases and get our result of 26 unique publications for our systematic review as shown in line 3. The corresponding RIS-files can directly be loaded into our citation manager.

<table>
<thead>
<tr>
<th>Line</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>py literature-cli.py –m alternate –db pubmed –q ' “Kawasaki disease” AND “machine learning” ' </code></td>
<td>q1: 16 q2: 7 q1∩q2: 2</td>
</tr>
<tr>
<td>2</td>
<td><code>py literature-cli.py –m alternate –db pubmed –q ' “Kawasaki disease” AND “deep learning” ' </code></td>
<td>q1: 21 q2: 23 q1∩q2: 21</td>
</tr>
<tr>
<td>3</td>
<td><code>py literature-cli.py –v –o ris –q ' “Kawasaki-disease” AND (machine OR deep) AND learning ' </code></td>
<td>pubmed: 23 dblp: 4 pubmed∩dblp: 1</td>
</tr>
</tbody>
</table>

### 4. Discussion

All pre-defined requirements of the tool have been met. As illustrated by the example workflow, it can help identifying publications during a systematic review. By providing it as open-source, it can be used by other researchers freely.

Tools like DistillerSR (www.distillersr.com) or Rayyan [6] also support systematic reviews. However, both are not open-source. They use advanced artificial intelligence to filter duplicate references and guide the user through the entire PRISMA workflow. In case of DistillerSR, even an API call to PubMed is supported. Nevertheless, these tools require a final query or already exported lists of references. Thus, our tool can be applied as a kind of preprocessing to determine the required references in a rapid-prototyping fashion, before the main PRISMA workflow begins.

A few limitations need to be addressed though. The tool was designed as a command line interface to integrate it programatically into complex analysis scripts. The nature of the command line may be off-putting to technically unsophisticated users or even limit its usability. A graphical user interface could certainly promote acceptance in the context...
of future work. Until then, there is still the possibility to use the search engines of the literature databases directly as shown in the example workflow.

Secondly, only two literature databases are currently integrated, primarily due to problems with not freely usable APIs. Well-known literature databases such as Web of Science (www.webofscience.com) or Scopus (www.scopus.com) have paid licenses or usage restrictions (number of hits per week), which make meaningful free use harder. Another example is the literature database Google Scholar (scholar.google.de), which requires a paid third party provider license (www.serpapi.com).

The support of search tags and a general pagination approach are planned as future work. Many literature databases allow the restriction of individual search terms to specific metadata, such as searching only for authors’ names, e.g., by using the postfix “[AU]” in PubMed or prefix “author:” in DBLP. The usage of such terms is generally supported by the application. However, corresponding queries should only be sent to a single database, since the query can be misinterpreted by other databases due to the greatly varying syntax. This issue could be handled similarly to logical operators during query conversion by applying mappings from the configuration files.

Some APIs limit the number of results returned per request. Currently, the maximum number of possible results of a single query is delivered, which is 10000 hits for PubMed and DBLP. Results above 10000 hits are considered as too low precision and are therefore ignored. In the future, this should be addressed by dynamically loading all publications, if pagination is supported by the API or heuristically implemented by adding the publication year to the search string and iterating over each year that contains at least one publication in the result set.

5. Conclusions

In this work, we presented a command line tool, for the calculation of intersection and differences of article result sets from automated queries to literature databases. The tool fulfills all pre-defined requirements and can help during the process of conducting a systematic review. Currently, the connection to PubMed and DBLP is implemented but a connection to further databases providing an API can easily be added. The source code is available from https://imigitlab.uni-muenster.de/published/literature-cli.

6. References