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]" avant le Dimanche 19 Juin 23:59:59.

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

Les classes et méthodes proposées dans ce TP ne sont que des exemples, libre à vous de les nommer/architecturer autrement. Le TP sera évalué manuellement (sans moulinette).

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 des documents 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 (minusculisation + suppression des accents).

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

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

L'indexeur

Son rôle est d'inverser les documents. C'est à dire transformer une liste de documents contenant des mots, en une liste de mots associés à des documents.

```
class Index
    Map<String, String[]> wordToUrls;
end
Index build(TokenizedDocument[] documents)
```

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 et de retourner les URLs des documents qui matchent la requête.

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

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

Puis effectuer des requêtes mono-mot et multi-mots:

```
String[] search(Index index, String word)
String[] search(Index index, String[] words) # ANDed
```

L'indexation incrémentale

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. On utilise un "support" pour stocker dans quelle génération se trouve la dernière version de chaque document.

```
class Generation
    Map<String, String[]> wordToUrls;
end
class Index
    Generation[] generations
    Support support
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, String[]> wordToUrls
    Map<String, Map<String, String>> urlToMetaDatas
end
```

Et le searcher peut désormais renvoyer des méta-données.

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

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.