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 6 | Correction

1 Représentation des scores

Exercice 1

```
private static NumberFormat NUMBER_FORMAT =
    NumberFormat.getInstance(Locale.ENGLISH);
{  NUMBER_FORMAT.setMaximumFractionDigits(4);
   NUMBER_FORMAT.setMinimumFractionDigits(4);
}

public String toString()
{  String scoreStr = NUMBER_FORMAT.format(score);
   String result = docId + " (" + scoreStr + ")";
   return result;
}
```

Exercice 2

```
public int compareTo(DocScore docScore)
{  int result = (int)Math.signum(score - docScore.score);
   if(result==0)
      result = docId - docScore.docId;
   return result;
}

public boolean equals(Object o)
{  boolean result = false;
   if(o instanceof DocScore)
   {  DocScore docScore = (DocScore) o;
      result = compareTo(docScore) == 0;
   }
   return result;
}
```

2 Décompte des termes

Exercice 3

```
public String toString()
{  String result = "<" + docId + " [" + frequency + "]>";
   return result;
}
```

Exercice 4

```
public int filterTokens(List<Token> tokens, List<Integer> frequencies)
{  int result = 0;
   int count = 0;

   // on passe chaque paire de tokens consécutifs en revue
   Iterator<Token> it = tokens.iterator();
   Token t1 = null;
```

```
    while(it.hasNext())
    {  Token t2 = it.next();
       if(t1==null)
       {  result = 1;
          count = 1;
       }
       else
       {  // si deux tokens consécutifs sont identiques,
          // on supprime le second
          if(t1.equals(t2))
          {  it.remove();
             count++;
          }
          else
          {  frequencies.add(count);
             count = 1;
          }
          // on compte les termes
          String type1 = t1.getType();
          String type2 = t2.getType();
          if(!type1.equals(type2))
             result++;
       }
       t1 = t2;
    }
    frequencies.add(count);
    return result;
 }
```

Exercice 5

```
 private int buildPostings(List<Token> tokens, List<Integer> frequencies,
     AbstractIndex index)
 {  int result = 0;
    int i = 0;
    IndexEntry entry = null;

    // on traite chaque token séparément
    Iterator<Token> itTok = tokens.iterator();
    Iterator<Integer> itFreq = frequencies.iterator();
    while(itTok.hasNext())
    {  Token token = itTok.next();
       int frequency = itFreq.next();
       String type = token.getType();
       // si besoin, on crée une nouvelle entrée
       if(entry==null || !entry.getTerm().equals(type))
       {  entry = new IndexEntry(type);
          index.addEntry(entry,i);
          i++;
       }

       // dans tous les cas, on met àjour la liste de postings
       int docId = token.getDocId();
       Posting posting = new Posting(docId, frequency);
       entry.addPosting(posting);
       result++;
    }

    return result;
```

```
28  }
```

Exercice 6

```
1   public AbstractIndex buildIndex(List<Token> tokens, LexiconType lexiconType)
2   {  int indexSize;
3      AbstractIndex result = null;
4      List<Integer> frequencies = new LinkedList<Integer>();
5
6      ...
7
8      {  System.out.println(" Filtering tokens...");
9         long start = System.currentTimeMillis();
10        indexSize = filterTokens(tokens, frequencies);
11        ...
12     }
13
14     {  ...
15        int postingNumber = buildPostings(tokens, frequencies, result);
16        ...
17     }
18
19     return result;
20  }
```

3 Calcul des scores

Exercice 7

```
1   private float processWf(Posting posting)
2   {  int tf = posting.getFrequency();
3      float result = 0;
4      if(tf>0)
5         result = 1 + (float)Math.log10(tf);
6      return result;
7   }
```

Exercice 8

```
1   private float processIdf(IndexEntry entry)
2   {  float df = entry.getFrequency();
3      int docNbr = index.getDocumentNumber();
4      float result = (float) Math.log10(docNbr/df);
5      return result;
6   }
```

Exercice 9

```
1   private void sortDocuments(List<IndexEntry> queryEntries, int k, List<DocScore>
        docScores)
2   {  // initialisation
3      TreeSet<DocScore> orderedIds = new TreeSet<DocScore>();
4      int docNbr = index.getDocumentNumber();
5      final float scores[] = new float[docNbr];
6      Arrays.fill(scores, 0);
7      float norms[] = new float[docNbr];
8      Arrays.fill(norms, 0);
9      float queryNorm = 0;
10
11     // on parcourt tous les termes de la requête
12     for(IndexEntry entry: queryEntries)
13     {  float idf = processIdf(entry);
```

```
14
15          // on calcule le poids individuel du terme pour la requête
16          float stq = idf;
17          // on met àjour sa norme
18          queryNorm = queryNorm + (float)Math.pow(stq,2);
19
20          // pour tous les postings contenant le terme traité
21          List<Posting> postings = entry.getPostings();
22          for(Posting posting: postings)
23          {  // on calcule le score individuel du terme pour le document
24             float std = processWf(posting) * idf;
25             // on met àjour les scores et normes
26             int docId = posting.getDocId();
27             scores[docId] = scores[docId] + stq*std;
28             norms[docId] = norms[docId] + (float)Math.pow(std,2);
29          }
30       }
31
32       // on termine le calcul des normes
33       for(int i=0;i<norms.length;i++)
34          norms[i] = (float)Math.sqrt(norms[i]);
35       queryNorm = (float)Math.sqrt(queryNorm);
36
37       // on termine le calcul des scores et on ordonne les documents
38       for(int i=0;i<scores.length;i++)
39       {  if(norms[i]==0)
40             scores[i] = 0;
41          else
42             scores[i] = scores[i] / (norms[i]*queryNorm);
43          DocScore docScore = new DocScore(i, scores[i]);
44          orderedIds.add(docScore);
45       }
46
47       // on garde tout si k vaut zéro
48       if(k==0)
49          k = orderedIds.size();
50       // on ajoute dans le bon ordre
51       Iterator<DocScore> it = orderedIds.descendingIterator();
52       int i = 0;
53       while(i<k && it.hasNext())
54       {  DocScore docScore = it.next();
55          docScores.add(docScore);
56          i++;
57       }
58    }
```

Exercice 10

```
1   private void splitAndQuery(String query, List<IndexEntry> result)
2   {  // on tokénize la requête
3      Tokenizer tokenizer = index.getTokenizer();
4      List<String> types = tokenizer.tokenizeString(query);
5
6      // on normalise chaque type
7      Normalizer normalizer = index.getNormalizer();
8      for(String type: types)
9      {  // la normalisation du type donne le terme (ou null)
10        String term = normalizer.normalizeType(type);
11        if(term!=null)
```

```
12          {  // on récupère l'entrée associée au terme dans l'index
13             IndexEntry entry = index.getEntry(term);
14             // si pas dans l'index, on n'ajoute rien dans la liste
15             // sinon, on ajoute l'entrée dans la liste résultat
16             if(entry!=null)
17                result.add(entry);
18          }
19       }
20    }
```

Exercice 11

```
1   public List<DocScore> processQuery(String query, int k)
2   {  System.out.println("Processing query \""+query+"\"");
3      long start = System.currentTimeMillis();
4
5      // on décompose la requête et identifie les termes
6      List<IndexEntry> queryEntries = new LinkedList<IndexEntry>();
7      splitAndQuery(query,queryEntries);
8
9      // on calcule les scores
10     List<DocScore> result = new LinkedList<DocScore>();
11     sortDocuments(queryEntries,k,result);
12
13     long end = System.currentTimeMillis();
14     System.out.println("Query processed, returned "+result.size()+" values,
           duration="+(end-start)+" ms");
15     return result;
16  }
```

4 Évaluation des performances

Exercice 12

```
1   public static List<String> getFileNamesFromDocScores(List<DocScore> docScores)
2   {  List<String> result = new LinkedList<String>();
3      File folder = new File(getCorpusFolder());
4      String fileNames[] = folder.list();
5      Arrays.sort(fileNames);
6
7      for(DocScore docScore: docScores)
8      {  int docId = docScore.getDocId();
9         String name = fileNames[docId];
10        result.add(name);
11     }
12
13     return result;
14  }
```

Exercice 13

```
1   private static void testQuery() throws IOException, ClassNotFoundException
2   {  List<String> queries = Arrays.asList(
3         "bible",
4         "solar panel electricity"
5      );
6
7      // chargement de l'index
8      AbstractIndex index = AbstractIndex.read();
9
10     // résolution de la requête
```

```
11      int k = 5;
12      RankingQueryEngine engine = new RankingQueryEngine(index);
13      for(String query: queries)
14      {  List<DocScore> docScores = engine.processQuery(query,k);
15         System.out.println("Result: "+docScores.size()+ " document(s)\n"+docScores);
16         System.out.println("Files:\n"+getFileNamesFromDocScores(docScores)+"\n");
17      }
18   }
```

Si la requête ne contient qu'un seul terme t, comme c'est le cas pour la première, alors on se ramène à la situation suivante :

$$\text{sim}(d,q) = \frac{\vec{V}(d)\cdot\vec{V}(q)}{|\vec{V}(d)|\times|\vec{V}(q)|} \tag{1}$$

$$= \frac{s(t,d)s(t,q)}{\sqrt{s(t,d)^2}\sqrt{s(t,q)^2}} \tag{2}$$

$$= \frac{s(t,d)s(t,q)}{s(t,d)s(t,q)} \tag{3}$$

Autrement dit : si le document contient le terme t alors $s(t,d)$ sera non-nul et $sim(d,q)$ sera donc égal à 1, ce que l'on interprétera comme une réponse positive (document d jugé pertinent) ; tandis que s'il ne le contient pas, $s(t,d)$ sera nul, et le rapport $sim(d,q)$ sera indéfini, ce que l'on interprétera comme une réponse négative (document d jugé non-pertinent). Dans ce cas particulier, on se ramène donc à une évaluation booléenne.

La qualité est assez mitigée, avec des résultats pertinents concernant des auteurs et la page sur la littérature, et d'autres plus éloignées thématiquement (Algérie, Géographie).

Pour la deuxième, les résultats sont dominés par des pages traitant de la transmission d'information sur le net ou le Web : clairement, l'absence de contrainte spatiale sur la requête se fait sentir (les pages renvoyées sont relatives au mots de la requête pris séparément, mais pas forcément au sens de l'expression prise dans son ensemble). Pour la troisième requête, tous les résultats renvoyés sont dans le thème (panneaux solaires dans l'aérospatiale, investissements par Google, économies d'énergie).

Notez que des méthodes à base de similarité plus avancées existent, employant notamment des pondérations différentes lors du calcul de score.

Exercice 14

```
1   private List<Map<MeasureName,Float>> evaluateQueryAnswers(List<List<DocScore>>
        answers, int k)
2   {  List<List<Posting>> convAnswers = new ArrayList<List<Posting>>();
3
4      // on convertit chaque réponse séparément
5      for(List<DocScore> answer: answers)
6      {  List<Posting> convAnswer = new ArrayList<Posting>();
7         convAnswers.add(convAnswer);
8         Iterator<DocScore> it = answer.iterator();
9         int i = 0;
10        // on ne garde que les k premiers documents
11        while(it.hasNext() && i<k)
12        {  DocScore docScore = it.next();
13           int docId = docScore.getDocId();
14           Posting posting = new Posting(docId);
15           convAnswer.add(posting);
16           i++;
17        }
18     }
19
```

```
20      // on applique ensuite la méthode existante
21      List<Map<MeasureName,Float>> result = evaluateQueryAnswers(convAnswers);
22      return result;
23    }
```

Exercice 15

```
1   public List<Map<MeasureName,Float>> evaluateEngine(RankingQueryEngine engine)
        throws FileNotFoundException, UnsupportedEncodingException
2   {  System.out.println("Evaluating the search engine");
3
4      AbstractIndex index = engine.getIndex();
5      int docNbr = index.getDocumentNumber();
6
7      // on traite chaque requête d'évaluation
8      List<List<DocScore>> answers = new ArrayList<List<DocScore>>();
9      List<String> queries = groundTruth.getQueries();
10     for(String query: queries)
11     {  List<DocScore> answer = engine.processQuery(query,0);
12        answers.add(answer);
13     }
14
15     // on calcule les performances correspondant aux réponses
16     List<Map<MeasureName,Float>> result = new ArrayList<Map<MeasureName,Float>>();
17     for(int k=1;k<=docNbr;k++)
18     {  System.out.println("k="+k);
19        List<Map<MeasureName,Float>> temp = evaluateQueryAnswers(answers,k);
20        Map<MeasureName,Float> meanVals = temp.get(temp.size()-1);
21        result.add(meanVals);
22     }
23
24     writePerformances(result);
25     return result;
26  }
```

Exercice 16

On obtient les graphiques de la Figure 1, qui représentent les Precisions/Rappels moyennés sur l'ensemble des requêtes :

Le meilleur compromis en termes de *F-mesure* moyenne est obtenu pour les valeurs indiquées dans la Table 1. Il est représenté par la croix dans le graphique de gauche (Figure 1).

k	Précision	Rappel	F-mesure
22	0,36	0,58	0,41

Table 1. Meilleur compromis de performance.

Comme ce n'est qu'une courbe moyenne, il est tout à fait possible que le comportement diffère beaucoup d'une requête à l'autre. Ainsi, en représentant les courbes de Précision/Rappel de chaque requête séparément sur le même graphique, on obtient le résultat présenté dans le graphique de droite (Figure 1), qui montre effectivement une grande dispersion des performances : le moteur est très bon pour certaines requêtes, et très mauvais pour d'autres. La Figure 2 montre les performances obtenues pour deux cas extrêmes.

Pour information, voici les scripts R utilisés pour générer ces graphiques.

Valeurs moyennes :

```
t <- read.table("data/springer_filter_stem_score_performance.txt")
plot(t[,2], t[,1], col="red", xlab="Rappel moyen", ylab="Précision moyenne",
    type="l", xlim=c(0,1), ylim=c(0,1))
idx <- which.max(t[,3])
```

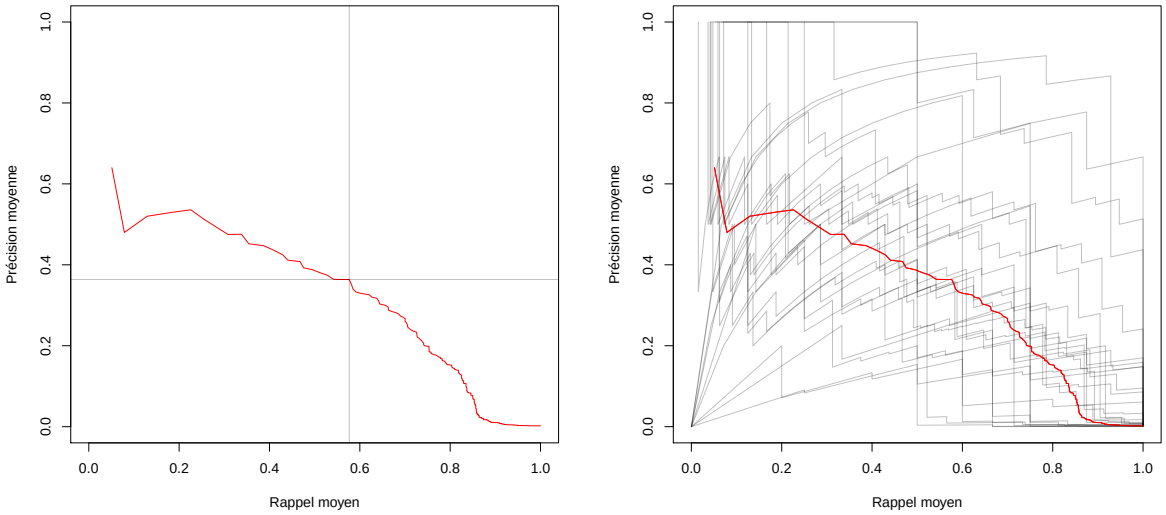

Figure 1. Performances moyennes (gauche) et performances individuelles pour chacune des 25 requêtes.

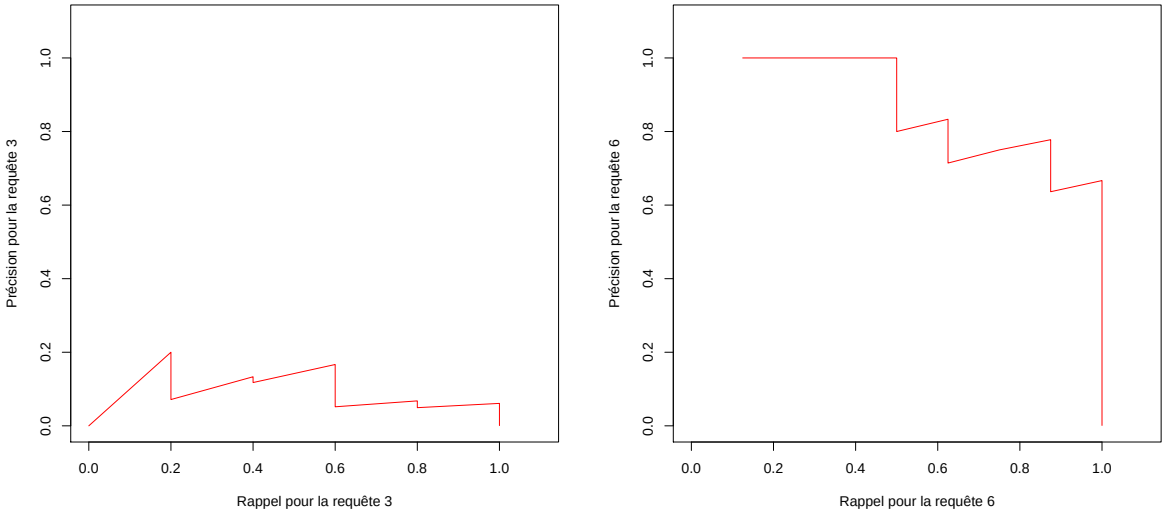

Figure 2. Deux cas extrêmes parmi les 25 requêtes traitées.

```
abline(v=t[idx,2],col=rgb(0,0,0,0.25))
abline(h=t[idx,1],col=rgb(0,0,0,0.25))
```

Requêtes individuelles :

```
i <- 3
t <- read.table(data/springer_filter_stem_score_performance.txt_",i,".txt"))
plot(t[,2], t[,1], col="red", xlab=paste("Rappel pour la requête",i),
ylab=paste("Précision pour la requête",i), type="l", xlim=c(0,1.1), ylim=c(0,1.1))
```