Lemur CGI details

Courtesy James Allan http://www-ciir.cs.umass.edu/~allan/
Courtesy Hema Raghavan

Created: *Mon 2 May 2005*
Last modified: 10/25/2013 03:35:40

The Lemur CGI Interface is available from: http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi

---

Q : ***Why is http://fiji4.ccs.neu.edu/ not working?***

IMPORTANT : from "community ports" http://fiji4.ccs.neu.edu/ is **not** accessible. Instead use the NAT http://10.0.0.176/~zerg/lemurcgi/lemur.cgi.

The instructions about accessing this interface are given below.

The basic format of a command to access the CGI is to access the above URL followed by a "?" which is followed by a command. The command is in the form: Name=value&Name=value&....&Name=value. The Name,value pairs are arguments to the CGI script. They are processed left to right. I'll explain why this is important in the section marked IMPORTANT TIPS below. The CGI program returns a web-page that contains a header (everything before the BODY tag), the results, and a footer (everything after the HR tag). The header and footer can be ignored. They are the same for every command, and their only purpose is to identify the software.

1. COMMANDS:

a. Help. To obtain a list of the available commands use "?h" as follows:
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?h

---

b. Available databases. To obtain a list of the available databases and corpus statistics use "d=?" :
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=? .

---

c. Database specific information. To obtain a database specific information start the query string with "d=<DATABASE NUMBER>" followed by a command as in:
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&c=star

---

d. Word Statistics Only. To obtain corpus statistics for the word "star" use the "c=<TERM>" command preceded by a database command "d=<DATABASE NUMBER>"
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&c=stars
This command retrieves the ctf (collection term frequency) and df (document frequency) of the term. Collection term frequency = number of times the term appears in the collection. Document frequency = number of documents in the collection that contain the term. Those statistics are important for the retrieval models. For stemmed databases the terms are automatically stemmed by Lemur. You do not need to stem. You should remove stop words from the query before you use Lemur.

---

e. Inverted Lists and Word Statistics. To obtain the inverted list for a term use the the "v=<TERM>" command. Prefix this command by the the database command "d=<DATABASE NUMBER>":
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&v=star
This command retrieves the ctf (collection term frequency) and df (document frequency) of the term in addition to the inverted list. For stemmed database you do not need to stem the provided terms. However, you need convert all the words to lowercase. Remove stop words from the query before using Lemur.

---

f. External Ids. Documents have both internal and external ids. External ids like WSJ890803-0148 are typically a

combination of source and date information. To retrieve a document with it's external ID use the "e?=" command. For example,
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?e=AP890101-0001
The above command will return the document AP890101-0001 in SGML format.

---

g. Internal Ids. Documents have both internal and external ids. Internal ids are integers. They are used by Lemur for efficiency. To retrieve a document in SGML format from the internal ID, use the "i=" command. For example:
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?i=1

---

h. Convertion from internal to external ids. Use the following file to convert from internal to external ids:
http://www.ccs.neu.edu/home/jaa/IS4200.10X1/projects/doclist.txt
Lemur will return the documents are internal ids, but to carry out the evaluation the internal ids need to be mapped to the corresponding external ids.

2. IMPORTANT TIPS:

a. Commands are processed left to right. Hence, parameters like database ids, or length of the ranked list must be specified before the query. For example, if you specify http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?n=5&q=star, you get back the top 5 documents for the query "star". However, http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?q=star&n=5 has little meaning, since the CGI processes commands from left to right and the value of the length of the ranked list would be set only after

"q=star" is processed and returned to the client, which has no meaning. Similarly
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&c=star returns the statistics of the word "star" in database 0.

3. IMPLEMENTATION TIPS

a. Process the query file manually. Remove punctuation. Remove stop words. Convert the remaining words to lower case. Check words with hyphen as they may not be handled properly by lemur. Check words like U.S.A.

---

b. We recommend the use of a scripting language like Perl, Python or Ruby to implement the project.
c. Add the command "g=p" to obtain a format which is easy to parse. For example the command:
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&v=star
becomes
http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&g=p&v=star
d. Use regular expressions for parsing where possible. Parsing will be the most code-instensive part. It is possible to implement the project in less than 160 lines (600 words, 6000 characters).
Here is an example how Ruby can handle parsing of the inverted list page:

```
#!/usr/bin ruby
#note: error handling is not included

#include libraries to be used
require 'open-uri' #for the http call
require 'ostruct'  #Open struct for creating structures

data = open("http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&g=p&v=star"). #create an http call
         read. #read the contents
         match(/<BODY>([^>]*)<HR>/)[1]. #use a regular expression to extract the numbers
         split(" ") #split the numbers by white space

ctf,df = data[0].to_f,data[1].to_f #take the ctf and df and convert to float
inverted_list = #put the inverted list in a list of (docid,doclen,tf) elements
          (0 ... (data.length - 2)/3).collect {|i|
                    doc = OpenStruct.new
                    doc.docid = data[2 + 3*i].to_i
                    doc.doclen = data[3 + 3*i].to_f
                    doc.tf = data[4 + 3*i].to_f
                    doc
```

```
          }
#print in-memory structures
puts "ctf= #{ctf} df= #{df}"
inverted_list.each {|entry|
  puts "#{entry.docid} #{entry.doclen} #{entry.tf}"
}
```

Example in Python:

```
#!/usr/bin python
import urllib, re
text = urllib.urlopen("http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?d=0&g=p&v=star").read()
data = re.compile(r'.*?<BODY>(.*?)<HR>', re.DOTALL).match(text).group(1)
numbers = re.compile(r'(\d+)',re.DOTALL).findall(data)

ctf,df = float(numbers[0]), float(numbers[1]) #take the ctf and df and convert to float
inverted_list = map(lambda i: (int(numbers[2 + 3*i]),
                               float(numbers[3 + 3*i]),
                               float(numbers[4 + 3*i]))
                    ,range(0, (len(numbers) - 2)/3))

print "ctf= %(ctf)f df= %(df)f" % {'ctf': ctf, 'df':df}
for (docid,doclen,tf) in inverted_list:
        print docid,doclen,tf
```

---

e. Make sure the retrieval formulas are implemented correctly. A common error occurs in the language model. Under the language model words that appear in the query but not in a retrieved document obtain a score different than zero. In traditional IR a document is considered for scoring even if it contains only one from a few query words. Therefore missing words still receive a score.