Pyndri*: a Python Interface to the Indri Search Engine

Christophe Van Gysel, Evangelos Kanoulas, and Maarten de Rijke

University of Amsterdam
Amsterdam, The Netherlands
{cvangysel,e.kanoulas,derijke}@uva.nl

Abstract. We introduce **pyndri**, a Python interface to the Indri search engine. Pyndri allows to access Indri indexes from Python at two levels: (1) dictionary and tokenized document collection, (2) evaluating queries on the index. We hope that with the release of pyndri, we will stimulate reproducible, open and fast-paced IR research.

1 Introduction

Research in Artificial Intelligence progresses at a rate proportional to the time it takes to implement an idea. Therefore, it is natural for researchers to prefer scripting languages (e.g., Python) over conventional programming languages (e.g., C++) as programs implemented using the latter are often up to three factors longer (in lines of code) and require twice as much time to implement [9]. Python, an interactive scripting language that emphasizes readability, has risen in popularity due to its wide range of scientific libraries (e.g., NumPy), built-in data structures and holistic language design [6].

There is still, however, a lack of an integrated Python library dedicated to Information Retrieval (IR) research. Researchers often implement their own procedures to parse common file formats, perform tokenization, token normalization that encompass the overall task of corpus indexing. Uysal and Gunal [11] show that text classification algorithms can perform significantly differently, depending on the level of preprocessing performed. Existing frameworks, such as NLTK [7], are primarily targeted at processing natural language as opposed to retrieving information and do not scale well. At the algorithm level, small implementation differences can have significant differences in retrieval performance due to floating point errors [4]. While this is unavoidable due to the fast-paced nature of research, at least for seminal algorithms and models, standardized implementations are needed.

2 Introducing Pyndri

Fortunately, the IR community has developed a series of indexing frameworks (e.g., Galago, Lucene, Terrier) that correctly implement a wide range of retrieval models. The Indri search engine [10] supports complex queries involving *evidence combination* and the ability to specify a wide variety of constraints involving *proximity*, *syntax*, *extracted entities* and *document structure*. Furthermore, the framework has been efficiently implemented using C++ and was designed from the ground up to support *very*

* https://github.com/cvangysel/pyndri

```
index = pyndri.Index('/opt/local/clueweb09')

for int_doc_id in range(index.document_base(),
                        index.maximum_document()):
    ext_doc_id, doc_tokens = index.document(int_doc_id)
```

Code snippet 1: Tokenized documents in the index can be iterated over. The `ext_doc_id` variable in the inner loop will equal the document identifier (e.g., `clueweb09-en0039-05-00000`), while the `doc_tokens` points to a tuple of integers that correspond to the document term identifiers.

```
index = pyndri.Index('/opt/local/clueweb09')
dictionary = pyndri.extract_dictionary(index)

_, int_doc_id = index.document_ids(
    ['clueweb09-en0039-05-00000'])
print([dictionary[token_id]
       for token_id in index.document(int_doc_id)[1]])
```

Code snippet 2: A specific document is retrieved by its external document identifier. The index dictionary can be queried as well. In the above example, a list of token strings corresponding to the document's contents will be printed to `stdout`.

large databases, *optimized query execution* and *fast and concurrent indexing*. A large subset of the retrieval models [1, 2, 5, 12, 13] introduced over the course of history can be succinctly formulated as an Indri query. However, to do so in an automated manner, up until now researchers were required to resort to C++, Java or shell scripting. C++ and Java, while excellent for production-style systems, are slow and inflexible for the fast prototyping paradigm used in research. Shell scripting fits better in the research paradigm, but offers poor string processing functionality and can be error-prone. Besides, shell scripting is unsuited if one wants to evaluate a large number of complex queries or wishes to extract documents from the repository as this incurs overhead, causing avoidable slow execution. Existing Python libraries for indexing and searching, such as PyLucene, Whoosh or ElasticSearch, do not support the rich Indri language and functionality required for rapid prototyping.

We fill this gap by introducing pyndri, a lightweight interface to the Indri search engine. Pyndri offers read-only access at two levels in a given Indri index.

2.1 Low-level access to document repository First of all, pyndri allows the retrieval of tokenized documents stored in the index repository. This allows researchers to avoid implementing their own format parsing as Indri supports all major formats used in IR, such as the trectext, trecweb, XML documents and Web ARChive (WARC) formats. Furthermore, standardized tokenization and normalization of texts is performed by Indri and is no longer a burden to the researcher. Code snippet 1 shows how a researcher can easily access documents in the index. Lookup of internal document identifiers given their external name is provided by the `Index.document_ids` function.

The `dictionary` of the index (Code snippet 2) can be accessed from Python as well. Beyond bi-directional token-to-identifier translation, the dictionary contains corpus statistics such as term and document frequencies as well. The combination of index

```
index = pyndri.Index('/opt/local/clueweb09')

for int_doc, score in index.query('obama family tree'):
    # Do stuff with the document.
```

Code snippet 3: Simple queries can be fired using a simple interface. Here we query the index for topic `wt09-1` from the TREC 2009 Web Track using the Indri defaults (Query Language Model (QLM) with Dirichlet smoothing, $\mu = 2500$).

```
index = pyndri.Index('/opt/local/clueweb09')
query_env = pyndri.QueryEnvironment(
    index, rules=('method:dirichlet,mu:5000',))

results = query_env.query(
    '#weight( 0.70 obama 0.20 family 0.10 tree )',
    document_set=map(
        operator.itemgetter(1),
        index.document_ids([
            'clueweb09-en0003-55-31884',
            'clueweb09-en0006-21-20387',
            'clueweb09-enwp01-75-20596',
            'clueweb09-enwp00-64-03709',
            'clueweb09-en0005-76-03988'
        ])),
    results_requested=3,
    include_snippets=True)

for int_doc_id, score, snippet in results:
    # Do stuff with the document and snippet.
```

Code snippet 4: Advanced querying of topic `wt09-1` with custom smoothing rules, using a weighted-QLM. Only a subset of documents is searched and we impose a limit on the size of the returned list. In addition to the document identifiers and their retrieval score, the function now returns snippets of the documents where the query terms match.

iteration and dictionary interfacing integrates conveniently with the Gensim[1] package, a collection of topic and latent semantic models such as LSI [3] and word2vec [8]. In particular for word2vec, this allows for the training of word embeddings on a corpus while avoiding the tokenization mismatch between the index and word2vec. In addition to tokenized documents, pyndri also supports retrieving various corpus statistics such as document length and corpus term frequency.

2.2 Querying Indri from Python Secondly, pyndri allows the execution of Indri queries using the index. Code snippet 3 shows how one would query an index using a topic from the TREC 2009 Web Track using the Indri default retrieval model. Beyond simple terms, the `query()` function fully supports the Indri Query Language.[2]

In addition, we can specify a subset of documents to query, the number of requested results and whether or not snippets should be returned. In Code snippet 4 we create a `QueryEnvironment`, with a set of custom smoothing rules. This allows the user to apply fine-grained smoothing settings (i.e., per-field granularity).

[1] `https://radimrehurek.com/gensim`

[2] `http://lemurproject.org/lemur/IndriQueryLanguage.php`

3 Conclusions

In this paper we introduced pyndri, a Python interface to the Indri search engine. Pyndri allows researchers to access tokenized documents from Indri using a convenient Python interface. By relying on Indri for tokenization and normalization, IR researchers are no longer burdened by this task. In addition, complex retrieval models can easily be implemented by constructing them in the Indri Query Language in Python and querying the index. This will make it easier for researchers to release their code, as Python is designed to be readable and cross-platform. We hope that with the release of pyndri, we will stimulate **reproducible**, **open** and **fast-paced** IR research. More information regarding the available API and installation instructions can be found on Github.[3]

Acknowledgements. This research was supported by the Google Faculty Research Award program and the Bloomberg Research Grant program. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

Bibliography

[1] Balog, K., Azzopardi, L., de Rijke, M.: Formal models for expert finding in enterprise corpora. In: SIGIR. pp. 43–50. ACM (2006)
[2] Bendersky, M., Metzler, D., Croft, W.B.: Learning concept importance using a weighted dependence model. In: WSDM. pp. 31–40. ACM (2010)
[3] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6), 391–407 (1990)
[4] Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys 23(1), 5–48 (1991)
[5] Guan, D., Zhang, S., Yang, H.: Utilizing query change for session search. In: SIGIR. pp. 453–462. ACM (2013)
[6] Koepke, H.: Why python rocks for research. https://www.stat.washington.edu/~hoytak/_static/papers/why-python.pdf (2010), accessed October 13, 2016
[7] Loper, E., Bird, S.: NLTK: The natural language toolkit. In: ACL Workshop on Effective Tools and Methodologies for teaching NLP and CL. pp. 63–70. Association for Computational Linguistics (2002)
[8] Mikolov, T., Corrado, G., Chen, K., Dean, J.: Efficient estimation of word representations in vector space. arXiv 1301.3781 (2013)
[9] Prechelt, L.: An empirical comparison of seven programming languages. Computer 33(10), 23–29 (Oct 2000)
[10] Strohman, T., Metzler, D., Turtle, H., Croft, W.B.: Indri: A language model-based search engine for complex queries. In: ICIA (2005)
[11] Uysal, A.K., Gunal, S.: The impact of preprocessing on text classification. Information Processing & Management 50(1), 104–112 (2014)
[12] Van Gysel, C., Kanoulas, E., de Rijke, M.: Lexical query modeling in session search. In: ICTIR. pp. 69–72. ACM (2016)
[13] Zhai, C., Lafferty, J.: A study of smoothing methods for language models applied to ad hoc information retrieval. In: SIGIR. pp. 334–342. ACM (2001)

[3] https://github.com/cvangysel/pyndri