Web Search, Télécom ParisTech

Inverted Index

Pierre Senellart (pierre.senellart@telecom-paristech.fr)

16 March 2010

The purpose of this lab session is to build an inverted index out of a collection of text documents, so that keyword queries (both conjunctive and disjunctive) can be processed over the corpus.

Dataset

The dataset we use for these labs is the content of the Simple English Wikipedia (`http://simple.wikipedia.org/` (a simpler and smaller encyclopedia than the regular English Wikipedia). The whole content of the encyclopedia can be downloaded from `http://download.wikimedia.org/`, but we provide here for these labs a preprocessed version of the content, that you can find either on the course website, or as `~senellar/inf396/text` on the computers used during the labs. The format of this file is as follows: The first line is the title of the first article, while the following lines (up to the first blank line) form the content of this article, in Wiki format. The second article comes after the next blank line, and so on. There are 62298 articles in total.

1 Inverted Index

1. Create a class `InvertedIndex` that will be used to store an in-memory version of an inverted index (that is, for each token occurring in the collection, this token and the set of all documents that contain this token, along with the weight of the token in this document, ordered by decreasing weight). You can use the collections from the Java API, especially `java.util.TreeMap` and `java.util.TreeSet`. To define a `TreeSet<A>` or a `TreeMap<A,B>`, class `A` must implements the `Comparable` interface.
2. Add a constructor to this class that takes for argument a filename (containing the collection, in the format described above) and a list of stop words (one per line) and creates the in-memory index from the collection. The following rules can be used to preprocess the text:
   - Tokens are defined as succession of either alphabetical characters or digits (e.g., “Hello”, “123”, “a22”; “vice-versa” or ”it’s” are decomposed into two tokens). One can use the following line to compile a `java.util.regex.Pattern` that will match these tokens:
     ```
     Pattern alphaDigit = Pattern.compile("[\\p{L}\\p{N}]+");
     ```
     All tokens should be made lower case.
   - The Porter stemming can be used. An implementation of the Porter stemmer in various programming languages is available at `http://tartarus.org/~martin/PorterStemmer/`. The Java implementation is used as follows:
     ```
     Stemmer stemmer=new Stemmer();
     stemmer.add(word.toCharArray(),word.length());
     stemmer.stem();
     String token=stemmer.toString();
     ```

- Stop words will be removed from the index if they are contained in the provided file (found either on the Web site or at `~senellar/inf396/stop_words.txt`).

A file can be read line by line in Java using a `java.io.BufferedReader` constructed as follows:

```
BufferedReader r=new BufferedReader(new FileReader(filename));
```

Use standard tf-idf weighting.

3. Test your index construction. In case you encounter memory limitation issues, just consider only the n first articles from the dataset (say, $n = 15{,}000$). Use debugging facilities or logging to have a look at the tokens stored in your inverted index and check if everything goes right.

4. Add a method `disjunctiveQuery` that takes as argument an array of String representing a disjunctive query, and run this query with the help of the inverted index to get the list of all matching results, along with their weight. Addition can be used to combine weights. `disjunctiveQuery` returns the whole set of results (ordered by decreasing total weight), so it is not necessary to use advanced top-k evaluation methods.

5. Same with `conjunctiveQuery`.

6. Test these methods. You can use for instance the query "president france 2007". Are the results relevant?

2 To go further

1. Use Java serialization facilities to store a copy of the inverted index on file, and reload it directly without having to construct it.

2. Use this inverted index jointly with the crawler you developed yesterday, to index the textual content of Web pages.

3. Build the `text` file yourself from the XML dump of the Simple English Wikipedia that can be downloaded from `http://download.wikimedia.org/`.