
UNIVERSITÉ D'AVIGNON
ET DES PAYS DE VAUCLUSE

M2 ILSEN – 2019/20

UE Ingénierie du document et de l'information

# UCE Indexation & recherche

Vincent Labatut

## TP 4 | Évaluation des performances

Ce TP est basé sur les classes développées lors des trois premiers TP. Nous avons pour l'instant un outil d'indexation et un moteur de résolution de requête. On veut maintenant exploiter le corpus springer, qui est annoté, afin de réaliser une évaluation de la qualité des résultats produits par notre outil.

## 1 Vérité terrain

Dans un premier temps, il nous faut charger la vérité terrain. Celle-ci prend la forme de 25 requêtes prédéfinies, pour lesquelles des experts ont manuellement identifié les documents pertinents dans le corpus.

### Exercice 1

Dans la classe `FileTools`, complétez la méthode `List<Posting> getPostingsFromFileNames(List<String> fileNames)` qui prend en entrée une liste de noms de fichiers du corpus, et renvoie la liste de postings correspondante.

*Exemple :* pour la liste `Arthroskopie.00130217.eng.abstr`, `Arthroskopie.80110114.eng.abstr`, `Bundesgesundheitsblatt.00430210.eng.abstr`, on obtient la liste de postings suivante :

```
[10, 33, 101]
```

> **Remarque :** notez qu'il s'agit donc de la fonction inverse de la méthode `getFileNamesFromPostings` écrite précédemment dans la même classe.

### Exercice 2

La vérité terrain est stockée dans le fichier XML `springer_reference.xml`, qui est lui-même situé dans `Common`. Ouvrez ce fichier, et vérifiez qu'il respecte la syntaxe suivante :

```
<queries>
  <query expr="xxxxx" >
    <doc>yyyyy</doc>
    <doc>zzzzz</doc>
    ...
  </query>
  ...
</queries>
```

Où `xxxxx` est une requête, et `yyyyy` et `zzzzz` sont les noms de fichier des documents vérifiant cette requête.

### Exercice 3

C'est la classe `GroundTruth` du package `performance` qui a pour rôle de charger la vérité terrain. Elle contient un champ `queries` destiné à stocker les requêtes d'évaluation, et un champ `postingLists` défini pour contenir les listes de docID des documents vérifiant ces requêtes.

Complétez le constructeur `GroundTruth()` de manière à ce qu'elle lise le fichier XML dont le chemin est renvoyé par `FileTools.getGroundTruthFile()`, et initialise les champs `queries` et `postingLists` en fonction de son contenu.

**Remarque :** Rappel : pour lire un fichier XML `myfile.xml` (avec la bibliothèque standard) :

```
DocumentBuilderFactory documentBuilderFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse("myfile.xml");
```

Puis on utilise le package `org.w3c.dom` pour manipuler les éléments composant le fichier XML, ainsi que leurs valeurs et attributs.

Pour contrôle, on veut obtenir la sortie suivante dans la console (les valeurs entre parenthèses correspondent aux nombres de documents associés à chacune des 25 requêtes) :

```
Reading ground truth file ..\Common\springer\_reference.xml
Found 25 queries (14 23 4 5 4 3 8 14 6 24 19 16 23 65 17 15 27 22 21 7 15 14 8 16 24)
```

**Remarque :** vous avez besoin d'utiliser la méthode `getPostingsFromFileNames` pour convertir les noms de fichiers contenus dans le document XML en docID.

## 2 Mesures de performance

Dans un premier temps, nous allons implémenter les mesures de performance présentées en cours pour le modèle booléen, à savoir : *Précision*, *Rappel*, et *F-mesure* (cf. le cours pour les formules exactes).

### Exercice 4

La classe `AbstractEvaluator` du package performance est utilisée pour mutualiser certains champs et méthodes nécessaires au calculer des mesures de performance. Complétez son constructeur de manière à initialiser un objet de type `GroundTruth`.

### Exercice 5

Complétez la méthode `Map<MeasureName,Float> evaluateQueryAnswer(int queryId, List<Posting> answer)`. Celle-ci reçoit en entrée un numéro de requête `queryId` désignant l'une des requêtes composant la vérité terrain, et la liste de postings `answer` renvoyée par le moteur de recherche pour cette même requête.

La méthode doit calculer la *Précision*, le *Rappel* et la *F-measure* en comparant la liste de postings renvoyée par le moteur de recherche et celle fournie par la vérité terrain. Ces valeurs sont ensuite placées dans une map en utilisant le type énuméré `MeasureName` qui est prédéfini dans `AbstractEvaluator`.

*Exemple :* supposons que le moteur renvoie la liste composée des postings 15 à 20 (inclus) pour la requête 1 ; alors on obtient la map suivante :

```
{PRECISION=0.5, RECALL=0.13043478, F_MEASURE=0.20689656}
```

**Remarque :** par convention, on considère que la *Précision* est nulle lorsqu'aucun document n'est renvoyé par le moteur. Au contraire, quand aucun document pertinent n'existe pour une requête, on considère que le *Rappel* est de 1. Enfin, quand La *Précision* et le *Rappel* sont nuls, on considère que la *F-measure* l'est aussi (puisque c'est leur moyenne).

### Exercice 6

Complétez la méthode `List<Map<MeasureName,Float» evaluateQueryAnswers (List<List<Posting» answers)` qui fait la même chose, mais pour toutes les requêtes d'évaluation. La fonction reçoit une liste `answers`, contenant les réponses produites par le moteur de recherche pour chacune des requêtes d'évaluation. Elle applique `evaluateQueryAnswer` sur chacune d'entre elles pour produire une liste de maps. À la fin de cette liste, elle rajoute une map supplémentaire contenant les valeurs moyennes pour chacune des trois mesures, sur l'ensemble des requêtes traitées.

*Exemple :* supposons qu'on utilise la même liste de postings de 15 à 20 que précédemment, pour chacun des 25 requêtes d'évaluation ; on obtient alors la liste de maps suivante (la partie

surlignée correspond aux valeurs moyennes) :

```
[{PRECISION=0.0, RECALL=0.0, F_MEASURE=0.0},
 {PRECISION=0.5, RECALL=0.13043478, F_MEASURE=0.20689656},
 {PRECISION=0.0, RECALL=0.0, F_MEASURE=0.0},
 ...
 {PRECISION=0.0, RECALL=0.0, F_MEASURE=0.0},
 {PRECISION=0.0, RECALL=0.0, F_MEASURE=0.0},
 {PRECISION=0.033333335, RECALL=0.008332776, F_MEASURE=0.013038987}]
```

### Exercice 7

Écrivez la méthode `void writePerformances(List<Map<MeasureName,Float» values)` qui enregistre les performances de l'index dans un fichier texte. Chaque map de la liste values prend la forme d'une ligne dans le fichier, tandis que chaque mesure est une colonne.

Le fichier obtenu doit être de la forme suivante :

```
0.64    0.051079914 0.09146231
0.48    0.078756616 0.12947558
0.52    0.12913457  0.19492045
```

Où les valeurs situées sur la même ligne sont séparées par des tabulations.

### Exercice 8

Pour l'évaluation d'index utilisant le modèle booléen, nous allons utiliser `BooleanEvaluator`, qui est une classe-fille d'`AbstractEvaluator`.

Complétez sa méthode `List<Map<MeasureName,Float>> evaluateEngine (AndQueryEngine engine)`, qui réalise l'évaluation complète du moteur de recherche passé en paramètre. Elle sollicite ce moteur pour traiter chacune des requêtes d'évaluation constituant la vérité terrain, puis invoque `evaluateQueryAnswers` pour calculer les performances correspondantes. Elle les enregistre dans un fichier grâce à `writePerformances`, avant de finalement les renvoyer.

## 3 Évaluation

Nous allons maintenant utiliser l'évaluateur pour déterminer les performances de nos index et moteur de recherche.

### Exercice 9

Dans `Test1`, complétez la méthode `testEvaluation` de manière à charger l'index, créer le moteur et l'évaluateur, appliquer ce dernier, et afficher les résultats obtenus.

On veut détailler la sortie console pour faire apparaître les performances correspondant à chaque requête d'évaluation.

*Exemple :*

```
PRECISION   RECALL   F_MEASURE
0.00        0.00     0.00        Arthroscopic treatment of cruciate ligament injuries
0.00        0.00     0.00        Complications of arthroscopic interventions
....        ....     ....        ...
0.35        0.29     0.32        Treatment of acute myocardial infarction
....        ....     ....        ...
0.00        0.00     0.00        New approach in cruciate ligament surgery
----------------------------------------
0.38        0.10     0.14
```

Discutez les résultats obtenus, notamment en comparant les 3 mesures calculées.

### Exercice 10

Pour explorer ces résultats et comprendre la raison de ces piètres performances, nous allons modifier les méthodes traitant les requêtes, afin de leur faire afficher plus d'informations. Dans `AndQueryEngine`, modifiez `splitQuery` afin de lui faire afficher le résultat de la normalisation.

*Exemple :* pour la première requête d'évaluation, on obtient :

```
Processing query "Arthroscopic treatment of cruciate ligament injuries"
   Normalizing: "arthroscopic"(83) "treatment"(2092) "of"(7667) "cruciate"(54)
   "ligament"(99) "injuries"(256)
Query processed, returned 0 postings, duration=1 ms
```

### Exercice 11

Dans `AndQueryEngine`, modifiez `processConjunctions` afin de lui faire afficher les tailles des listes de postings après qu'elles aient été ordonnées.

*Exemple :* pour la première requête d'évaluation, on obtient :

```
Processing query "Arthroscopic treatment of cruciate ligament injuries"
  Normalizing: "arthroscopic"(83) "treatment"(2092) "of"(7667) "cruciate"(54)
"ligament"(99) "injuries"(256)
   Ordering posting list: (54) (83) (99) (256) (2092) (7667)
Query processed, returned 0 postings, duration=1 ms
```

### Exercice 12

Dans `AndQueryEngine`, modifiez `processConjunction` afin de lui faire afficher les tailles des listes de postings reçues et produite.

*Exemple :* pour la première requête d'évaluation, on obtient :

```
Processing query "Arthroscopic treatment of cruciate ligament injuries"
  Normalizing: "arthroscopic"(83) "treatment"(2092) "of"(7667) "cruciate"(54)
"ligament"(99) "injuries"(256)
  Ordering posting list: (54) (83) (99) (256) (2092) (7667)
   Processing conjunction: (54) AND (83) >> (15)
   Processing conjunction: (15) AND (99) >> (14)
   Processing conjunction: (14) AND (256) >> (3)
   Processing conjunction: (3) AND (2092) >> (0)
Query processed, returned 0 postings, duration=1 ms
```

### Exercice 13

Discutez les résultats obtenus, à la lumière des informations affichées maintenant lors du traitement des requêtes d'évaluation, et en allant vérifier dans les fichiers du corpus.

Par exemple, pourquoi aucun des documents dont le nom de fichier commence par `Arthroskopie` ne sont considérés comme pertinents lors du traitement de la première requête d'évaluation ?

## 4 Racinisation

Pour essayer d'améliorer ces performances, nous allons modifier la partie normalisation en intégrant une étape de racinisation (stemming).

Le package `indexation.processing` contient une classe `AbstractStemmer` représentant un racinisateur en général, et possédant une méthode `String stemType(String string)` : celle-ci prend en paramètre un mot `string` à raciniser, et renvoie sa racine.

Le même package contient également `PorterStemmer`, une classe-fille de `AbstractStemmer`. Elle implémente l'approche à base de règles proposée par Martin Porter en 1979. Il s'agit simplement d'une succession de tests réalisés sur le mot à traiter, et aboutissant à des réécritures destinées à supprimer des suffixes prédéfinis. Par exemple, le mot `generalizing` est transformé en `generalize`, lui-même transformé ensuite en `general`.

### Exercice 14

Dans la classe `Normalizer`, remarquez que le constructeur n'initialise le champ `stemmer` (i.e. le racinisateur) que si la classe `Configuration` le spécifie : autrement, ce champ reste `null`.

Modifiez la méthode `normalizeType` de manière à appliquer, en toute fin de traitement, le racinisateur contenu dans le champ `stemmer` sur la chaîne considérée. Attention, vous devez vérifier si `stemmer` a été initialisé ou pas (i.e. s'il est `null`). En l'absence de racinisateur, on ne réalise pas de racinisation, bien entendu.

### Exercice 15

Dans `Test1`, modifiez le `main` pour indiquer dans `Configuration` qu'il faut utiliser la racinisation. Puis, refaites l'indexation en utilisant le racinisateur, et répétez l'évaluation de l'index résultant. Discutez les changements observés dans les performances obtenues.

**Remarque :** pour être rigoureux, il faudrait distinguer un ensemble de requêtes de développement, servant à identifier le paramétrage optimal du moteur de recherche, et un ensemble de requêtes de test, utilisé pour déterminer si les performances se généralisent à de nouvelles données. Cependant, en raison de la faible quantité de données disponibles pour le TP, on utilisera toutes les requêtes pour les deux tâches.