Technologies du Web, Master COMASIC

TP: Moteur de recherche

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

15 février 2013

Le but de ce TP est de construire un index inversé à partir d'un corpus de documents textuels, pour pouvoir répondre à des requêtes par mots-clefs (conjonctives), et de concevoir une interface Web pour interroger cet index.

Le résultat du TP est à rendre par email à pierre.senellart@telecom-paristech.fr au plus tard le dimanche 17 février 2013 à 23 :59 heure de Paris, sous la forme d'une archive ZIP contenant l'ensemble du code source. Les TP sont à faire individuellement et le résultat de l'évaluation de ce TP servira d'évaluation du module dans le cadre du Master COMASIC. Même s'il est possible de poursuivre le TP au-delà de la séance (et jusqu'au dimanche soir), ce n'est pas requis ; un travail sérieux pendant les trois heures de la séance de TP suffit à obtenir une note validant le module. Tout retard de rendu sera sanctionné de manière très stricte.

Corpus

Le jeu de données que nous utilisons pour ce TP est le contenu de Wikipedia *Simple English* http://simple.wikipedia.org/ (une encyclopédie plus simple et plus petite que Wikipedia English). L'ensemble du contenu de cet encyclopédie peut être téléchargé depuis http://dumps.wikimedia.org/ mais nous fournissons pour ce TP une version prétraitée, qui peut être récupérée soit sur le site Web du cours http://pierre.senellart.com/enseignement/2012-2013/comasic/ soit comme ~senellar/comasic/text sur les machines de la salle de TP. Le format de ce fichier est comme suit : la première ligne est le titre du premier article, tandis que les lignes suivantes (jusqu'à la première ligne vide) contiennent le contenu de l'article, en texte brut. Le deuxième article vient après la première ligne vide, et ainsi de suite. Il y a 50 441 articles au total.

1 Index inversé

1. Créer une classe InvertedIndex qui sera utilisée pour stocker en mémoire vive une version de l'index inversé (c'est-à-dire, pour chaque lexème apparaissant dans le corpus, ce lexème et la liste des documents qui le contiennent, ainsi que le poids de ce lexème dans ce document, par ordre décroissant de poids). On pourra utiliser les collections de l'API Java, en particulier java.util.TreeMap et java.util.TreeSet. Pour définir un TreeSet<A> ou un TreeMap<A,B>, la classe A doit implémenter l'interface Comparable<A>.
2. Ajouter un constructeur à cette classe qui prend en argument un nom de fichier (contenant le corpus, au format décrit ci-dessus) et une liste de *mots vides* (un par ligne) et qui crée l'index en mémoire depuis le corpus. Les règles suivantes peuvent être utilisées pour prétraiter le texte :

- Les lexèmes sont définis comme une succession de caractères alphabétiques ou numériques (p.ex., “Hello”, “123”, “a22” ; “vice-versa” ou ”it’s” sont décomposés comme deux lexèmes). On peut utiliser la ligne suivante pour compiler un `java.util.regex.Pattern` qui correspond à ces lexèmes :
  ```
  Pattern alphaDigit = Pattern.compile("[\\p{L}\\p{N}]+");
  ```
  Tous les lexèmes doivent être mis en minuscules.
- Le lemmatiseur de Porter peut être utilisé. Une implémentation de celui-ci dans de nombreux langages de programmation est disponible à l’URL `http://tartarus.org/~martin/PorterStemmer/`. L’implémentation Java s’utilise comme suit :
  ```
  Stemmer stemmer=new Stemmer();
  stemmer.add(word.toCharArray(),word.length());
  stemmer.stem();
  String token=stemmer.toString();
  ```
- Les mots vides seront supprimés de l’index s’ils sont contenus dans le fichier fourni (sur le site Web ou sur `~senellar/comasic/stop_words.txt`).

On peut lire un fichier ligne-à-ligne en Java avec un `java.io.BufferedReader` que l’on construit comme suit :

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

Utiliser une pondération tf-idf standard.

3. Tester la construction de votre index. Si vous rencontrez des problèmes de temps ou de mémoires, considérez simplement les n premier articles du corpus (par exemple, $n = 15\,000$). Utilisez des fonctionnalités de débogage ou de journalisation pour vérifier que les lexèmes stockés dans l’index inversé sont corrects.
4. Ajouter une méthode `conjunctiveQuery` qui prend en argument un tableau de `String` représentant une requête conjonctive, et qui évalue cette requête à l’aide de l’index inversé pour récupérer l’ensemble de tous les résultats. On pourra utiliser l’addition pour combiner les poids. `conjunctiveQuery` retourne l’ensemble complet des résultats (ordonnés par poids décroissant), donc il n’est pas nécessaire d’utiliser des méthodes avancées d’évaluation top-k.
5. Tester, par exemple avec la requête “president france 2007”. Les résultats sont-ils pertinents ?

2 Interface Web

1. Suivre le tutoriel `http://wiki.eclipse.org/Jetty/Tutorial/Jetty_HelloWorld` pour implémenter un serveur Web minimal en Java.
2. Utiliser ce serveur Web pour ajouter une interface Web simple à l’index inversé. Il doit être possible d’entrer une requête dans un formulaire, et en cliquant sur « Ok », d’afficher la liste des k premiers résultats.
3. S’il vous reste du temps, ajouter des styles CSS, une autocomplétion en AJAX des mots entrés (en utilisant les requêtes passées, ou les mots existant dans l’index), des fonctionnalités de pagination pour afficher l’ensemble des résultats page par page, ou toute autre fonctionnalité qui vous semblera utile.