

Adaptive Filtering Software - Appendix 1: Lemur Guide

Contents

1. Installing Lemur
 - Patching Lemur
 - Special note on using Lemur 1.1
2. Lemur Environment Variables
3. Formatting Your Corpus for Lemur

Adaptive Filtering Software uses the Lemur Toolkit for Language Modeling and Information Retrieval (<http://www-2.cs.cmu.edu/~lemur/>) from Carnegie Mellon University for managing compressed index files of the data sets being processed. This document explains issues related to installing Lemur on your system and preparing your data corpus for use with Lemur.

Installing Lemur

The Adaptive Filtering Software uses the Lemur Toolkit for Language Modeling and Information Retrieval from Carnegie Mellon University. You need to download the current version of the Lemur toolkit from its website, <http://www-2.cs.cmu.edu/~lemur/>, and install it on your system. Follow the instructions provided at that site ("Installing and Running Lemur"); however, please check the note on "Patching Lemur", below, before compiling Lemur. Several versions of Lemur may be available at that site. At the time of this writing (January 2003), we have successfully compiled and run our software with an older Lemur version (Lemur 1.1), as well as with the most current version (Lemur 2.1.1). If a more recent version is available to you, you can try it instead.

What Lemur version to use?

It is probably advisable to use Lemur 2.1, or higher version. However, that requires GCC 3.*; if you only have GCC 2.95 and don't want to upgrade, you can try to use Lemur 2.0, or even 1.1 instead. To avoid linkage problems, it is desirable to use the same compiler when building Lemur and when building Adaptive Filtering software. The table below summarizes recommended compilers.

Lemur Version	Recommended compiler	Notes
1.1	GCC 2.9* only	Special care needed to build the lemur library. See the <u>note</u> below.
2.0	GCC 3.* or GCC 2.9*	
2.1 and higher	GCC 3.* only	

Patching Lemur

The source code of the currently tested versions of Lemur (both Lemur 1.1 and Lemur 2.1.1) contains a small bug in their source code, which you need to (partially) correct before you compile it. The file in question is `/utility/src/BasicDocStream.cpp`. In that file, find the method `bool BasicDocStream::hasMore()` and in that method replace the line

```
bool moreDoc;
```

with

```
static bool moreDoc;
```

Special note on using Lemur 1.1

If you do choose to use Lemur 1.1 for some reason, be aware that you may have problems using their library, `$LIB_LEMUR/lib/liblemur.a` for linking. You won't know for sure that you have this problem until you try to run `make` for your application (e.g. the Adaptive Filtering Software), and find that, although everything compiles, linking fails. If this is the case on your platform, read on; otherwise, you can safely skip this section.

If you look at the content of this library using `ar`, you may find that it consists of other libraries, rather than directly of object files:

```
%ar t /usr/local/lemur-1.1/lib/liblemur.a
libutility.a
libindex.a
liblangmod.a
libretrieval.a
```

If this is the case, a special procedure needs to be followed: you need to link your program with these four component libraries instead of `liblemur.a`. This is how this can be done:

1. Create subdirectory `$LIB_LEMUR/libsplit`
2. Obtain the four component library files listed above, and copy them to `$LIB_LEMUR/libsplit`. These files may be found in the subdirectories of the Lemur source directory after Lemur was built:

```
% find . -name '*.a'
./index/obj/libindex.a
./langmod/obj/liblangmod.a
./retrieval/obj/libretrieval.a
./utility/obj/libutility.a
```

or you can extract them from `liblemur.a` using the command

```
ar x liblemur.a
```

3. Modify the Makefile for your application to link with these library files. For example, in the Makefile that comes with the Adaptive Filtering application this can be done by commenting the lines

```
LS = $(LEMUR)/lib  
LIBS = $(LS)/liblemur.a
```

and uncommenting the lines

```
LS = $(LEMUR)/lib-split  
LIBS = $(LS)/libretrieval.a $(LS)/liblangmod.a $(LS)/libindex.a $(LS)/libutility.a
```

If the instructions above seem confusing, consider upgrading to Lemur 2.* (which you probably should do anyway), or contact the Adaptive Filtering Software developers at vmenkov@aplab.rutgers.edu for details.

Lemur Environment Variables

Once Lemur has been installed on your machine, you need to set environment variable `LIB_LEMUR`. It should be set in this way every time in the future when you either compiling the Adaptive Filtering Software or using it. Therefore, it is advisable that you put an appropriate command into your shell startup file, as shown below.

If you use `csh` or `tcsh`: put the following command into your `.cshrc` or `.login` file:

```
setenv LIB_LEMUR /usr/local/lemur
```

If you use `sh` or `bash`: put the following command into, respectively, your `.profile` or `.bashrc` file:

```
export LIB_LEMUR=/usr/local/lemur
```

The variable value above implies that you have used `LEMUR_INSTALL_PATH` set to `/usr/local` during Lemur installation (`make install`), and the Lemur toolkit was thus installed in `/usr/local/lemur`. Change the value as appropriate if Lemur was installed elsewhere.

Formatting Your Corpus for Lemur

This section explains how to prepare your own data set for analysis by DIMACS Adaptive Filtering Software or some other DIMACS tools that use Lemur. We will illustrate everything on a simple sample corpus, found in the directory `data/mini1` of the DIMACS Adaptive Filtering Software distribution. (There is also another very small data set, in `data/mini2`, and a larger one in `data/orig-cacm-full`.)

Let us now take a look at this data set. If you are familiar with the [TREC competition](#), you may notice that the data format is quite similar to the one used there.

Below is the list of files in that directory, with explanations.

```
% cd data/mini1
% wc *
  90      90      743 train.sgml
  63      60      512 test.sgml
 153     150     1255 all.sgml
  10      15      128 query.sgml
   6       6       48 doc-order
 429     429     2485 stopwords.txt
   9       9      165 train-qrel
  12      36      220 test-qrel
```

1. train.sgml: Training set as an SGML file. Each document in this file looks like this:

```
<DOC>
<DOCNO> doc_id </DOCNO>

Document text
goes here

</DOC>
```

Doc_id is the unique document id in your corpus. It can be any combination of digits and letters.

2. test.sgml: The test set in the same SGML format as the training.set.
3. all.sgml: Training set + test set. You can create this file by concatenating train.sgml and test.sgml.
4. query.sgml: Description of all topics (categories) to which documents may belong. The file format is exactly the same as for train.sgml and test.sgml. Each topic is represented in this file as a "document"; it has an ID (topic ID) and optional "text", that is the topic description.

The Lemur application `ParseToFile` (invoked by the script `prepare.sh` coming with the DIMACS Adaptive Filtering Software) will be used to convert this *raw query file* to the query file in the format used by Adaptive Filtering Software. If you want to avoid that conversion stage, **and** none of your query has a non-empty query description, then you may instead create a query description file in the format directly used by Adaptive Filtering Software. In this file, typically named `query`, the entry for each topic looks as follows:

```
<DOC topic_id>
</DOC>
```

5. doc-order: Test set document order file. Contains the list of test documents that you want classified, and specifies the order in which test set documents will be processed. The order should not affect the classification results in batch filtering, but it matter for adaptive filtering. In any event, this file must list all test set documents that you want classified on this run, and only them. You probably will normally want to classify the entire test set, which means that you would list all test set documents in this file; but if you only want a subset of the test set classified, list only those. This will allow you to control what part of the test set is classified without having to rebuild Lemur indexes.

Normlly this is a 2-column text file with the following TREC-compatible format:

```
xx 301
xx 302
xx 303
...
```

The value in the first column is ignored; the second column is the document id (same as used in test.sgml).

With Adaptive Filtering Software, a simpler alternative one-column format is also allowed. In such a file, each line contains one document ID:

```
301
302
303
...
```

The document order file can be prepared from test.sgml using a scripting language such as Perl or awk. For example, you can use the following UNIX command to prepare a one-column document order file:

```
grep DOCNO test.sgml | perl -p -e 's/\<.\?DOCNO>//g' > doc-order
```

and then, if you desired, convert it to two-column format with:

```
perl -pi.bak -e 's/^/xx /' doc-order
```

Or you can run the one-line script mkDocOrder.sh found in the directory mini1.

6. stopwords.txt: The list of *stop words*. These are the terms that the Lemur parser will remove from all documents. Currently we use the list from SMART.
7. train-qrel: This is a *judgment file*, or *qrel file* for the training set. It contains the experts judgments for the training set documents, which will be used to train the classifier. Each line represents a judgment for a (document, topic) pair and may look like this:

```
Washington WA1 1
```

Here the first column contains the topic ID, the second the document ID, and the third expert judgment (1=relevant, 0=non-relevant). Please see the note on [Interpreting judgment files](#) to see how the classifier processes these judgments.

8. test-qrel: The experts ("oracle") judgments for the test set documents. The file is in the same format as the one for the training set, subject to the same note on [Interpreting judgment files](#).

What is the use of this file? In a real-life application, where the test set documents have not been looked at by the human experts (or other classification programs) yet, this file would be empty. But if you want to experiment with adaptive filtering, you can put there judgments that the expert would give to those documents. Or if you already know how the test-set documents are truly categorized, and want to see how well one classifier or another handles them, you can have the true judgments stored in this file, and the classifier will report its success at getting them right.

Note: you can set up your input data so that `test-qrel` and `train-qrel` to be in fact the same file (via a symbolic link, or by using the same file name for both `trainQrelFile` and `testQrelFile` in the parameter file. In this case, the program would use the same file twice, only use those lines from this joint file that correspond to the documents in training or test set, as appropriate.

You, of course, can use other file names than those given above. In that case you will have to modify the parameters in the Lemur utilities configuration files (`data/filterparam/build_all_param`, `data/filterparam/build_train_param`, `data/filterparam/parse_train_param`, `data/filterparam/parse_query_param`, etc.) and in your Adaptive Filtering Software configuration file (such as `data/filterparam/filter_param`) accordingly.

You may want to explore Lemur options, such as using a different stemmer, a different (or no) stop word list, or a different input file format. Refer to the documentation at the Lemur web site for details.