Dockerizing indri for OSIRRC 2019

Hauff, Claudia

Publication date
2019

Document Version
Final published version

Published in
OSIRRC 2019 The Open-Source IR Replicability Challenge

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.
Dockerizing Indri for OSIRRC 2019

Claudia Hauff
Delft University of Technology
Delft, The Netherlands
c.hauff@tudelft.nl

ABSTRACT
The Lemur Project was set up in 2000 by the Center for Intelligent Information Retrieval at UMass Amherst. It is one of the longest lasting open-source projects in the information retrieval (IR) research community. Among the released tools is Indri, a popular search engine that was designed for language-modeling based approaches to IR. For OSIRRC 2019 we dockerized Indri and added support for the Robust04, Core18 and GOV2 test collections.

1 OVERVIEW

As part of the Lemur Project\(^1\) a number of tools have been developed, most notably Galago, Indri \(^4\), and RankLib. Indri has been—and still is—a widely popular research search engine implemented in C++ which allows for the efficient development and evaluation of novel language-modeling based approaches to IR. In addition, Indri offers a query language that provides support for constraints based on proximity, document fields, syntax matches, and so on.

We here describe the implementation of the Indri Docker image\(^2\) for the OSIRRC 2019 challenge, the incorporated baselines, results and issues observed along the way.

2 DOCKER IMAGE DESIGN

The design of our Docker image is tied to the jig,\(^3\) a toolkit developed specifically for OSIRRC 2019, which provides a number of “hooks” (such as index and search) that are particular to the workflow of search systems.

2.1 Dockerfile

The Dockerfile builds an image based on Ubuntu 16.04. Apart from Indri v5.13 itself, a number of additional software package are installed such as node.js (one of the scripts to prepare the Core18 collection is a Node.js script) and python (to interact with the jig).

2.2 index

This hook indexes the corpora mounted by the jig, making use of Indri’s IndriBuildIndex. We support three corpora (Core18, GOV2 and Robust04), which each require different preprocessing steps:

Robust04 The original Robust04 corpus is .z compressed, a compression format Indri does not support. And thus, we first need to uncompress the corpus and filter out undesired folders such as cr (as Indri does not support excluding particular subfolders from indexing) before starting the indexing process.

Core18 The corpus is provided in JSON format and first needs to be converted to a document format Indri supports.

GOV2 Among the three corpora only GOV2 is well suited for Indri, it can be indexed without any further preprocessing.

The created indices are stemmed (Krovetz) with stopwords removed. For the latter, we relied on the Lemur project stopword list\(^4\) which contains 418 stopwords.

2.3 search

This hook is responsible for creating a retrieval run.

**Topic files.** In a preprocessing step, the TREC topic files (an example topic of Robust04 is shown in Figure 1) have to be reformatted as Indri requires topic files to adhere to a particular format.

Next to reformatting, special characters (punctuation marks, etc.) have to be removed. Indri does not provide specific tooling for this step, and one either has to investigate how exactly Indri deals with special characters during the indexing phase (thus matching the processing of special characters in order to achieve optimal retrieval effectiveness) or rely on very restrictive filtering (removing anything but alphanumeric characters). We opted for the latter. In contrast, stemming does not have to be applied, as Indri applies the same stemming to each query as specified in the index manifest (creating during the indexing phase).

Only standard stopword removal is applied to the topics; this means that in the TREC description and TREC narrative phrases

---

\(^1\)https://www.lemurproject.org/
\(^2\)https://github.com/osirrc/jig
\(^3\)https://github.com/osirrc/indri-docker/
\(^4\)http://www.lemurproject.org/stopwords/stoplist.dft
language modeling approach and their combination performs best. BM25 performs somewhat worse than expected, an outcome we argue is due to our lack of hyperparameter tuning. The biggest differences can be found in the results we report for queries solely derived from the TREC topic descriptions (instead of a combination of title and description): our results are significantly worse than the title-only baseline, which we attribute to a lack of “cleaning up” those descriptions (i.e. removing phrases like Relevant documents include).

4 CONCLUSIONS
Creating the Docker image for Indri was more work than anticipated. One unexpected problem turned out to be the sourcing of the original corpora (instead of processed versions suited for Indri that had been “passed down” from researcher to researcher within our lab). In addition, for almost every corpus/topic set combination a different preprocessing script had to be written which turned into a lengthy process as (i) Indri tends to fail silently (e.g. a failure to process a query with special characters will only be flagged when running trec_eval as the exception is simply written to the result file) and (ii) debugging a Docker image is not trivial.

In the next step, we will implement automatic hyperparameter tuning.

ACKNOWLEDGEMENTS
This research has been supported by NWO project SearchX (639.022.722).

---

All retrieval methods as documentend at https://lemurproject.org/doxygen/lemur/html/IndriRunQuery.html are supported.
Table 1: Overview of the optional parameter settings of the search hook and the corresponding retrieval effectiveness as measured in MAP.

<table>
<thead>
<tr>
<th>Parameter Settings</th>
<th>Robust04</th>
<th>GOV2</th>
<th>Core18</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;method:dirichlet,mu:1000&quot; topic_type=&quot;title&quot;</code></td>
<td>0.2499</td>
<td>0.2800</td>
<td>0.2332</td>
</tr>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;method:dirichlet,mu:1000&quot; topic_type=&quot;title&quot; sd=&quot;1&quot;</code></td>
<td>0.2547</td>
<td>0.2904</td>
<td>0.2428</td>
</tr>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;method:dirichlet,mu:1000&quot; topic_type=&quot;title&quot; use_prf=&quot;1&quot;</code></td>
<td>0.2812</td>
<td>0.3033</td>
<td>0.2800</td>
</tr>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;method:dirichlet,mu:1000&quot; topic_type=&quot;title&quot; use_prf=&quot;1&quot; sd=&quot;1&quot;</code></td>
<td><strong>0.2855</strong></td>
<td><strong>0.3104</strong></td>
<td><strong>0.2816</strong></td>
</tr>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;okapi,k1:1.2,b:0.75&quot; topic_type=&quot;title+desc&quot;</code></td>
<td>0.2702</td>
<td>0.2705</td>
<td>0.2457</td>
</tr>
<tr>
<td><code>--opts out_file_name=&quot;outfile&quot; rule=&quot;method:dirichlet,mu:1000&quot; topic_type=&quot;desc&quot;</code></td>
<td>0.2023</td>
<td>0.1336</td>
<td>0.1674</td>
</tr>
</tbody>
</table>

REFERENCES


