Rapport 1 – LO17

Préparation et Indexation du corpus

Yiwen WANG | Yinong QIU

30/10/2019

Sommaire

1. Introduction

Tout au long de l'UV LO17, nous devons réaliser un système d'indexation et de recherche d'information sur une archive construite à partir du site de l'ADIT. Nous allons étudier un échantillon de cette archive correspondant à plus de 300 articles extraits de bulletins de veille sur la France entre 2011 et 2014.

L'objectif de ces deux premier TDs est de d'abord préparer ce corpus d'articles en vue de son indexation. Pour réaliser l'indexation, nous allons d'abord construire un fichier, qui rassemble tous les articles dans une structure XML. Et ensuite nous devons créer les index des articles des bulletins électronique de l'ADIT, à partir du fichier que nous avons réalisé avant.

Le résultat de l'indexation sera donc un ensemble de fichiers inverses, chaque fichier correspondant à une balise ou un ensemble de balises.

Ce rapport représente principalement le processus et les méthodes utilisées pour préparer le corpus XML et réaliser son indexation.

2. Préparation et structuration du corpus

On veut réaliser un système d'indexation et de recherche d'information sur une archive construite à partir du site de l'ADIT. L'ADIT diffuse, sous forme de bulletins électroniques, des informations internationales de veille technologique et scientifique.

2.1. Analyse des fichiers html

Ici on prend un exemple de page web de l'ADIT, avec les différentes informations à traiter :

Numéro :280

Date :30/04/2013

Rubrique :Événement

Titre : Innovatives SHS: les …

Fichier : 72939.htm

Texte : En science humaines …

Images: (peut y avoir 0, 1 ou plusieurs)

Url de l'image : <url de l'image>

Légende d'image: Innovatives SHS, un salon de …

Contact :CNRS - Laetitia Louis tél et email ...

Notre corpus initial contient 326 fichiers html. L'objectif de est de créer un seul fichier XML contenant pour chaque page html les informations relatives à la partie information sélectionnée(toutes les balises au dessus).

2.2. Le choix des balises html

Puisqu'un script Perl traite un fichier ligne par ligne, nous devons observer les lignes contenant les informations utiles, analyser ses structures puis trouver ses caractéristiques uniques pour les extractions.

Nous avons d'abord choisi quelques échantillons depuis les fichiers html. Ici on prend un exemple, et on veux extraire son titre. La ligne contenant son titre est :

```
<p class="style96">
    <span class="style42">Ev&eacute;nement<br></span>
    <span class="style17">Innovatives SHS : les sciences humaines et
    sociales ont enfin leur salon de la valorisation !</span>
</p>
```

où « ***Evénement*** » est son rubrique et « ***Innovatives SHS : les sciences humaines et sociales ont enfin leur salon de la valorisation !*** » est son titre. On peut donc extraire ces deux informations en même temps. La caractéristique de cette ligne est qu'elle commence par deux balises « ***<p class="style96">*** ». Pour prouver sa unicité dans le fichier sélectionné, on l'a recherché dans le fichier et il nous renvoie un seul résultat. En premier temps on a décidé de considérer ces deux balise comme sa caractéristique. On peux donc rédiger une méthode Perl qui peut extraire les informations « titre » et « rubrique » depuis ce fichier. Puis on l'utilise pour tous les fichiers et vérifie la qualité de cette méthode.

De même façon, on a simplement extrait quelques informations tels que « numéro » , « date », « fichier », « contact » etc. car ses caractéristique sont relativement simples à trouver.

« image » et « texte » sont un peu particulières comme le nombre de images peut être 0,1 ou plusieurs selon les différents fichiers, et « texte » est parfois séparé en plusieurs lignes. On a donc proposé deux différentes méthodes pour chacun des deux.

Pour « image », on a trouvé « *** <img src="*** » comme sa caractéristique. On a défini deux variables du type liste, une pour stocker les URLs des images et l'autre pour stocker ses légendes. « **légendeImage** » est le texte après la balise « ****** ». Dans le cas où il n'y a pas de légende, on stocke une chaîne vide pour prendre la place correspondante à l'image

dans la liste de légende. A l'aide de cette méthode on a pu traiter tous les cas possible pour tous les fichiers.

Pour « texte », on a trouvé deux cas possibles : le cas où le texte se trouve sur une seule ligne et le cas où il est séparé en plusieurs lignes par
 ou par une image.

Comme tous les textes sont dans la balise « ****** », et ils commencent par balise « *** *** » et se terminent par « *** *** », on peut donc proposer une méthode particulière qui fusionne toutes les lignes concernant le texte pour extraire la balise « texte ». On va préciser la méthode utilisée dans la partie suivante.

Pour vérifier l'unicité des caractéristiques choisies et la qualité des méthodes sélectionnées, on a utilisé deux méthodes :

1) Grace à la commande **grep** , on peut facilement savoir le nombre des différentes balises dans les différentes fichiers. Pour prouver l'unicité on peut rechercher la balise caractérisée dans tous les fichiers et comparer la valeur de retour avec le nombre total de fichiers pour prouver l'unicité de cette méthode.
2) Après l' exécution de script Perl, on imprime un log directement sur console en indiquant le nombre total de chaque balise traitée ainsi que le nombre de fermetures de balise, et le comparer avec le nombre de fichiers.

2.3. Algorithmes Perl pour réaliser l'extraction

On a tout d'abord utilisé le script donnée « convert_P16 » pour transformer les caractères spéciaux en format normal et on obtient le fichier xml transformé « corpus.xml. » Puis dans le boucle while<> de notre script,on utilise pour chaque balise une expression rationnelle pour capturer les informations utiles. Prenons l'exemple de l'extraction de balise numéro/titre:

On obtient le numéro entre class="style32">BE France et .*?class="style42"> et le stocker dans la variable correspondante. Et on définie un format de date à l'aide d'expression rationnelle puis extrait la date depuis cette ligne.

```
if (/class="style32">BE France (\d+).*?class="style42">.*?(\d+?\/\d+?\/\d+)/){
        $num=$1;
        $date=$2;
}
```

On utilise le même façon pour extraire simplement les informations rubrique,titre,contact etc.

Pour l'extraction de texte et celle des images, on a proposé deux différentes méthodes:

1) **Texte** : Puisque tous les texte commencent par « *** *** », dans le premier cas où le texte se trouve sur une seule ligne, on utilise l'expression rationnelle pour capturer le texte entre la balise « *** *** » et « *** *** ».

   Si on trouve pas la fermeture de la balise « *** *** » dans cette ligne, on définie une variable $flag et en fournie la valeur 1 pour indiquer que le contenu de la balise ****** n'est pas complètement tiré. Puisque dans la balise *** *** on a probablement plusieurs balises ******, ici on définie également une variable **$paragraph** et l'affecte la valeur 1 pour indiquer qu l'on est entrain de fusionner tous les contenus de balises ****** de la paragraphe. On analyse et fusionner les prochaines lignes jusqu'à la fermeture de la balise ****** puis on affecte la valeur 0 au **$flag**, cela signifie que cette balise ****** est fermé et son contenu a été complément fusionné dans la variable $texte. Une fois on trouve la balise , on donne la valeur 0 au **$paragraph** pour signifier que la partie d'extraction de texte est terminée.

   Afin d'assurer que le texte possède une seule ligne, on enlève tous les symboles de saut de ligne en utilisant la fonction « **chomp** ».

2) **Image** : Pour traiter les fichiers qui peuvent contenir 0,1 ou plusieurs images, on a proposé une méthode générique. On a défini deux variable de type liste qui servent à stocker respectivement les liens des images et les légendes des images.

   Une fois on trouve une image, on extrait son lien à l'aide d'expression rationnelle et on l'ajoute dans la liste des images à l'aide de fonction « **push** ». De même, on extrait le légende correspondant et on l'ajoute dans la liste de légendes.

   Afin de traiter les images qui n'ont pas de légende, une fois on a l'url de l'image on ajoute une chaîne vide (par exemple un espace) pour prendre la place correspondante dans la liste de légendes. Cela nous permet d'éviter l'erreur de décalage entre le lien d'image et le légende de l'image lors qu'on lance l'extraction. Le résultat attendu pour légende vide est « ***<legendeImage></legendeImage>*** ».

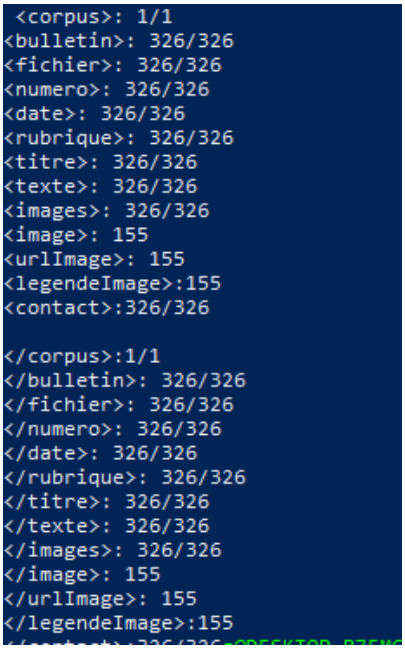
A la fin de script, on imprime toutes les balises et ses contenus correspondants :

```
print DATA "<bulletin>\n";
print DATA "<numero>$num</numero>\n";
print DATA "<date>$date</date>\n";
…
print DATA "</bulletin>\n";
```

On a testé ce script sur quelques fichiers. Puis on appelle le script précédent sur tous les fichiers de l'archive et écrit le résultat dans le fichier unique qui nomme « **newcorpus.xml** ».

2.4. Vérification de la qualité de l'extraction

Pour vérifier la qualité de l'extraction, on a choisi la méthode d'imprimer le log d'exécution directement sur le console. On a essayé d'afficher le plus d'éléments sur le log pour vérifier en même temps l'exhaustivité de fichier exporté, l'extraction de balises,la complétude des image etc.



On a finalement trouvé 155 images dans le fichier XML exporté. Ce nombre est identique avec celui de calcul à l'aide de fonction « grep ».

Et le nombre de fichiers et le nombre de balises traitées sont identiques avec nos résultats attendus.

On n'a pas trouvé une très bonne méthode pour vérifier la complétude des textes. On a choisi quelques échantillons depuis les fichiers html, et comparer leur parties de textes avec celles de fichier XML. Et on peut constater que notre extraction fonctionne bien pour ces fichiers. Cela a prouvé également que notre méthode d'extraction a un bon taux de précision.

Pour certaines balises comme « **images** » , « **légendeImage** » qui peuvent être vide, on a testé aussi ses existences dans le fichier XML.

Grace à ce TD on a pu connaître fonctionner le script Perl et extraire les information utiles depuis un grand nombre de fichiers html. Après quelques améliorations, on a finalement exporté un fichier « **corpus.xml », qui sera utilisé pour l'indexation dans le prochain TD.**

3. Indexation

On souhaite, créer les index des articles des bulletins électroniques de l'ADIT, à partir du fichier XML que l'on a réalisé dans les TD précédents. Le principe est de créer une série de fichiers (dits fichiers inverses) permettant de décrire les documents à l'aide de tout ou partie des éléments contenus dans ce fichier XML. Le résultat de l'indexation sera donc un ensemble de fichiers inverses, chaque fichier correspondant à une balise ou un ensemble de balises (date ou numéro du bulletin, rubrique, titre, titre+texte de l'article, texte de l'article, images, légende des images, . . .).

Pour analyser les documents il faut clairement définir l'unité documentaire. Pour ce TD on a choisi l'option **un document = un article** mais pas **un document = un bulletin**. Puisque les informations principales d'un document existent souvent dans l'article plutôt que d'autres informations du bulletin. Les informations telles que la date, la section, le contact, etc. ne peuvent pas être utilisés en tant que mots-clés du document. Par conséquent on considère un article comme l'unité documentaire. Et on calcule la fréquence des mots qui apparaissent dans le titre ou le texte de chaque article.

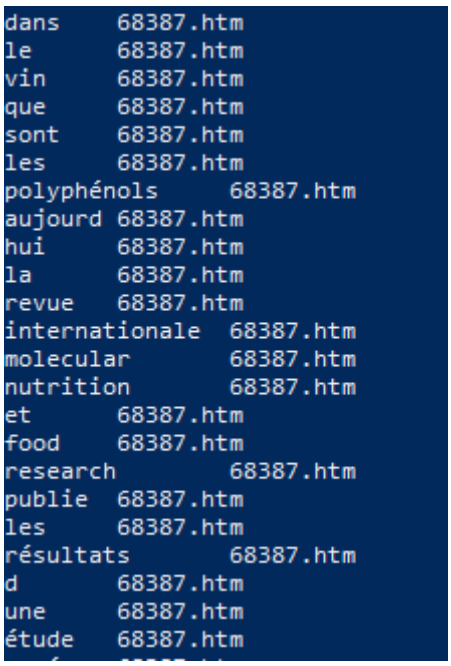
3.1. Calcul du tf * idf

3.1.1. Génération du fichier tf.txt

On a commencé par construire le fichier des coefficients **tf** de chaque mot t dans chaque article **d**.

On a tout d'abord essayé d'utiliser le script donné « **segmente_TT.pl** » pour découper le corpus (les titres et les textes) en tokens. On a trois options lors de l'exécution de script, ici on prend l'option -f car on veut obtenir une colonne « fichier» où se trouve les tokens.

L'exécution de **segmente_TT.pl** avec option **-f** donne le résultat ci-dessous, la première colonne contient le mot dans le fichier, et la deuxième colonne contient le fichier .htm correspondant :

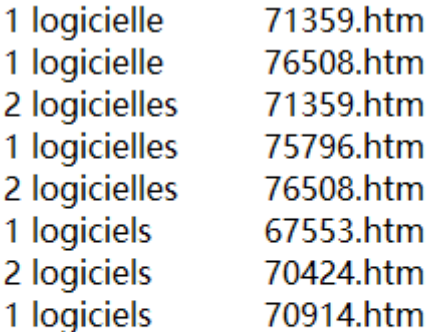


Comme tous les mots de tous les fichiers sont découpés en tokens, il peut avoir plein de lignes identiques dans le fichier généré. Pour chaque mot t, le nombre de lignes identiques est donc le nombre d'occurrence de mot t dans l'article d.

Notre commande pour obtenir la liste ordonnée de **tf** est :

```
cat corpus.xml |perl segmente_TT.pl -f|sort |uniq -c > tf.txt
```

Le résultat d'exécution (**tf.txt**) est comme ci-dessous :



tf.txt

3.1.2. Génération du fichier df.txt

Pour calculer le nombre de documents dans laquelle le mot t apparaît (**df**), on reprend le résultat de **tf.txt** et on a observé que la ligne qui contient le mot t et l'article d est unique dans le fichier **tf.txt**. On prend donc la seconde colonne « mot t » depuis le fichier **tf.txt** et on calcul pour chaque mot t son nombre d'occurrence, puis on obtient les valeurs de **df**. La commande que l'on utilise est ci-dessous :

```
cat corpus.xml |perl segmente_TT.pl -f|sort -u|cut -f 1|uniq -c >df.txt
```

Le résultat d'exécution (exemple):

```
13 abord
 1 aborde
10 aborder
 1 abordera
 1 aborderont
 1 abordé
```

df.txt

La deuxième colonne représente le mot, et la première colonne est la valeur de **df** correspondant.

3.1.3. Génération du fichier tf_idf.txt

Afin de pouvoir exporter le résultat de calcul de **tf*idf**, on a rédiger un script Perl **tf_idf.pl.**

Dans le script on a tout d'abord calculer **idf** en prenant les valeurs de fichier « **df.txt** ». On a définie une variable associative, puis on stocke le mot t comme clef et son résultat de idf comme valeur. Dans le fichier « **tf.txt** » on fait le calcul puis on remplace les « **tfs** » par les résultat de **tf*idf** en utilisant une expression rationnelle.

A la fin on obtient un fichier contenant trois colonnes : **tf*idf**, mot t et article d.

```
0.981738683025684   européenne   67804.htm
0.981738683025684   européenne   67939.htm
0.981738683025684   européenne   68277.htm
0.981738683025684   européenne   68383.htm
0.981738683025684   européenne   68645.htm
0.981738683025684   européenne   69181.htm
```

tf_idf.txt

3.2. Analyse des valeurs de tf*idf

On observe le fichier **tf_idf.txt** trié en ordre. La valeur minimum de tf*idf est 0, qui correspond aux mots « et », « l », « de » et « à ». Ces mots seraient certainement ajoutées dans la stop liste car ces mots apparaissent dans tous les 326 fichiers html. De même, les mots très souvent utilisés tels que« de », «le », « la » etc seraient aussi enlevés depuis le fichier xml initial.

Comme notre fichier **tf_idf.txt** est trié en ordre croissant, on parcoure le fichier de haut en bas pour chercher un seuil de stop-liste. Parfois on a vu des mots qui ne sont pas des prépositions mais qui sont assez souvent utilisés dans les articles(tels que « recherche », « chercheur »). Dans ce cas-là on les ajoute également dans la stop-liste. On continue à parcourir le fichier jusqu'à l'apparition de mot « développement ». A notre avis cela est un mot qui peut être considéré comme un mot-clé. La valeur **idf*tf** de ce mot dans l'article est 0,406 et celle du mot au dessus est 0,402. On a donc pris la valeur au milieu qui est environ 0,405. On a rédigé un autre script Perl qui extrait les mots dont les valeurs tf*idf sont inférieures à un seuil, où le seuil prend la valeur 0.405.

Ensuite on a utilisé le script donné « **newcreeFiltre.pl** » en passant l'argument « **stoplist.txt** » pour créer un script de filtre. On utilise le script généré pour supprimer les mots de stoplist depuis le fichier XML.

Pour vérifier que l'application de la stoplist nous renvoyait bien un corpus privé des mots de cette liste, on a cherché à observer que des mots comme « le », « la », « un » et ils étaient supprimés du fichier en sortie.

3.3. Création des lemmes et analyse critique des résultats obtenus

3.3.1. Création des lemmes

Après avoir filtré le fichier XML initial de façon à en supprimer les mots de la stoplist, on veut construire une liste à deux colonnes contenant un mot de titre ou de texte et son lemme.

Premièrement on a utilisé la commande :

```
cat newcorpus.xml | perl segmente_TT.pl  |sort -u
```

pour découper le fichier corpus en tokens uniques puis les trier en ordre.

Ensuite on a exécuté le script successeurs.pl pour générer la liste des successeurs pour chaque lettre des mots et on l'a exporté vers un fichier nommé « succ.txt ».

Puis on exécute le script « filtronc_P16.pl » en passant le fichier « succ.txt » comme argument, on obtient finalement un fichiers de deux colonnes qui sont respectivement les mots de texte/titre et les lemmes.

abouti	abouti
aboutir	abouti
aboutira	abouti
aboutiront	abouti
aboutissement	abouti
aboutissez	abouti
aboutit	abouti

mot-lemme.txt

On lance le script « **newcreeFiltre.pl** » en passant l'argument « **lemme.txt** » pour générer un nouveau filtre. Puis on l'applique sur le fichier XML afin de remplacer les mots par ses lemmes et on obtient le fichier XML filtré « **corpus_lemme.xml** ».

3.3.2. Analyse des résultats obtenus

Le lemme d'un mot est trouvé en cherchant dans la liste des successeurs de ses lettres la plus longue sous-chaîne se terminant par le plus grand maximum précédé par un 1. Donc il est nécessaire que le minimum soit 1 et non pas un minimum local car cela nous donnerait un lemme trop général alors qu'il pourrait être plus précis.

Prenant l'exemple des mots de famille « sélection » :

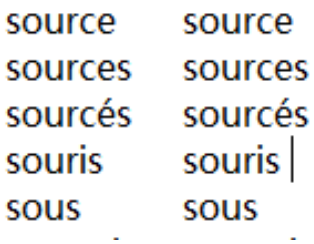
992111320	sélectif
9921113210	sélectifs
9921113120	sélection
992111312220	sélectionne
9921113122210	sélectionner
992111312220	sélectionné
9921113122210	sélectionnés
9921113210	sélective
992111321110	sélectivité
992111110	sélénium
9911111120	séminaire

succ.txt

sélectif	sélecti
sélectifs	sélecti
sélection	sélecti
sélectionne	sélecti
sélectionner	sélecti
sélectionné	sélecti
sélectionnés	sélecti
sélective	sélecti
sélectivité	sélecti
sélénium	sélénium
séminaire	séminaire

mot-lemme.txt

Il est évident que les mot « **sélectif** », « **sélection** », « **sélectionner** » appartient à la même famille et ils auront les même lemmes car ses chaînes correspondantes dans le fichier « succ.txt » sont très similaires. On peut donc constater que cela est un lemme de bon qualité. Et le résultat des lemmes obtenu dans le fichier « **mot-lemme.txt** » correspond bien à notre résultat attendu.



Ici on a un autre exemple qui montre quelques limites de notre algorithme, les mots « **source** » et « **sources** » n'ont pas le même lemme dans notre résultat obtenu et cela est différent de notre résultat estimé. Par contre si nous prenons le lemme "**sour**", le mot « **souris** » aura le même lemme que les mots de famille « **source** ». Et ces deux genres de mots expriment pas vraiment le même concept. Il faut donc proposer une méthode d'amélioration de l'algorithme afin de donner le lemme correct aux mots de même famille (comme le lemme «**sourc** » pour les mots de famille « **source** »), et les distinguer avec les autres mots similaires (comme le mot « **souris** »).

3.4. Génération des tables inverses selon le type

3.4.1. Fichier inverse sur la balise <date>

Dans les parties précédentes on a obtenu un fichier XML filtré « **corpus_lemme.xml** ». A l'aide de script donné « **index.pl** », on a créé les différents fichiers inverses en le passant l'argument « nom de la balise ».

Par exemple pour créer le fichier inverse sur la balise <date>, on utilise la commande :

```
cat corpus_lemme.xml| perl index.pl date
>fichier_inv_date.txt
```

Le fichier exporté est :

fichier_inv_date.txt

```
3   07  2013    73430.htm   282 73431.htm   282 73432.htm   282 73433.htm
29  11  2012    71612.htm   275 71614.htm   275 71615.htm   275 71616.htm
11  09  2012    70914.htm   273 70915.htm   273 70916.htm   273 70917.htm
21  12  2011    68638.htm   265 68639.htm   265 68640.htm   265 68641.htm
29  06  2012    70420.htm   271 70421.htm   271 70422.htm   271 70423.htm
7   08  2013    73683.htm   283 73684.htm   283 73685.htm   283 73686.htm
22  07  2011    67383.htm   259 67385.htm   259 67386.htm   259 67387.htm
26  03  2012    69533.htm   268 69534.htm   268 69535.htm   268 69536.htm
26  03  2013    72629.htm   279 72630.htm   279 72631.htm   279 72632.htm
20  06  2014    76206.htm   291 76207.htm   291 76208.htm   291 76209.htm
22  02  2012    69177.htm   267 69178.htm   267 69179.htm   267 69180.htm
5   05  2014    75788.htm   290 75789.htm   290 75790.htm   290 75791.htm
30  08  2011    67553.htm   260 67554.htm   260 67555.htm   260 67556.htm
24  01  2012    68881.htm   266 68882.htm   266 68883.htm   266 68884.htm
21  12  2012    71835.htm   276 71836.htm   276 71837.htm   276 71838.htm
29  09  2011    67794.htm   261 67795.htm   261 67796.htm   261 67797.htm
22  11  2011    68273.htm   263 68274.htm   263 68275.htm   263 68276.htm
3   06  2013    73182.htm   281 73183.htm   281 73184.htm   281 73185.htm
5   02  2014    75063.htm   288 75064.htm   288 75065.htm   288 75066.htm
20  03  2014    75457.htm   289 75458.htm   289 75459.htm   289 75460.htm
31  05  2012    70161.htm   270 70162.htm   270 70163.htm   270 70164.htm
23  12  2013    74744.htm   287 74745.htm   287 74746.htm   287 74747.htm
30  04  2013    72932.htm   280 72933.htm   280 72934.htm   280 72935.htm
10  09  2013    73875.htm   284 73876.htm   284 73877.htm   284 73878.htm
19  10  2011    67937.htm   262 67938.htm   262 67939.htm   262 67940.htm
1   12  2011    68383.htm   264 68384.htm   264 68385.htm   264 68386.htm
21  06  2011    67068.htm   258 67071.htm   258
19  04  2012    69811.htm   269 69812.htm   269 69813.htm   269 69814.htm
9   08  2012    70743.htm   272 70744.htm   272 70745.htm   272 70746.htm
28  02  2013    72392.htm   278 72393.htm   278 72394.htm   278 72395.htm
8   11  2012    71357.htm   274 71358.htm   274 71359.htm   274 71360.htm
31  01  2013    72113.htm   277 72114.htm   277 72115.htm   277 72116.htm
2   12  2013    74449.htm   286 74450.htm   286 74451.htm   286 74452.htm
6   08  2014    76507.htm   292 76508.htm   292 76509.htm   292 76510.htm
21  10  2013    74167.htm   285 74168.htm   285 74169.htm   285 74170.htm
```

fichier_inv_date.txt

3.4.2. Fichier inverse sur la balise <date>

De même, on a obtenu les différentes fichiers inverses en changeant l'argument de script :

fichier_inv_rubrique.txt

```
Focus   67068.htm   258 67383.htm   259 67553.htm   260 67554.htm   260 67555.htm
En direct des Laboratoires  72397.htm   278
Horizons Enseignement   68281.htm   263 68392.htm   264 68393.htm   264 68888.htm
Evénement   67392.htm   259 67561.htm   260 67803.htm   261 67804.htm   261 67945.htm
Horizon formation   71619.htm   275
Du côté des Pôles   69818.htm   269
Du côté des pôles   67388.htm   259 67799.htm   261 72396.htm   278 72634.htm   279
Actualités Innovations  69538.htm   268 75460.htm   289 76509.htm   292 76510.htm
Au coeur des régions    67798.htm   261 67943.htm   262 68279.htm   263 68387.htm
En direct des laboratoires  67389.htm   259 67390.htm   259 67391.htm   259 67558.htm
Horizon enseignement    73437.htm   282
Actualité Innovation    67071.htm   258 67385.htm   259 67386.htm   259 67387.htm
Actualité-Innovation    70425.htm   271
Actualités Innovation   73433.htm   282
Horizons Formation Enseignement 76211.htm   291
A lire  68283.htm   263 69821.htm   269 70429.htm   271 70753.htm   272 71366.htm
En direct des labos 75463.htm   289
Horizon Enseignement    67944.htm   262 69185.htm   267 70426.htm   271 70748.htm

```

fichier_inv_rubrique.txt

3.4.3. Fichier inverse sur les mots des textes/titres

On a d'abord créé un fichier qui contient un flux de données à l'aide de script « **segment_TT.pl** » car le script « **indexTexte.pl** » n'accepte que les arguments sous forme « lemme rubrique fichier numéro» .

La commande exécutée est :

```
cat corpus_lemme.xml |perl segmente_TT.pl -f -r -n| sort >flux.txt.
```

Ensuite on exécute le script « **indexTexte.pl** » en donnant l'argument « **flux.txt** », le résultat d'exécution est donc le fichier inverse des mots/lemmes des textes.

fichier_inv_date.txt | fichier_inv_rubrique.txt | tableinv.txt

```
1   démarrer        focus   71359.htm   274 focus   71359.htm   274 focus   72631.htm   279
2   volgograd       du côté des pôles   72396.htm   278
3   combinant       actualité innovation    71616.htm   275 au coeur des régions    71361.htm   274 en direct de
4   anonymes        focus   68383.htm   264 horizons enseignement    68281.htm   263
5   consist     actualité innovation    70424.htm   271 actualité innovation    70918.htm   273 actualité innova
6   parfum      au coeur des régions    70166.htm   270 evénement   67946.htm   262 evénement   76516.htm   292
7   flore       actualité innovation    69815.htm   269 en direct des laboratoires  68389.htm   264
8   achat       a lire  76213.htm   291 focus   67794.htm   261 focus   68274.htm   263 focus   68881.htm   266
9   habitude        du côté des pôles   72634.htm   279
10  navire      focus   70916.htm   273 focus   70916.htm   273
11  divisée     du côté des pôles   75793.htm   290 focus   68383.htm   264
12  concour     actualité innovation    69536.htm   268
13  travaillons     au coeur des régions    70745.htm   272 focus   72114.htm   277 focus   73876.htm   284 focu
14  instances       actualité innovation    68642.htm   265
15  mode        au coeur des régions    75068.htm   288 du côté des pôles   67388.htm   259 du côté des pôles
16  eurocopter      horizons enseignement   68393.htm   264
17  avionneur       actualité innovation    70920.htm   273 actualité innovation    71617.htm   275 actualité in
18  aucun       actualité innovation    68883.htm   266 actualité innovation    72935.htm   280 au coeur des rég
19  larf        au coeur des régions    68885.htm   266
20  clôtur      evénement   76514.htm   292
21  salzbourg       focus   69811.htm   269
22  dû      actualité innovation    70163.htm   270 au coeur des régions    71842.htm   276 en direct des labora
23  schwartz        evénement   70749.htm   272
24  économiste      a lire  74457.htm   286 focus   67068.htm   258 focus   68882.htm   266 focus   69533.htm
25  alcatel     actualité innovation    71839.htm   276
26  induisant       en direct des laboratoires  70746.htm   272
27  dusko       actualité innovation    69815.htm   269 actualité innovation    69815.htm   269 focus   73432.ht
28  rifler      au coeur des régions    68387.htm   264 au coeur des régions    68387.htm   264
29  têtes       du côté des pôles   72396.htm   278 focus   72933.htm   280
30  prouesse        actualité innovation    72934.htm   280 actualité-innovation    70425.htm   271 actualité-in
31  recensé     au coeur des régions    68279.htm   263
32  intensive       a lire  76213.htm   291 a lire  76213.htm   291 focus   70916.htm   273
33  syndicat        focus   74746.htm   287 horizon enseignement    67944.htm   262
34  succède     focus   72113.htm   277
35  marcellan       en direct des laboratoires  74750.htm   287
36  puisse      actualité innovation    67556.htm   260 actualité innovation    70163.htm   270 au coeur des rég
37  algorithme      en direct des laboratoires  68391.htm   264
38  anantharaman        focus   72932.htm   280 focus   72932.htm   280 focus   72932.htm   280 focus   72932.ht
39  pérenn      a lire  71366.htm   274 au coeur des régions    75068.htm   288
40  dépourvu        actualité innovation    68883.htm   266 en direct des laboratoires  68388.htm   264
```

tableinv.txt

Pour vérifier la qualité de fichier inverse, on a regardé si les mots, par exemple le mot « démarrer », se trouve bien dans la rubrique et l'article indiqué dans le fichier inverse, ce qui s'avère être le cas pour les quelques mots testés.