Lab 6: Information Retrieval

Due date: Monday, November 22, beginning of the lab period.

Note: You will receive **Lab 7** assignment on **November 17** (and it will be due November 29). Because of this, it is best to have as much of Lab 6 completed on or by **November 17-18** as possible. The weekend is provided to give you more flexibility.

Overview

In this assignment you will build a simple Information Retrieval system for a small collection of documents. We will use `Jester` jokes as a test dataset, but other text documents (stored as text files) can be used within your system. Your system will be an interactive command-line program which takes as input user commands and produces specified output.

Assignment Preparation

This is a pair programming assignment. Each student teams up with a partner. Each team submits only one copy of the assignment deliverables.

Data

You will be using *joke text collection* from the **Jester** project, run by Professor Ken Goldberg at UC Berkeley. **Jester**[**?**] is an on-line joke recommender system available at

http://shadow.ieor.berkeley.edu/humor/

Disclaimer. **Jester** has a database consisting of 100 jokes. The jokes are shown to the user, and the user's reaction to them is measured on a continuous scale. **Please be aware** that the jokes available through `Jester` may contain some examples that you personally will find **tasteless**, **sexist**, **inapropriate** or just plain **stupid**. It is not my intent to offend anyone's sensibilities by using this dataset.

The Dataset

I have created a page on the course wiki to post links to the files you will need for this assignment:

http://wiki.csc.calpoly.edu/csc466-2010/wiki/jester

The full **Jester** jokes data is available as a collection of `.html` files at

http://eigentaste.berkeley.edu/jester-data/jester-joke-texts.zip

It is also available as a single `XML` file `Jokes.xml` from the wiki page above. The structure of the file is:

```
<jokes>
  <joke>
     text of joke goes here ...
  </joke>
  <joke> ... </joke>
  ...
</jokes>
```

The jokes are not numbered internally in the XML file, but their order corresponds to the joke ids in the used in the program.

Lab Assignment

You will build a simple text-based command-line Information Retrieval system for the Jester jokes. Your program will give users a prompt, accept commands, provide output. The detailed specifications are below.

System Overview

R0. The information retrieval system you will create will work in an interactive mode. Name the main Java program `ir.java`.

R1. When the program is started, it shall provide the user with a prompt and prepare to accept commands, as shown below:

```
$ java ir
```

```
IR System Version 1.0

IR>
```

User enters one of the commands described in requirement **R2.** The system executes the command, produces necessary output, and, unless it was a `QUIT` command, returns back to the prompt and waiting for user input.

R2. Your system shall recognize, correctly parse and process the following list of commands:

Command	Meaning
`READ <file.xml>`	Read text document(s) from an XML file
`READ <file.txt>`	Read text document from a text file
`READ LIST <file>`	Read text documents from files listed in `<file>`
`LIST`	Output the list of documents available in the system
`CLEAR`	Remove all documents from the system
`PRINT <DocID>`	Print the content of the document to screen
`SHOW <DocID>`	Show the internal representation of the document
`SIM <Doc1> <Doc2>`	Compute and output the similarity between two documents
`SEARCH DOC <DocId>`	Search for documents similar to given document
`SEARCH "string"`	Search for documents relevant to the query string
`QUIT`	Quit the proram

Your program must accept the `ALLCAPS` command syntax. You may choose to make all commands case insensitive.

R3. Your system shall support persistent storage of information. This means that your system shall maintain the list of documents that have been processed and the actual data structures in which the vectors of keyword weights are stored and *remember them* between the runs of the program. The data read into your program during a session shall NOT be abandoned upon encountering the `QUIT` command. When your program is restarted, it shall be able to access the information about the documents already processed.

R4. The three commands to read data have the following syntax:

```
READ <file>.xml

READ <file>.txt

READ LIST <file>
```

The first command is used to parse the contents of an XML file, whose structure matches the structure of `Jokes.xml` file. It is invoked if the file-name ends in `.xml`. The XML file may contain one document or multiple documents.

The second comand is used to read in the contents of a single document from a plain text file containing it. The command is invoked when the file name ends with `.txt`.

The third command reads in a file that contains the list of text file names. It then proceeds to read in, one-by-one the documents stored in the files with those names.

R5. The reading of a document procedure is described here. Each document read into the system shall go through the following set of steps:

- **Parsing and tokenization.** Document split into tokens, each representing a single word in the document (plus, possibly some tokens for sentence delimiters and so on).
- **Stopword removal.** Stopwords need to be removed. For discussion on the list of stopwords, please consult the Appendix.
- **Stemming.** Use Porter's algorithm. Feel free to use the open-source implementations available. See Appendix for more information on it.
- **Vocabulary maintenance.** The *vocabulary* or *corpus* of your document collection is the list of all terms (keywords) found in all the documents in the collection. Because your are constructing the collection one document at a time, the vocabulary will also be built in steps. Each time a new document is processed, your program shall add any new keywords found in it to the vocabulary.

  Your program shall also maintain the *document frequency* (and *idf*, the inverse document frequency) in for each term in the vocabulary.
- **Vector Space representation.** The document, represented, after the stemming step by a bag of words shall be converted into a vector of keyword weights. The exact formulae to use to compute the keyword weights is left up to you, but you **must** use *term frequency* (tf) and *inverse document frequency* (*idf*) as in your computation. You can use *tf-idf* or any variation of it you know/learn.

  The vectors of keyword weights you construct **shall be stored as sparse vectors**. Your program shall maintain a copy of all vectors in persistent storage and be able to read them from persistent storage when necessary.

R6. The `LIST` command outputs the list of document IDs stored in the current document collection. The Document Ids are printed one per line of the output.

The Document Id of a document read from a text file is the name of the file. The document Id of a document read from an XML file containing multiple documents is the name of the XML file, followed by the ordinal, representing the position of the document in the XML file. You can choose to keep the `.txt` and `.xml` extensions, or remove them. Document IDs must be unique, however, the same document can show up under multiple ids (if, e.g., it was found in two text files with different names).

For example, let XML file `Test.xml` contain four documents in it. Then, the document Ids for these documents can be `Test1`, `Test2`, `Test3` and `Test4`, or `Test.xml-1`, `Test.xml-2`, `Test.xml-3` and `Test.xml-4`. You can use padding: e.g., if more than 10 documents are present, you can name them `Test01` of `Test.xml-03`.

R7. The `CLEAR` command takes no parameters and has no visible output. Its result is the removal of all memory-resident and persistent information about the documents in the document collection. Essentially, the `CLEAR` command removes all the documents from the document collection, or deletes the document collection itself.

R8. The `PRINT` command has the following syntax:

```
PRINT <DocID>
```

As the result of this command the full text of the document with the given document ID is printed out. The text should be printed verbatim as it was found in the original file (you can parse out XML, if the original of the document was in an XML file). Feel free to simply open the appropriate file (associated with the document) and print its contents verbatim (or parse out the appropriate XML element, and print its contents verbatim).

R9. The `SHOW` command has the following syntax:

```
SHOW <DocID>
```

This command results in the system printing the keyword weights vector associated with the document whose ID is listed in the command. The format of the output is left up to each team. The output shall contain the keyword/keyword stem/keyword ID and the keyword weight for all keywords that have a non-zero weight in the document vector.

R10. The `SIM` command has the following syntax

```
SIM <Doc1> <Doc2>
```

Given this command, the system performs a similarity computation on keyword weight vectors associated with the documents `<Doc1>` and `<Doc2>`. The resulting similarity score is reported to the user.

You can choose any way of computing similarity discussed in class/found in textbook/found in lecture notes/discovered by you independently. If you implement more than one similarity computation in your system, select a default one to use when the `SIM` command is given in the form above.

You can then modify the `SIM` command to have the syntax

```
SIM <Method> <Doc1> <Doc2>
```

and/or

```
SIM ALL <Doc1> <Doc2>
```

In the former case, you can pick the keywords to describe each method. E.g., if your system implements TF-IDF and Okapi similarity computations, you can use `TFIDF` and `OKAPI` as the values of the `<Method>` parameter with obvious effects.

`SIM ALL` command shall result in all similarity computations performed and reported.

R11. The `SEARCH` command has two variants:

```
SEARCH DOC <DocId>
```

and

```
SEARCH "string"
```

The first variant shall cause your system to retrieve all documents similar to the document with the given document ID. The second variant shall cause your system to retrieve all documents deemed relevant to the input query. The query should be enclosed in double quote symbols. The query shall be treated as a text representing a document and shall subjected to parsing, stopword removal and stemming just like the documents are. You may choose a way of determining the vector of term weights for a query that differs from what you do for a document when executing a `READ` command.

The output shall include *document IDs and similarity scores* for *all documents with a non-zero similarity* with the query (be it a document or a text query) *reported in descending order of the similarity score.*

You can also implement the limiting version of these two commands:

```
SEARCH DOC <DocId> <Number>
```

```
SEARCH "string" <Number>
```

In either case, `<Number>` is the number of relevant matches to show. E.g.,

```
SEARCH "Cal Poly basketball" 2
```

shall report the top two matches for the query `"Cal Poly basketball"`.

R12. The `QUIT` command causes the system to quit.

R13. The system shall have basic error-handling capabilities. In particular, the system shall never throw an unhandled exception and stop working as a result of an error in the command. Upon receiving a command, the system shall check for validity of the instruction and the parameters. If the parameters are incorrect (e.g., file not found, wrong document Id, incorrectly formatted string query and so on), an error message shall be reported by the system, and the system shall output the prompt and wait for the next command from the user.

Deliverables and submission instructions

This lab has only electronic deliverables. Submit the following:

- Source code for the IR system.
- `README` file describing the following:
  - names of all team members;
  - specifics of implementation, in particular, what weighting method(s) and similarity score(s) you used.
  - any compile/runtime instructions for the TA;
  - any extra credit claims. The extra credit is given for implementing more than one retrieval method. Any instructions related to extensions of the command list to realize extra credit in the system shall be included as well.

Submit all electronic deliverables as a single zip of gzipped tar archive (`lab06.zip` or `lab06.tar.gz`). Use the following command

```
$ handin dekhtyar-grader lab06 lab06.<ext>
```

Appendix: Stopword removal

A wide range of English stopword lists can be found on-line. The Lab 6 data page,

http://users.csc.calpoly.edu/∼dekhtyar/466-Fall2010/

contains links to a number of such resources and a collection of text files with the stopword lists found there.

You should feel free to either use any of the provided stopword lists, use any other stopword lists you discover, build your own list from scratch, or edit an existing list.

Appendix: Stemming and Porter Algorithm

The Lab 6 data page contains links to the official page for Porter Stemming Algorithm, the java code for it and a copy of the paper describing the algorithm. Porter algorithm implementations in other languages are found on the official page.

References

[1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2), 133-151. July 2001.

[2] Ken Goldberg, Anonymous Ratings Data from the Jester Online Joke Recommender System, http://www.ieor.berkeley.edu/~goldberg/jester-data/.