

TP de Text Mining - Création d'un moteur de recherche

Moteur de recherche

Introduction

Le but de ce TP est d'implémenter une version simplifiée d'un moteur de recherche de documents. Le choix du langage de programmation est libre mais l'architecture générale du moteur devra être respectée. Les étudiants souhaitant avoir des points bonus au projet pourront envoyer leur code à sylvain.utard@gmail.com avec la balise "[TEXTMINING]".

Pour le TP, le dataset <http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz> pourra être utilisé.

Les connecteurs

Leur rôle est d'extraire le texte des documents quelque soit leur format.

Pour commencer ce TP, seul un type de connecteur sera implémenté: le connecteur file system. Son rôle est de parcourir un répertoire et d'extraire tous les fichiers "lisibles" (texte ou équivalent).

```
class Document
  String text
  String url
end
Document[] fetch(String path, boolean recursive = true)
```

L'analyseur

Son rôle est de transformer le texte brut des documents d'entrées en mots, et d'effectuer des traitements sur ces mots.

```
class TokenizedDocument
  String[] words
  String url
end
TokenizedDocument[] analyze(Document[] documents, TextProcessor[] processors)
```

Par défaut, il faudra au moins gérer la normalisation de chaque mot (minusculation + suppression des accents).

```
class abstract TextProcessor
  String process(String word)
end

class Normalizer < TextProcessor
  String process(String word)
end
```

L'indexeur

Son rôle est de transformer des listes de mots appartenant à un document en des listes inversées de mots associés à des documents.

```
class Posting
  String word
  String[] url
end
Posting[] index(TokenizedDocument[] documents)
```

A partir des listes inversées, créer l'index:

```
class Index
  Map<String, int> urlToDid;
  Map<String, int[]> wordToDids;
end
Index build(Postings[] postings)
```

Il est important que l'index puisse être sauvegardé sur disque:

```
void save(Index index, String path)
```

Le Searcher

Son rôle est de lire les listes inversées afin de répondre à une requête.

Il devra dans un premier temps pouvoir lire un index sauvegardé précédemment sur disque:

```
Index load(String path)
```

Puis effectuer des requêtes dessus:

```
String[] search(String word)
String[] search(AST root) # AND, OR and NOT handling
```

L'indexation incrémental

La création d'un index devient une création d'une "génération" de l'index. Chaque génération dispose de ses propres listes inversées. Le searcher doit pouvoir charger en mémoire plusieurs générations et pouvoir fusionner les résultats à la requête. Si un document appartient à plusieurs générations, c'est sa version la plus récente qui l'emporte. De cette manière, si un document est ajouté plusieurs fois, on ne récupérera que sa version la plus récente.

```
class Generation
  Map<String, int[]> wordToDids;
end
class Index
```

```
    Map<String, int> urlToDid;  
    Generation[] generations  
end
```

Bonus 1: stockage de méta-données

Dans la quasi totalité des cas, un index doit remonter des méta-données en plus des URL des documents qui ont matchés. Ces méta-données sont fournies par le connecteur en plus du texte.

```
class Document  
    String text  
    Map<String, String> metaDatas  
    String url  
end  
  
class Generation  
    Map<String, int[]> wordToDids  
    Map<int, Map<String, String>> didToMetaDatas  
end
```

Et le searcher peut désormais renvoyer des méta-données.

```
class FoundDocument  
    String url  
    Map<String, String> metaDatas  
end  
FoundDocument[] search(AST root)
```

Bonus 2: gestion des suppressions

Ajouter la gestion de la suppression d'un document.

Bonus 3: gestion des positions

Dans le texte d'origine, chaque mot est à une position. Cette position peut servir au ranking des résultats. Ajouter la gestion des positions.