LO17
Rapport indexation

ASCHENBRENNER Céline MARECHAL Anaig

17 avril 2016

Table des matières

Résumé

Le but du projet de TD est de construire étape par étape un système complet d'indexation et d'interrogation en langage naturel d'une archive de bulletins de veille sur la France entre 2011 et 2014. Ce rapport concerne uniquement la partie d'indexation. En ce début de semestre nous avons donc commencé par transformer ces 326 bulletins en un document XML structuré, afin de préparer leur future indexation. Nous avons ensuite indexé ces fichiers en créant un ensemble de fichiers inverses qui contiennent le lexique du corpus et ses différentes affectations. Entre temps, ces fichiers ont été traités pour que l'indexation soit réellement pertinente.
En résumé, le défi relève donc ici à récupérer un corpus de bulletins présentés en format XML et à n'en garder que le lexique nous semblant important pour les futures recherches des utilisateurs. Il s'agit également de se constituer un dictionnaire qui nous servira de base de données et nous permettra d'associer rapidement un mot à un bulletin lors d'une interrogation.

1 Préparation du corpus

Pour générer le fichier XML qui synthétise chaque bulletin, nous avons procédé étape par étape.

1.1 Analyse des fichiers HTML

Tout d'abord nous avons étudié le contenu des différents documents HTML afin d'en comprendre la structure globale. Ainsi, chaque article contient des méta-informations telles que le numéro du bulletin dans lequel il se trouve, sa date de parution, son titre et sa rubrique. Ces méta-informations sont suivies par le contenu de l'article qui est constitué de texte et peut également contenir des images, éventuellement légendées. On distingue deux types d'images différentes.
En observant quelques bulletins dans le navigateur, il nous semble que les articles présentent tous globalement la même structure HTML : les différentes méta-informations ont chacune leur propre positionnement, couleur et taille de police. Cela nous met donc sur la piste que leur structure va nous être utile pour les distinguer.

FIGURE 1 – Structure d'un article

1.2 Choix des balises HTML de structure

Pour chaque élément à indexer (rubrique, date, ...) nous avons alors déterminé quelle(s) expression(s) régulière(s) permettait/aient d'identifier de façon certaine son contenu. Pour cela, on regardait dans notre fichier quelles sont les balises entourant l'élément nous intéressant. Généralement, nous retenions les classes CSS qui contenaient un style. On recherchait alors à chaque fois pour un échantillon de fichiers représentatifs :

— Si la balise est bien *unique*, c'est-à-dire qu'en recherchant son motif la seule information que l'on peut trouver est bien l'élément recherché, et aucun autre.

— Si la balise est bien *exhaustive* c'est-à-dire qu'on va pouvoir retrouver l'élément recherché dans tous les fichiers.

Bien sûr, tester ces critères sur un échantillon seulement ne peut pas garantir que notre choix de balise sera parfait pour tous les fichiers, et c'est ici la difficulté. En lançant notre script pour retrouver chaque élément, on vérifiait donc bien que toutes les balises étaient présentes et non vides grâce à un script de test.

Les balises que nous avons choisi devaient à la fois être assez longues et complètes pour garantir l'unicité de la chaîne dans le fichier, mais dans un autre temps ne pas contenir trop d'éléments qui nous feraient prendre le risque de ne pas être présents dans tous les éléments du corpus.

1.3 L'extraction

Nous avons donc écrit un script Perl qui procédait à l'extraction des éléments pour un fichier grâce à notre choix de balises HTML.

Au niveau du traitement des balises, certaines étaient standard (rubrique, titre, ...), et d'autres nécessitaient un traitement particulier. Ainsi nous avons dû récupérer le nom du fichier à partir de la commande qui lançait notre script à l'aide de ARGV. Une fois le nom du fichier récupéré, une boucle parcourait l'ensemble du fichier et recherchait les expressions régulières correspondant aux balises HTML grâce à des conditions "if". Pour le texte nous avons lu toutes les lignes que nous avons placé dans une même variable dans le but de n'obtenir plus qu'une ligne, ce qui facilite les traitements ultérieurs. Pour ce faire, nous avons utilisé un "do ... until" qui concatène chaque ligne les unes à la suite des autres jusqu'à la fin du texte. Nous avons alors supprimé toutes les balises
 du contenu, qui ne nous intéressent pas pour notre corpus.

Nous avons par la suite rencontré quelques problèmes pour récupérer les éléments images et légendes contenues dans le texte. Pour ce faire, nous parcourions la variable de texte grâce à une boucle "while" en utilisant l'option g pour effectuer une substitution globale. Le problème était du aux retours chariots qui faisaient que nos expressions régulières n'étaient pas appliquées sur la ligne entière. Nous avons donc pré-traité le texte avant de rechercher les images et légendes en supprimant tous les caractères "\n", "\r" et "\f".

Nous avons ensuite écrit un autre script Perl appelant le script Perl d'extraction sur chacun des fichiers contenus dans BULLETINS grâce à un "foreach". Enfin, un script Shell lance ce script et lui applique *convert.pl* pour que le corpus en sortie soit au format UTF8.

1.4 Obtention du corpus

Grâce au script d'extraction, nous avons pu obtenir un fichier XML corpus. Nous avons alors voulu vérifier s'il était bien formé.

Tout d'abord nous l'avons simplement ouvert dans un navigateur, ce qui nous a permis de voir si le fichier s'affiche bien à vue d'oeil, de corriger les erreurs d'indentations et de vérifier que toutes les balises sont bien fermées.

Le fichier obtenu prend la forme :

```
<corpus>
 <bulletin>
    <fichier>Nom du fichier</fichier>
    <numero>Numéro du bulletin</numero>
    <date>Date du bulletin au format jj/mm/aaa</date>
    <rubrique>Rubrique de l'article</rubrique>
    <titre>Titre de l'article</titre>
    <texte>Texte de l'article</texte>
    <images>
    (S'il n'y a pas d'images cette balise reste vide.)
       <image>
           <urlImage>URL de l'image</urlImage>
           <legendeImage>la légende de l'image</legendeImage>
           (Certaines images n'ont pas de légendes)
       </image>
       <image>
            ...
       </image>
            ...
       </images>
        <contact>Les informations de contact</contact>
    </bulletin>
    <bulletin>
         ...
    </bulletin>
         ...
</corpus>
```

La DTD correspondante est :

```
<!ELEMENT corpus (bulletin+)>
<!ELEMENT bulletin (fichier, numero, date, rubrique, titre, contact, images, texte)>
<!ELEMENT fichier (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT rubrique (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT contact (#PCDATA)>
<!ELEMENT texte (#PCDATA)>
<!ELEMENT images (image*)>
<!ELEMENT image (urlImage, legendeImage?)>
<!ELEMENT urlImage (#PCDATA)>
<!ELEMENT legendeImage (#PCDATA)>
```

Après test, le corpus XML est valide avec la DTD, donc on peut considérer qu'il est bien formé. Nous avons également créé un script de test qui compte le nombre de différentes balises.

Lorsque des tests supplémentaires étaient possibles nous les avons effectué. Ainsi nous avons vérifié le format de la date (date bien entre 2011 et 2014), le nom du fichier (5 chiffres, possibilité avec ou sans le .htm) le numéro (entre 200 et 299), et les url des images avec des regex les plus précises possibles. De même pour les rubriques, comme elles sont en nombre fini, nous en avons fait une liste exhaustive pour la regex. Cette technique est cependant plus compliquée à mettre en oeuvre pour un corpus plus

```
amarecha-hz-1:TD1 ana$ perl log_script.pl corpus.xml
Dans notre corpus...:
Il y a 326 bulletins.
Il y a 326 dates.
Il y a 326 fichiers.
Il y a 326 numéros de bulletin.
Il y a 326 rubriques.
Il y a 326 textes.
Il y a 155 images.
Il y a 155 urls d'images.
Il y a 134 legendes d'images.
Il y a 326 contacts.
```

FIGURE 2 – Test : Nombre d'éléments dans notre corpus

grand, car il y a un certain nombre de divergences d'orthographe dans les rubriques qui impactent sur le temps nécessaire pour les référencer.

```
if (/<date>[0-9]{1,2}\/[0-9]{2}\/201[1-4]</) {
    $date++;
}
if (/<fichier>[0-9]{5}(.htm)?</) {
    $fichier++;
}
if (/<numero>2[0-9]{2}</) {
    $numero++;
}
if (/<rubrique>((Actualité(s?)[ -][Ii]nnovation(s?)) |
    (Focus)|(En direct des [Ll]aboratoires) |
    (En direct des labos)|(Evénement) |
    (Du côté des [pP]ôles) |
    (Au coeur des [Rr]égions) |
    (Horizon(s?) [eE]nseignement)|
    (A lire)|(Horizon(s?) [Ff]ormation(s?)) |
    (Horizons Formation Enseignement))</) {
    $rubrique++;
}
```

FIGURE 3 – Exemples d'expressions régulières pour vérifier le contenu des balises XML du corpus.

2 Indexation du corpus

Pour pouvoir réaliser les fichiers inverses, nous sommes d'abord passées par des étapes intermédiaires. Tout d'abord, nous avons construit une liste de mots non pertinents lors d'une recherche (pronoms, trop généraux, sans apport d'information,...) à partir du corpus initial. Pour cela nous nous sommes appuyées sur leur fréquence par document.

2.1 Unité documentaire choisie

Pour construire notre liste de mots non-pertinents, il faut tout d'abord pouvoir distinguer les mots qui n'apportent rien de significatif des mots qui vont intéresser un utilisateur lors d'une recherche. Afin de distinguer ces deux types de mots, on va déterminer un seuil de sélection qui nous indiquera si un mot doit être conserver ou non.
Pour nous aider dans notre recherche du seuil de sélection, on s'appuie sur la méthode de pondération par tf x idf. Basée sur la fréquence d'apparition des termes dans le document et dans l'ensemble des documents, cette méthode permet de ne pas apporter trop d'importance aux mots très communs, qui apparaissent de nombreuses fois mais ne sont alors pas forcément pertinents pour notre recherche et les mots très rares qui ne vont rien apporter non plus. Trois éléments importants vont nous permettre de calculer le tf x idf :

- df_i qui représente la fréquence du terme i dans les documents ;
- idf_i qui correspond au degré d'information du terme i dans le corpus. Il se calcule ainsi : $idfi = log_{10}\frac{N}{df_i}$ avec N le nombre de documents dans le corpus ;
- tf_{ij} qui représente la fréquence du terme i dans le document j ;

Ainsi un mot très fréquent dans les documents aura un logarithme de presque 1, et donc un coefficient idf_i proche de 0. Ce mot aura un poids faible.
Un mot très rare en revanche aura un logarithme de n avec n grand. Le coefficient augmente donc et le mot malgré sa faible apparition est mis en valeur.

Dans notre projet, le choix qui nous a paru le plus pertinent est qu'un document corresponde à un article. En effet un article traite un sujet particulier, et les mots correspondant à ce sujet risque donc d'avoir une fréquence importante dans l'article, mais une fréquence moindre dans le bulletin qui correspond à plusieurs articles de sujets différents.
Si par hasard deux articles traitant de sujets similaires apparaissaient dans le même bulletin, la fréquence (tf) des mots correspondants au sujet sera plus grande pour ce document, mais ils apparaitront dans moins de documents différents donc leur df, sera plus faible, et donc leur idf plus important. Ainsi le produit des deux sera plus élevé et par conséquent les mots concernés seront considérés comme plus pertinents que si ces articles avaient été dans deux bulletins différents alors que ce n'est pas forcément le cas.
En revanche pour calculer le coefficient tf x idf cela n'aurait pas forcément été plus compliqué. En effet nous avons utilisé l'option -f du script *segmente.pl* pour classer les mots par fichier (=article) mais nous aurions obtenu la même chose avec les bulletins avec l'option -n.

2.2 Pondération par tf x idf

Nous avons commencé par utiliser le script *segmente.pl* qui permet de lister tous les mots du corpus. Nous avons ajouté l'option -f qui affiche le nom du fichier de provenance en face de chacun des mots. Nous avons ensuite trié ce fichier par mot, et compté combien de fois chaque mot apparaissait dans chaque fichier, grâce à l'option -c sur un uniq, pour obtenir la fréquence.

Pour calculer le df, on va aussi lancer le script *segmente.pl*, puis trier par mot pour avoir pour chaque mot la liste des fichiers dans lesquels il apparait. On lance un uniq pour que chaque fichier n'apparaisse qu'une seule fois pour un mot. On n'a plus qu'à comptabiliser le nombre de fichiers par mot pour obtenir le df.

Pour calculer l'idf on a repris la formule $idfi = log_{10}\frac{N}{df_i}$, grâce aux dfs obtenues précèdemment.

Pour calculer le tf x idf nous avons utilisé un tableau associatif. Nous avons fait un script qui parcourait le fichier des idf, et ajoutait dans le tableau associatif le mot comme clé et son idf comme valeur. Ensuite le script parcourait le fichier des tf et pour chaque mot récupérait son df dans le fichier et son tf dans le tableau associatif et imprimait le produit des deux.

2.3 Création de la stop-list

Pour la stop-list nous avons fait plusieurs choix.
Tout d'abord, concernant les nombres, nous avons décidé de garder uniquement ceux correspondant à des années récentes (entre 1900 et 2050). En effet, nous pensions que des requêtes pouvaient être effectuées sur des dates de découverte, mais que les autres nombres avaient peu de chance d'être pertinents lors d'une requête. Concernant les mots, nous avons calculé la moyenne de leur tf x idf. Pour cela, nous avons créé un script Perl qui fait la somme de tous les mots identiques se succédant, puis en cas de changement de mot, divise cette somme par le nombre de mots identiques parcourus. Cette méthode de moyenne nous paraissait pertinente car ainsi si un mot a un poids fort pour un article, cela aurait un plus grand impact avec la moyenne qu'avec la médiane par exemple.

2.4 Recherche du seuil de sélection

Pour définir le seuil en dessous duquel un mot serait ajouté à la stop list, nous avons rangé les mots par ordre croissant de leur coefficient.
Ensuite nous avons parcouru la liste mot par mot. A chaque fois que nous trouvions un mot qui nous paraissait pertinent, nous regardions s'il n'avait pas un mot de la même famille et qui aurait donc le même lemme avec un coefficient plus élevé. Lorsque c'était le cas nous l'ignorions. Le premier mot pertinent et sans mot de la même famille que nous avons trouvé est "CNRS" de coefficient 0.95. Cette valeur était donc notre seuil pour établir notre stop-list.
Cette méthode de sélection du premier mot pertinent, bien que nous l'ayons adoptée ici, aurait pu être fastidieuse pour un corpus plus important. On aurait pu alors simplement s'en tenir au premier mot nous semblant important, ou encore, si le nombre de mots est si grand qu'il est long d'en chercher le premier mot important, on aurait pu ne sélectionner que le premier quartile pour la stop-list par exemple. Cette méthode aurait alors été moins arbitraire mais on aurait pu perdre des mots importants.

Nous avons préféré mettre un seuil relativement bas ; en effet, de notre point de vue il paraissait préférable que des mots non pertinents restent dans le corpus plutôt que des mots pertinents en soient supprimés.
Nous avons ensuite généré le script pour éliminer les mots de notre stop-list du corpus, puis nous avons généré la liste des successeurs et crée le script correspondant grâce à *filtronc.pl*. Nous l'avons alors appliqué à nouveau au corpus pour remplacer tous les mots par leur lemme.

2.5 Lemmatisation

Concernant les lemmes obtenus, le résultat n'est pas optimal. Dans certains cas des mots de la même famille ont des lemmes différents. Par exemple validé, validée, et validés ont pour lemme eux-

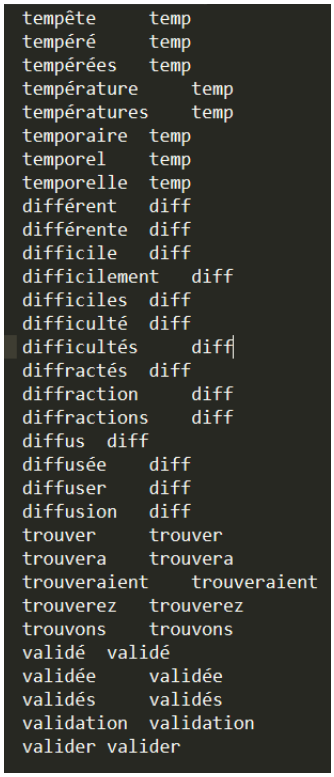

FIGURE 4 – Sélection de mots avec l'algorithme filtronc.pl original

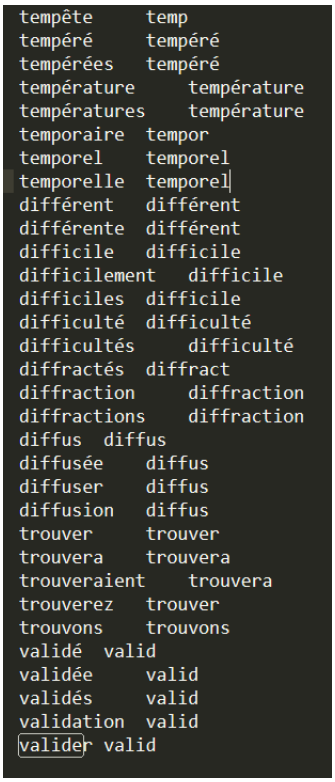

FIGURE 5 – Sélection de mots avec l'algorithme filtronc.pl après modifications

même alors qu'ils ont la même signification, de même pour toutes les conjugaisons du verbe trouver. Au contraire, temporaire, température, temps et tempête ont le même lemme alors que toutes leurs significations sont différentes. Difficile, diffraction, diffusion, et différent ont aussi le même lemme. Toutefois ces résultats incohérents sont rencontrés rarement au vu de du grand nombre de mots lemmatisés. Celle-ci permet de faire une indexation cohérente malgré ces erreurs. L'algorithme de*filtronc.pl* pouvait être en revanche amélioré. Pour cela nous avons fait une liste de successeurs de test, contenant les mots problématiques évoqués plus haut. Pour corriger l'erreur sur les verbes et la fin des mots qui varie, nous avons fait le choix de relever le maximum acceptable. Si les valeurs entre 1 et 3 sont acceptées comme prédecesseur d'une autre valeur, les résultats obtenus sont meilleurs même si non optimaux. En effet toutes les déclinaisons de valider donneront bien valid, mais trouveraient donne encore trouvera au lieu de trouver. Il ne nous a pas sembler pertinent de relever encore plus cette limite car cela doit déjà induire d'autres erreurs. Concernant les mots différents ayant la même racine, nous avons fait un test, en enlevant la notion de limite qui s'incrémente au fur et à mesure, l'algorithme s'arrêtant juste au bout d'un certain nombre de cycle. Si les valeurs entre deux et neuf sont toujours acceptées, alors les mot selectionnés retrouvent un lemme qui leur correspondent. vraiment. Lorsque ces modifications sont appliquées sur tous les successeurs, la liste obtenue semble plus précise, cependant si cela induit un nombre de lemmes plus important, on risque de perdre certaines informations pertinentes.

D'autres algorithmes auraient toutefois pu être appliqués autre que la lemmatisation, comme le stemming, qui lui utilise un procédé heuristique pour retirer la fin des mots. Il s'agit de différentes règles qui enlèvent par exemple les suffixes connus. Il contient aussi des exceptions pour tous les cas connus où ces règles générales ne s'appliquent pas. L'algorithme de Porter (Stemming) aurait donc

aussi pu être appliqué dans le cas présent.

2.6 Création des fichiers inverses

Afin de créer les fichiers inverses à partir de notre fichier XML contenant le corpus filtré, nous avons utilisé les scripts index.pl et indexText.pl.
Le premier script va permettre, pour chaque balise, d'associer un fichier à des métadonnées. Ainsi, pour les balises titre, urlImage, contact, date, legendeImage, rubrique et texte nous avons créé un fichier inverse qui associe le contenu de la balise entrée en paramètre avec tous les fichiers dans lequel il apparaît, ainsi que le numéro du bulletin associé.
Le deuxième script permet d'associer à un mot du flux de données sa rubrique, son fichier et le bulletin dans lequel il se trouve.
Ainsi, ces fichiers inverses vont par la suite pouvoir nous servir de correspondance entre un mot clé et un fichier dans lequel il se situe, ce qui va nous permettre d'y accéder rapidement lors d'une recherche.

3 Conclusion

En conclusion, malgré le fait que nous nous sommes appuyées sur un protocole rigoureux, certains choix restent subjectifs. En effet concernant le choix du seuil pour la stop-list, même si nous avons effectué différents calculs pour trouver la valeur seuil, elle aurait très bien pu être différente. Cela concerne aussi la lemmatisation, qui est aussi sujette à son algorithme, dont des améliorations sont toujours possibles.
Par ailleurs, la stop list aurait aussi pu être plus exhaustive. En effet, certains mots, des verbes par exemple, ont un poids moindre car ils apparaissent de nombreuses fois mais sous des déclinaisons différentes, ce qui diminue leur fréquence. Refaire un filtre après la lemmatisation aurait donc pu être pertinent pour rajouter des mots à la stop-list. Cependant nous ignorons si ce phénomène est réellement récurrent ou au contraire négligeable compte tenu de notre étude.
Travailler avec les synonymes aurait également pu être très intéressant dans le cadre de ce TD. En effet, nous aurions ainsi pu regrouper les mots par leurs différentes variantes et ainsi avoir un tf idf qui les prenne en compte. Enfin, bien que des améliorations semblent encore possible, les résultats obtenus nous semblent de bonne qualité.

Ce projet nous a permis de nous familiariser avec les expressions régulières, le langage Perl ainsi que d'utiliser un mode de réflexion qui diffère de nos habitudes. De plus travailler sur un grand nombre de fichiers a été une expérience intéressante, car nous devions effectivement contrôler chacun de nos scripts grâce à des tests, pour vérifier qu'ils s'adaptaient bien à chaque spécificité possible dans un si grand ensemble de fichiers, chose qui demandait une analyse plus approfondie qu'une simple approche de surface.