How-To: Indri

In this document we briefly explain how to install and use Indri. We only give details on the most basic functions, more advanced usage is up to you. This How-To is created for Indri beginners, if you are somewhat more experienced, this How-To won't give you any new insights. The How-To discusses usage for Windows, Mac, and Linux.

The first section explains how to install Indri, followed by sections on indexing, index checking, and retrieval. The sections contain example parameter files and queries, so that you can easily start exploring Indri using only this document.

Installing Indri

Go to the Lemur website[1] and download the latest version of Indri for your OS. Currently (Sep 27, 2011), the latest version is 5.1. For Windows there are installers available (32 and 64 bit), make sure you choose `full` during the installation process. For Mac and Linux you can simply compile Indri; follow the regular sequence of (1) configuring (`./configure`), where you can select the installation directory using `--prefix=path/to/location`, (2) `make`, and (3) `(sudo) make install`. Installing is as simple as that.

Although Indri offers a graphical user interface (GUI) we won't be using it (in this document). The options offered by the GUI are limited, especially on the retrieval side (e.g., it doesn't support batch retrieval, running multiple queries in a batch). To run a proper IR experiment we need to use the command line interface, which isn't as hard as it sounds. To use a command line in Windows, type `cmd` in the search box in your start menu. Locate the Indri binaries using the command line, that is, go to the installation directory of Indri (e.g., `cd C:\Program Files\Indri\indri-5.1\bin` or `cd /usr/local/bin`). Here, you find binaries for indexing (`IndriBuildIndex`), index checking (`dumpindex`) and retrieval (`IndriRunQuery`). In the remainder of the document we use the command line notation for Windows (including `.exe`), just remember that is exactly the same for Mac and Linux, without the `.exe` part.

We now continue with creating an index for one document, just to get acquainted with Indri's indexing options.

[1] http://sourceforge.net/projects/lemur/

Indexing

To create an index using Indri we need `IndriBuildIndex.exe`. Running this on the command line without any options doesn't do anything (except complain), so we need a set of parameters to tell the indexer what to do. Below is an example parameter file for indexing one document. We go over the various options below.

```
<parameters>

 <index>D:\indri\index</index>

 <corpus>
  <class>txt</class>
  <path>D:\indri\collection\test.txt</path>
 </corpus>

 <stemmer>
  <name>porter</name>
 </stemmer>

 <stopper>
  <word>a</word>
  <word>the</word>
  <word>of</word>
 </stopper>

</parameters>
```

The `index` tag contains the location where the to-be-created index should be stored. Make sure you have write permissions to this location. Indri can be sensitive to white spaces in paths, so it's best to keep the location simple, as in the example.

The `corpus` part of the parameter file indicates which documents should be indexed and what format the documents have. In the example we only index one document (`test.txt`), but you can also point to a directory, which causes Indri to index all documents in this directory. The `class` tells Indri how the document(s) should be processed. In the example, we try to index a text file, so we add `<class>txt</class>`. Other commonly used options here are `doc` (Word documents, only works on Windows), `pdf`, `xml`, and `html`. If you run TREC-like experiments, you will mostly use `trectext` (or `warc` for Clueweb09). Note that the `class` tag is NOT a filter: all documents in your `path` are processed as if they belong to this class. If some files are a different class, it will most likely cause Indri to fail, index them faulty, or skip them.

The following two parts, `stemmer` and `stopper`, are not obligatory. Leaving these two out makes that Indri creates a non-stopped and non-stemmed index. For the `stemmer` there are three options: leave out, `porter`, or `krovatz`. We do not go into details about the actual stemming algorithms here. For stopword removal, Indri requires us to provide a list of stopwords, where each word is contained within `<word>` tags. In the example we added three stopwords.

Tip: You can combine multiple parameter files on the command line. One easy way would be to make a separate stopword parameter file, that you can re-use every time you want to create an index without stopwords. Below is an example of a separate stopword parameter file.

```
<parameters>
 <stopper>
  <word>a</word>
  <word>the</word>
  <word>of</word>
 </stopper>
</parameters>
```

We are now ready to index a file. Create a text file that contains something like this

```
John uses a stick to scare the dogs away.
One dog sticks to his jeans.
```

and save it. Make sure the parameter file(s) point to the right locations and run the following command:

```
IndriBuildIndex.exe indexparameters.xml stopwords.xml
```

Indri now indexes the file (it takes only a few ms) and creates the index on the specified location. If you encounter errors, make sure your paths are ok, permissions are ok, filetypes are ok, etc.

Checking the index

Having indexed one document, we now need to check the index. Indri offers us a simple tool, `dumpindex.exe`, that allows us to get some basic statistics from an index. To see what you can do with it, just type `dumpindex.exe` and it shows you all the options.

For now, we only need to do one thing: check basic statistics of the index. To this end, type `dumpindex.exe D:\indri\index s`, where `D:\indri\index` is the location of your index. If everything is ok, you get an output that shows the number of documents in the index, the number of terms, and the number of unique terms. This is a very simple and fast way to check if your index is ok.

To get used to indexing and Indri's options, try creating various indexes with different setting, e.g. with and without stemming, with and without stopwords, one file vs. multiple files, different file classes, etc. After each setting, use `dumpindex.exe` to check the stats, you'll see that these change depending on the settings. Can you predict what would change in the statistics if you index the example document with and without stemming? Or with and without (the example) stopwords?

Retrieval

So far we have indexed one document and checked the index to see if it "looks" like we expected. It is always a good idea to first check the index before starting your retrieval experiments. Running a query against the index is fairly simple and involves creating another parameter file, to be used with `IndriRunQuery.exe`. Below we list an example parameter file.

```
<parameters>

 <index>D:\indri\index</index>
 <count>100</count>
 <trecFormat>1</trecFormat>

 <query>
  <number>1</number>
  <text>dog</text>
 </query>

</parameters>
```

The `index` tag once again contains the location of the index. `count` tells Indri how many results to return for each query (this is a maximum, there can be less results) and `trecFormat` is a boolean field indicating whether or not Indri should return results in a TREC format. The TREC format can be used in case you want to evaluate your results using `trec_eval` (this is the case for most IR experiments, so it is best to set this option to `1`).

The actual query we want to execute is listed between the `query` tags. Queries usually have a number/ID (to allow evaluating them all at once), which is contained in `<number>`. The actual query is written in `<text>`. In this case it only contains one term, but we could also add more terms (`dog stick`).

Tip: As with indexing, we can combine various parameter files. In most cases we split "general" retrieval settings, like index and result count, and the actual queries. An example query parameter file would look like this:

```
<parameters>
 <query>
  <number>1</number>
  <text>dog</text>
 </query>

 <query>
  <number>2</number>
  <text>bike path</text>
 </query>
</parameters>
```

We can now invoke the retrieval tool:

```
IndriRunQuery.exe retrievalparam.xml queries.xml
```

Not that `IndriRunQuery` outputs the results of the queries to the terminal. You need to redirect the output to a file to save the results somewhere (and being able to evaluate them). Add `> D:\indri\runs\output.txt` at the end to save the retrieval results in the specified location. Make sure you do not overwrite previous results!

TREC result files, that is, the file you just created (`output.txt`), contain six columns and look somewhat like this:

```
1 Q0 GHJ-001 1 -7.03 indri
1 Q0 JHV-101 2 -7.34 indri
1 Q0 GHJ-231 3 -8.93 indri
```

The first column, `1`, is the query ID (number), the second column, `Q0`, is a legacy field and can be ignored, the third column, e.g., `GHJ-001`, is the ID of the retrieved document (in trectext format document IDs are stored in `<DOCNO>` tags), the fourth column, e.g., `1`, is the rank of the document, the fifth column, e.g., `-7.03`, is the retrieval score of the document (higher is better), and the sixth column, `indri`, is the ID of the run. You can use this last column to give a name to the run, although you would mostly do that in the filename anyway.

Given a TREC result file and a TREC evaluation file (qrels) you can run `trec_eval` to evaluate your retrieval performance.

A bit more

After indexing a collection and obtaining a set of queries and qrels, what kind of IR experiments can you run? There are quite a few options here. Of course, you can play around with stemming and stopword removal. You can look into smoothing methods (`<rule>` parameter) or different retrieval models (`<baseline>` setting). Finally, you can explore relevance feedback (`<fbDocs>` and other `fb` parameters). For more Indri parameter options, check:
http://lemur.sourceforge.net/indri/IndriParameters.html

Wouter Weerkamp and **Edgar Meij**
September 28, 2011, Amsterdam