FeatureLens : System Manual

Author : anthony DON – don@cs.umd.edu, don@labri.fr
Date : April, 21th, 2007

Introduction

The FeatureLens application allows the user to load frequent patterns of words that occur in a collection of documents. These frequent patterns are computed out of the application and stored in a MySQL database along with the text of the documents from where they where extracted.

The application is a tool to select « interesting » frequent patterns and to show where they occur in the collection of documents at different scales. The collection of documents is structured by section, then by document. For a book, a section is a chapter and a document is a paragraph.

Architecture Overview

Some OpenLaszlo visual components are tied to an XML data source using the Dataset object. The XML document may be static or returned by a Web Service over HTTP. The application heavily relies on textual data and full text queries, it needed to :

- store the text documents in a structured way,
- have an efficient way to make full-text queries and format the output with text coloring,
- format the output documents into XML so that OL could use the results.

I use a MySQL server for storing text documents and frequent patterns (words and frequent patterns of 3-grams).

For the backend, Ruby and the WebRick package are used. They allow to quickly create a Web server that can process multiple connections and declare URLs that are bound to a class that will process the query strings (POST or GET) and produce the XML document for OL.

The Ruby ReXML package is used to load or create XML documents, append nodes, attributes to nodes and process Xpath queries.

The Ferret package (a Ruby port of the Lucene tool) was used to build text indices for fast text queries within text documents (~3,000 paragraphs) and the set of frequent patterns (~40,000). This is very efficient and has a lot of useful features for building text indices (stemming, stop-words removal) or for querying the index (boolean queries, sort filters).

Finally, the DBI package is used to connect Ruby to the MySQL database. I used an ODBC connection because the Native MySQL driver did not work under WinXP.

These tools allowed to build a backend (ruby/server.rb).

All the tools where installed on MacOSX, Linux and WinXP.

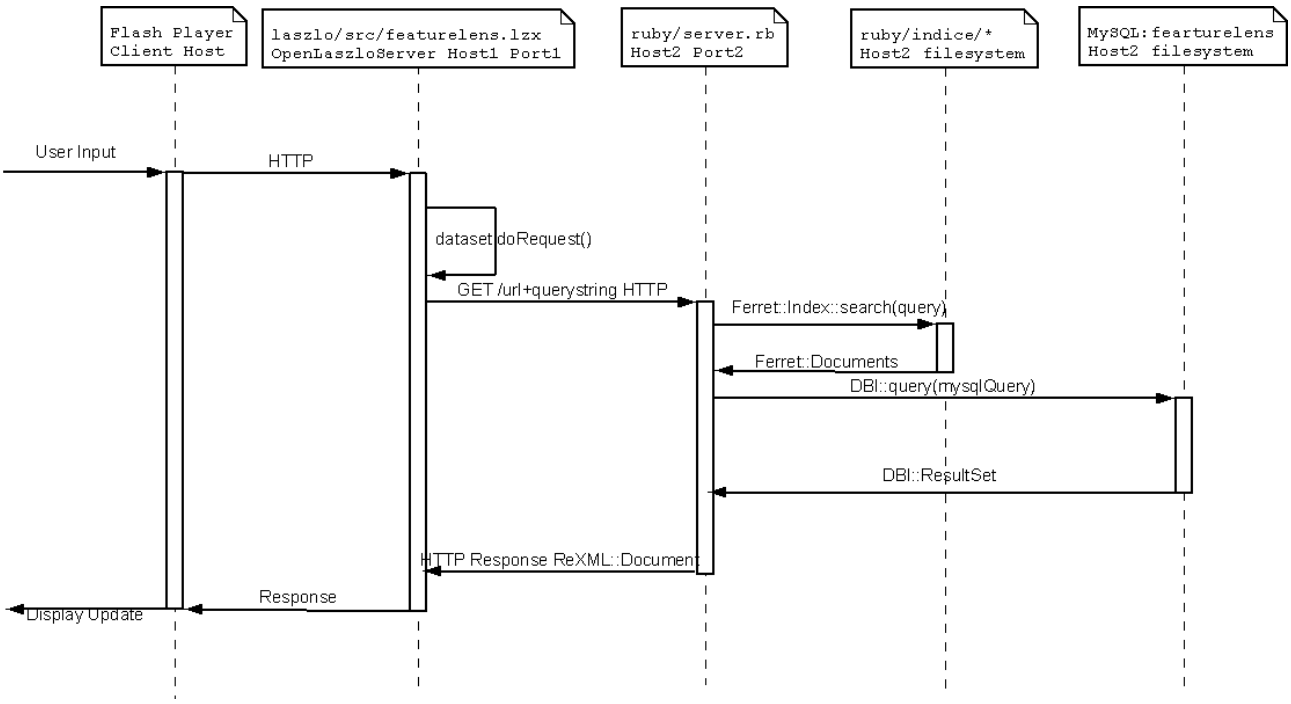

Figure 1: Architecture Overview. Interaction between OpenLaszlo client, Ruby backend and file system.

Data

This section explains how data for a collection of texts is stored. There is a MySQL database and Lucene indices. These data sources are then used by the Ruby backend to feed the OpenLaszlo interface (`ruby/server.rb`).

MySQL tables

The content of a book or collection of texts is stored in a MySQL database along with the frequent text-patterns extracted using D2K. The following tables describe the content of the database. The SQL script to build the empty database structure is stored in '`ruby/datasets/repetitions.sql`'.

chunk

Field	Type
chunk_id	int(10)
n	varchar(10)
type	text
text	text
parent_id	int(10)

Table 1: The "chunk" table contains the text chunks for a given collection. Chunks can be of different types (type=paragraph,line,abstract...). The "n" field is an external reference to the chunk, e.g, paragraph number. The "parent_id" field is a foreign key from the 'collection' table to group chunks by chapter or year.

collection

Field	Type
collection_id	int(10)
n	int(10)
type	text
title	text
parent_id	int(10)

Table 2: The "collection" table defines the hierarchy of a collection (e.g. collection->chapter). The 'type' field tells if the record is for a collection or a chapter. 'title' store the title of the record e.g. book title or chapter title. 'parent_id' defines a recursive relation, it contains the 'collection_id' of the parent of the current record. For example, the records for chapters of a book will store the 'collection_id' of the record that defines the book.

metrics

Field	Type
pattern_id	int(10)
collection_id	int(10)
section_values	text
section_nvalues	text
mean	float(7,5)
n_mean	float(7,5)
max	int(10)
min	int(10)
std_dev	float(7,5)
goodness	float(7,5)
intercept	float(7,5)
slope	float(7,5)
spike_pattern	float(7,5)
sink_pattern	float(7,5)
drop_pattern	float(7,5)
float_pattern	float(7,5)
czeros	int(10)

Table 3: The "metrics" table stores the frequency values per section for each pattern. The "section_values" stores the distribution as a coma-separated list of frequencies. The other fields store analysis values about the current distribution of frequency (mean value, std deviation, linear regression, min, max) as well as trends.

pattern

Field	Type
pattern_id	int(10)
size	int(10)
text	text
support	int(10)
delete	tinyint(1)
collection_id	int(10)

Table 4: The 'pattern' table stores frequent text patterns extracted by the text-mining algorithms. The 'text' field stores the text of the single word or pattern of 3-grams. The size is the number of 3-grams in the pattern (0 for single words). 'support' is the number of chunks that contain this pattern. 'collection_id' tells to which collection this pattern belongs (in case several different books or collections are stored in the database).

section_metrics

Field	Type
pattern_id	int(10)
collection_id	int(10)
section_id	int(10)
section_values	text
section_nvalues	text
mean	float(7,5)
n_mean	float(7,5)
max	int(10)
min	int(10)
std_dev	float(7,5)
goodness	float(7,5)
intercept	float(7,5)
slope	float(7,5)
spike_pattern	float(7,5)
sink_pattern	float(7,5)
drop_pattern	float(7,5)
float_pattern	float(7,5)
czeros	int(10)

Table 5: The 'section_metrics' stores distribution of frequency for each pattern in each section. It has a multiple key 'pattern_id' + 'section_id'. The meaning of the other fields is the same as for the 'section' table.

Ferret (Lucene) indices

For full-text queries and text highlighting, two different indices are built in the '`ruby/indices/`' directory.

The '`ruby/indices/text<collection_id>'` directory contains the search index for all the chunks in the collection.
The '`ruby/indices/pattern<collection_id>`' directory contains the search index for all the patterns of 3-gram in the collection.

Installation instructions

Required software

Here is the list of components used to develop the current FeatureLens demo and preprocess data

- Ruby 1.8.5 with the WebRick package (Web server), ReXML package (XML processing), DBI package (MySQL link), Ferret package (Lucene port in Ruby for fast text indexing and full-text queries). Data import into MySQL and Lucene indices and backend WebServices.
- MySQL (using the EasyPHP 1.8.0.1 distribution) : data storage.
- OpenLaszlo 3.3.3 : client user-interface.
- D2K 4.2.1 : text-mining algorithms for frequent words and patterns of n-grams (D2K/words.itn and D2K/patterns3grams.itn)

Installation steps

Once the required software is installed, do the following steps in that order :

- Create a MySQL database with name '`featurelens`' (and a ODBC link (WinXP))
- Change the DB name in `ruby/datasets/repetitions.sql` and import the database structure stored in this file.
- Import the collections using the `ruby/filter_import_analyze.rb` script
  Check that the MySQL tables are populated and the `ruby/indices` directory contains `text<collection_id>` and `pattern<collection_id>` directories.
- Start OpenLaszlo server
- Start the ruby backend

  ```
  $> cd ruby/
  $> ruby server.rb
  ```

- Open the OLaszlo client application by pointing your web browser to the URL serving `laszlo/src/featurelens.lzx`

Howtos

How to install the ruby backend ?

Installing the ruby backend means loading a collection of text as well as the precomputed patterns in the database and in the indices. Each collection is stored as a XML document in TEI form in the '`ruby/dataset/collection_name`' directory. In the same directory there is a '`pattern_filtered.xml`' file that contain the list of frequent patterns of 3-grams for that collection and a '`words.txt`' file that contains frequent single words for the collection.

To install a collection, use the '`ruby/filter_import_analyze.rb`' script from the 'ruby/' directory.

```
Usage : $> ruby import_filter_analyze.rb
                <dataset/collection/teiFile.xml>
                <dataset/collection/pattern_filtered.xml>
                <dataset/collection/words.txt>
```

This script will take from 5 to 120 minutes depending on the size of the collection to install and the number of patterns.

How to install the OpenLaszlo application ?

Once OpenLaszlo is installed, you will be able to install the featureLens demo

- Go to `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\my-apps\`
- Create a `featurelens` directory.
- Copy the `laszlo/src/` directory in the `featurelens` directory.

You now need to import necessary files for the OpenLaszlo server to be able to interpreted your .lzx files.

- Copy `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\lps` to `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\my-apps\featurelens\`
- Copy `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\WEB-INF\lib` to `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\my-apps\featurelens\WEB-INF\`
- Copy `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\WEB-INF\lps` to `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\my-apps\featurelens\WEB-INF\`
- Copy `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\WEB-INF\web.xml` to `\Path\To\OpenLaszlo Server 3.3.3\Server\lps-3.3.3\my-apps\featurelens\WEB-INF\`

The demo should then appear using the following URL :

```
http://127.0.0.1:8080/lps-3.3.3/myapps/featurelens/src/featurelens.lzx
```

To consult the OpenLaszlo documentation, start the OpenLaszlo server and point your browser to :

```
http://127.0.0.1:8080/lps-3.3.3/laszlo-explorer/coverpages/documentation_cover.html
```

See the following for deployment details :

```
http://127.0.0.1:8080/lps-3.3.3/docs/deploy/deployers-guide.html
```

How to change the backend URL in OpenLaszlo ?

In `laszlo\src\settingsWindow.lzx`, change the value of the following attribute to the URL and port of the server where the backend is running :

```
<attribute name="backendURL" type="string" value="http://127.0.0.1:4040/"/>
```

How to change the MySQL access in the backend ?

The connection to the MySQL database is a global variable in ruby/server.rb.
On windows platforms it should look like. Make sure that the MySQL ODBC driver is installed an configured to use the right database. Change « link_name », « mysql_user », « mysql_password » to appropriate values :

```
$dbh = DBI.connect("DBI:ODBC:link_name", "mysql_user", "mysql_password")
```

On Linux platforms and MacOSX, change « db_name », « mysql_user », « mysql_password » to appropriate values :

```
$dbh = DBI.connect("DBI:Mysql:db_name:localhost", "mysql_user", "mysql_password")
```

How to change the backend port number ?

The line that sets the port number is in ruby/server.rb at the end of the file. Change 4040 to a different value :

```
server = WEBrick::HTTPServer.new(:Port => 4040)
```

How to install a new collection ?

To install a new collection, you should use the '`ruby/filter_import_analyze.rb`' script. But prior to that step, you have to precompute the frequent text patterns using D2K with the '`D2K/`' itineraries.

You first need to put your collection of text in a valid TEI XML form. See existing examples in '`ruby/datasets/`'.

Launch D2K and load the '`D2K/words.itn`' itinerary to compute frequent words. Figure 2 shows the itinerary. To change the input XML document, click on the '`InputFileName`' module and change the value of the path to point to your TEI XML document. If you don't want to exclude '`stop words`', directly connect '`FilterNonTokenWords`' to '`Ngram`'. Check that '`Ngram`' has its size parameter set to '1' (compute single words, or 1-grams). Set the '`FeatureSupportFilter`' parameters to filter out too frequent patterns. Change the path in '`InputFileName1`' to indicate where the coma-separated list of words should be stored.

Launch D2K and load the '`D2K/patterns3grams.itn`' itinerary to compute frequent patterns

of 3-grams. Figure 3 shows the itinerary. To change the input XML document, click on the '`InputFileName`' module and change the value of the path to point to your TEI XML document. Set the '`FeatureSupportFilter`' parameters to set the minimum support for a pattern to be considered 'frequent'. The result will be stored in a XML file in '`path_to_D2K/D2KToolkit/pattern.xml`'. It is not possible to change this path in the current version of the itinerary.

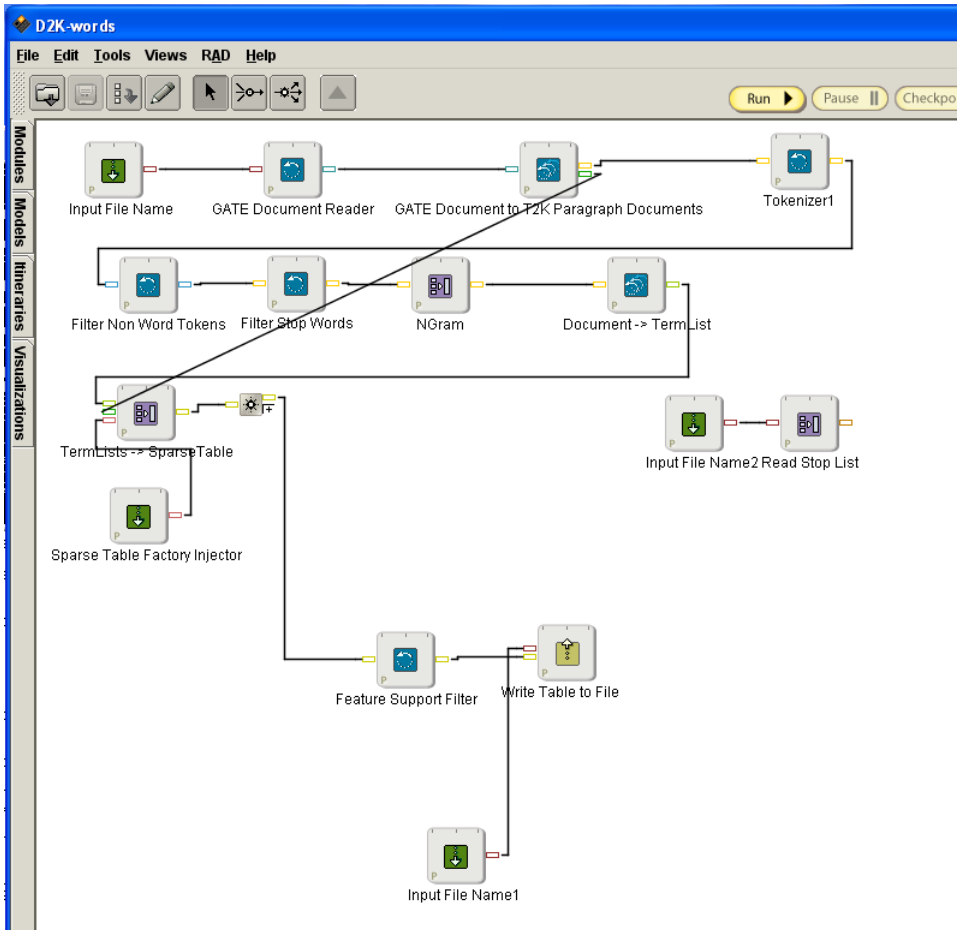

Figure 2: D2K 'words.itn'

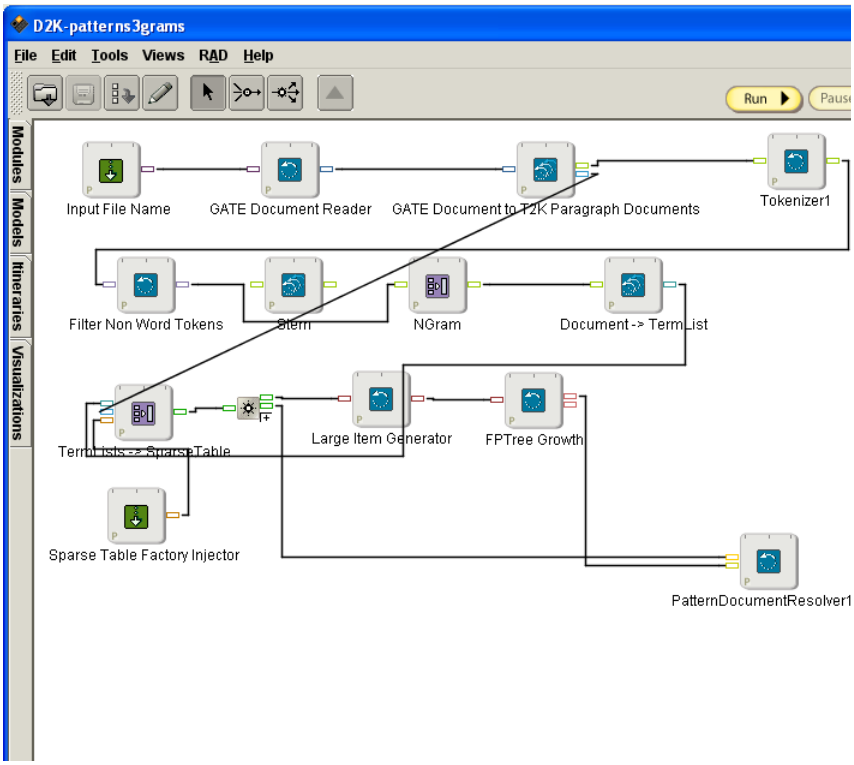

Figure 3: D2K patterns3grams.itn