Spring 2009	**CSC 466:** Knowledge Discovery from Data	Alexander Dekhtyar

Lab 6:

Due date: Tueseday, June 1, midnight.

Note: You will receive **Lab 7** assignment on **May 28** (and it will be due June 4). Because of this, it is best to have Lab 6 completed on or by **May 28**. The extra 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 (`Jester` jokes). Your system will have two main use cases: (a) given a joke, find jokes that are similar to it (textually, topically, etc. . .), and (b) given a user information need (search query), find all jokes relevant to it).

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**[1] is an on-line joke recommender system available at

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

Disclaimer. Please read this note before proceeding! 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**. While each team will work with the texts of the jokes as well as with the numeric data, please be aware, that ***it is not my intent to offend anyone's sensibilities*** (and neither is it the intent of the authors of the dataset and **Jester**).

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/csc466-2009/wiki/Lab6

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 Information Retrieval system for the Jester jokes. In doing so, you will be extending the source code provided to you. The overall system is similar in its look-an-feel to the recommendation system you were implementing in Lab 5. The code provided to you shares a lot of features with Lab 5 code.

System Overview

Figures 2 and **??** show the GUI of the IR system you are building. As the Lab 5 recommender system, the IR system GUI contains two list boxes and two read-only text boxes. The list box on the left contains the complete list of jokes. When a joke is selected from this list, its text is displayed in the text box immediately below it.

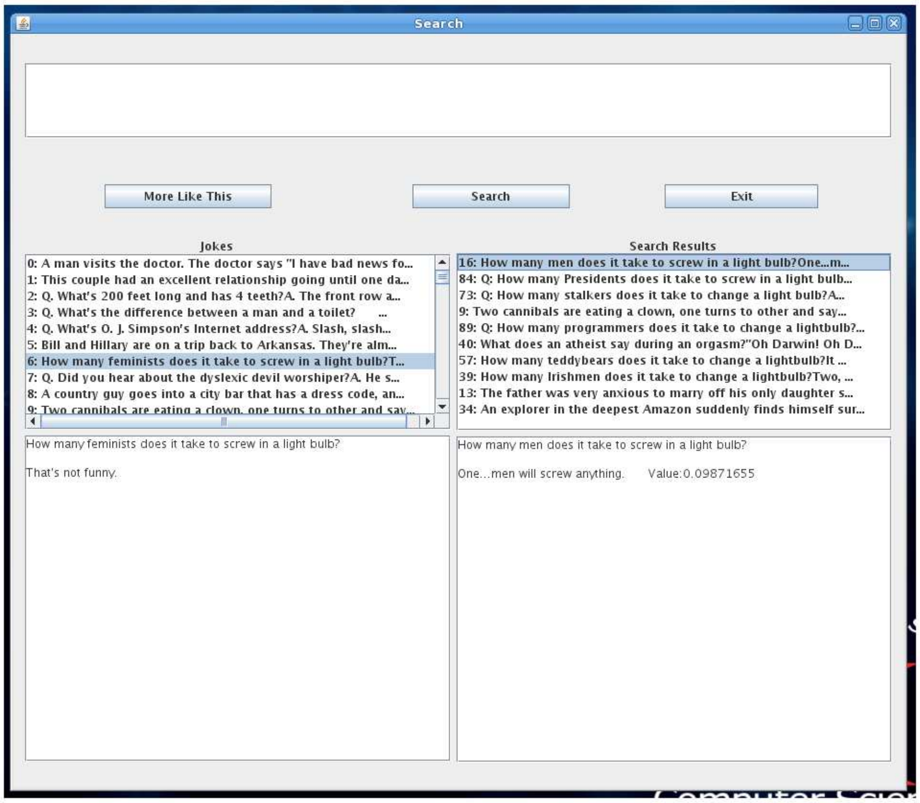

Figure 1: The GUI of the joke IR system. `"More Like This"` button pressed.

There are three buttons on the GUI: `"More Like This"`, `"Search"` and `"Exit"`. Additionally, there is a *editable text field* at the top of the GUI, which is to be used to enter search queries.

Use Cases. The IR system shall behave as follows:

UC1. Search function. The user types a search query (one or more words) into the *search text box* at the top of the GUI. The user then presses the `"Search"` button.

The IR system shall search the list of jokes for all jokes relevant to the user query. The list of jokes shall be displayed in the list box on the right of the GUI in the order of decreasing relevance. When a user selects a joke from the provided list its text will be displayed in the text box below the list. (the latter feature is already available in the source code provided to you).

The IR system shall display either 10 most relevant jokes, or, if fewer than 10 jokes were retrieved, all retrieved jokes.

UC2. "More like this" function. The user selects a joke from the list on

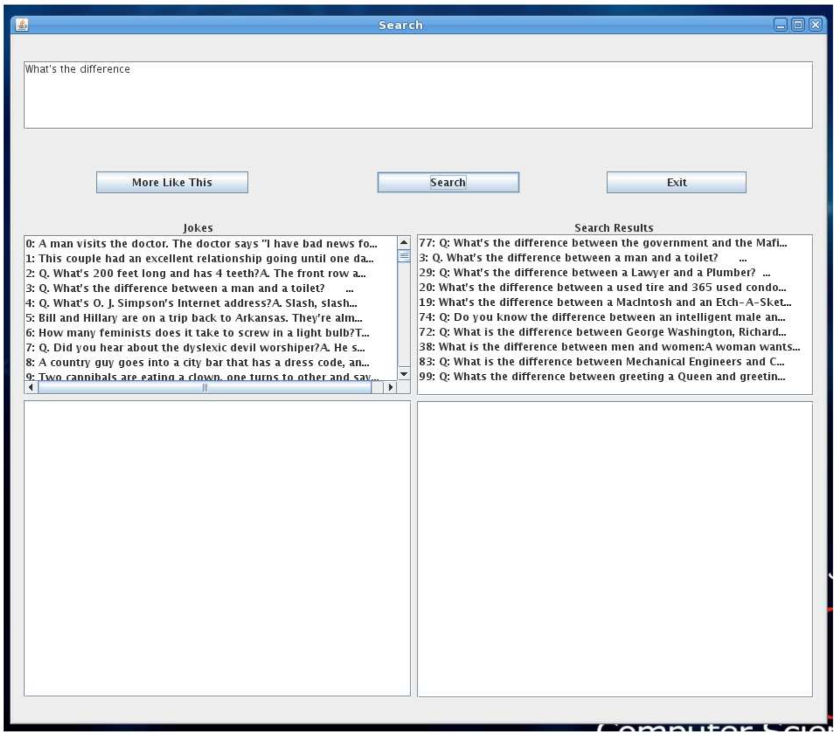

Figure 2: The GUI of the joke IR system. `"Search"` button pressed.

the left side of the GUI (the text of the joke is displayed in the text box below). The user pressesds the `"More Like This"` button.

The IR system shall search the list of jokes for the jokes that are similar to the joke selected by the user. The list of jokes shall be displayed in the list box on the right of the GUI in the order of decreasing relevance. When a user selects a joke from the provided list its text will be displayed in the text box below the list. (the latter feature is already available in the source code provided to you).

The IR system shall display either 10 most relevant jokes, or, if fewer than 10 jokes were retrieved, all retrieved jokes.

UC3 Exit. When the user presses the `"Exit"` button the system quits (this feature is implemented in the code you receive).

System Architecture

Just like your Lab 5 recommendation system, the IR system will consist of two parts:(i) preprocessing and (ii) query engine. Unlike Lab 5, for Lab 6, you shall incorporate both parts into a single program, `search.java`.

Preprocessing

Note: Here and below, each individual joke will be referred to as *document* and all jokes will be referred to as `document collection`.

The preprocessing component of your IR system shall be responsible for the following activities:

1. Parsing of the `Jokes.xml` file.
2. Removing all stopwords from the documents.
3. Stemming the terms in the documents.
4. Creation of the keyword weight vectors for each document.

The preprocessing component is invoked upon startup of the system. Each activity is performed exactly once during the runtime of the `search` program. You can find the stub of the preprocessing code in the `search` class constructor method. To build the preprocessor/indexer, you need to create and modify a number of methods inside the `search` class, and class these methods (in proper order) from the `search` class constructor. You also may need to declare appropriate data structures to hold the results of your processing (one way to do so is to declare them as members of the `search` class).

Parsing involves reading the XML document from the input file, breaking it into individual jokes and extracting the text of each joke, and breaking this text into a series of *tokens* (words, punctuation, etc). Part of this is performed by the `search.PopulateJokes()` method, which is provided to you in its entirety. This method populates the list `search.myJokeList` of jokes. This list is used to display jokes. It also serves as the input to the tokenizer.

You are responsible for developing a method inside the `search` class that tokenizes (splits into words, punctuation, etc) individual jokes and for creating the appropriate data structure to store tokenized representations of the jokes (if you believe you need it).

Stopword removal involves searching the text of each joke for **stopwords** and removing the stopwords from the text representing the jokes. The list of stopwords is initialized in the

```
private ArrayList<String> populateStop()
```

method of the `search` class in the code. It is then stored in the

```
ArrayList<String> stopwords
```

member of the `search` class. Feel free to examine the list of stopwords (found in the code of the `populateStop()` method) and to update it to your needs.

The actual stopword removal shall be performed by the

```
private void removeStopWords()
```

method, which is available *as a stub* in the provided code.

You are also responsible for creating the data structures to store the bag-of-words representations of the jokes after stop words are removed (this can be done with the same data structure as the one used in the parsing step, or with a new one).

Stemming involves replacing each word remaining after stopword removal with *its stem*. This is done using Porter's algorithm.

The open-source implementation of Porter's algorithm is included in the code provided to you in the file `Stemmer.java`.[1]

It is wrapped into the `search.stem()` method. The method is currently stubbed, but contains some commented out code to make it easier for you to set it up properly.

You are responsible for declaring any data structures you need to complete the stemming.

Creation of the document vectors is the last step of the preprocessing stage. You are expected to implement the **vector space retrieval model** with a *variant* of **TF-IDF term weighting** (or any other term weighting discussed in class) to convert the lists of stems representing each joke into vectors of term weights. The computed vectors are stored for further use during the work of the query engine.

You are responsible for both declaring the necessary data structures to store the vectors, and for implementing the appropriate method/methods that process the stemmed joke bags of words and generate the document vectors.

Query Engine

The query engine of the IR system is represented in the code by two methods designed to produce reactions for use cases **UC1** and **UC2**.

1. **Search Function.** When the `"Search"` button is pressed the following method is invoked.

   ```
   private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
   ```

[1] The code is available from http://tartarus.org/~martin/PorterStemmer/.

The method is stubbed in the provided code. The following actions shall be performed:

(a) Read the contents of the *search text field*.

(b) Parse/remove stopwords/stem the search string.

(c) Build a query vector for the search string.

(d) Find the similarity between each document (joke) in the collection and the search string.

(e) Report the 10 (or fewer) jokes with the highest similarity scores.

2. **"More Like This" Function.** When the `"More Like This"` button is pressed the following method is invoked.

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
```

The method is stubbed in the provided code. The following actions shall be performed:

(a) Determine which joke was last selected in the list box on the left.

(b) Find the similarity between each document (joke) in the collection and the selected joke.

(c) Report the 10 (or fewer) jokes with the highest similarity scores.

Available code. To get the code provided to you, download the `Lab6-Student.zip` from

http://wiki.csc.calpoly.edu/csc466-2009/wiki/Lab5

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, the version of the TF-IDF score you used.
  - any compile/runtime instructions for the TA;
  - any extra credit claims;

Submit all electronic deliverables as a single zip of gzipped tar archive (`lab05.zip` or `lab05.tar.gz`). Use the following command

```
$ handin dekhtyar-grader lab05 lab05.<ext>
```

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/.