

Introduction

L'extraction terminologique consiste en l'indentification des termes potentiels, à partir d'un corpus. Dans cet article, nous présentons les résultats de l'implémentation d'une méthode d'extraction semi-automatique de termes à partir d'un corpus. L'apport de ce type d'approche à la terminologie classique est considérable. En effet, la combinaison de connaissances linguistiques et de connaissances informatiques apporte un gain de temps considérable au terminologue.

Dans la suite de cet article, nous présentons, en premier lieu, un état de l'art des différentes approches ayant trait à l'extraction terminologique (section 1), puis nous présentons notre corpus de travail (section 2). Après une description de notre méthodologie (section 3), nous présentons nos résultats d'expérience (section 4). Enfin, nous concluons par une discussion des résultats et les perspectives de recherche (section 5).

1 État de l'art

L'ère de la terminologie classique wüsterienne est révolu comme le souligne Monique Slodzian dans son article magistral intitulé : « *Émergence d'une terminologie textuelle* »⁴³. Slodzian tient à rappeler que depuis les années 1970, l'apport de l'informatique à la terminologie s'est limité aux systèmes de bases de données relationnelles dans le but de traiter et de stocker les termes, le tout en respectant scrupuleusement les préceptes wüsteriens. En effet, la lexicographie spécialisée, l'ingénierie documentaire et l'intelligence artificielle : « font de plus en plus appel aux techniques de la linguistique de corpus en vue de l'extraction semi-automatique de termes et de contextes. Ce virage méthodologique a créé une onde de choc qui ébranle les fondements de la doctrine wüsterienne »⁴⁴. L'extraction semi-automatique de termes se trouve donc au cœur de cette nouvelle terminologie « textuelle ». Bien entendu, quelque soit la méthode employée pour extraire les termes, la démarche est résolument semi-automatique, car la liste de candidats termes qui est générée « requiert en dernière instance la sélection finale des termes par les experts »⁴⁵. La phase de nettoyage de ces listes est donc nécessaire pour ne garder que les termes pertinents.

Pour réaliser une extraction terminologique, différents outils peuvent être employés, lesquels correspondent à trois grandes approches :

- Approche statistique.
- Approche linguistique (symbolique).
- Approche mixte.

⁴³ Slodzian, 2000.

⁴⁴ Ibid., p. 61.

⁴⁵ Ibid., p. 75.

1.1 Approche statistique

Cette approche est basée sur la distribution d'une unité terminologique dans un texte (fréquence et probabilité). Cette méthode est portable, facile à implémenter et indépendante des langues. Majoritairement, les méthodes statistiques sont basées sur le calcul de valeurs numériques. Elles sont fondées sur le fait que les termes apparaissent fréquemment dans les textes spécialisés, des mots simples qui apparaissent fréquemment ensemble sont forcément significatifs. Il s'agit alors d'exploiter différents types de calculs. Ainsi, si des mots simples apparaissent plus de n fois ensemble dans un texte, la suite est significative (calcul de segments répétés). Ou si un mot X apparaît plus fréquemment dans l'entourage d'un mot Y qu'ailleurs dans le texte, alors X et Y forment une combinaison significative (calcul de l'information mutuelle).

Cette approche génère du silence. Les termes à fréquence élevée sont repérés, mais les termes peu fréquents (happax) sont souvent omis. En effet, bien qu'il existe de nombreux travaux sur le filtrage statistique de candidats termes extraits à partir de corpus, l'expérience montre qu'aucune mesure statistique ne peut suppléer l'expertise de l'analyste, en particulier parce qu'il y a toujours des candidats termes de fréquence 1 dont l'analyse est intéressante.

1.2 Approche linguistique

Cette approche est basée uniquement sur des patrons de formation des termes complexes. Elle consiste à filtrer les termes en fonction de leur classe grammaticale (classe de mot) et chercher les relations sémantiques dans un texte. Elle est plus précise que la première approche et génère moins de bruit. Les stratégies linguistiques se fondent avant tout sur le fait que les termes complexes sont des syntagmes nominaux composés de suites de catégories grammaticales régulières (Nom Adj, Nom prep Nom, Nom Nom). Des séquences de mots correspondant à des patrons préalablement définis sont donc repérées.

Exemple : intelligence artificielle/robinet de commande /traitement de la demande/machine à coudre/imprimante laser /rayon laser/.

Deux méthodes d'extraction se présentent alors à l'analyste :

- « En positif » : méthode classique par patrons de termes (Nom Adj, Nom Prep Nom, Nom Nom etc.)
- « En négatif » : par patrons de frontières (repérage de marqueurs de frontières entre les groupes nominaux).

Le pré requis à ce type d'analyse est la phase d'étiquetage morphosyntaxique, qui consiste en l'attribution, à chaque mot, de sa catégorie grammaticale, de son lemme ainsi que certaines propriétés morphologiques (genre, nombre, temps, mode, personne).

1.3 Approche mixte

Il est possible de fusionner les deux précédentes approches pour augmenter la précision et évacuer tous les termes qui ne sont pas pertinents. Dans cette optique, des filtres statistiques sont appliqués aux candidats termes pour améliorer les résultats de l'approche linguistique :

- fréquence d'occurrence ;
- listes d'exclusion ;

- présence de chiffres.

À quelques rares exceptions, les outils d'extraction de termes combinent deux méthodes. On parle alors d'extraction faisant appel à des stratégies hybrides ou mixtes : les extracteurs génèrent une liste de candidats termes à partir d'informations linguistiques (patrons) et épurent cette liste au moyen de calculs statistiques; soit, au contraire, ils établissent une première liste de termes au moyen de calculs sur des chaînes de caractères et exploitent par la suite de l'information linguistique. Par exemple, Acabit⁴⁶ recherche des suites de parties du discours (POS) dans des textes préalablement étiquetés⁴⁷ (connaissances linguistiques) et fait des calculs statistiques sur les termes préalablement extraits (approche probabiliste). Cet outil effectue une analyse linguistique afin de transformer les termes nominaux en termes binaires qui sont triés selon des mesures statistiques.

2 Présentation du corpus

Notre point de départ est un corpus composé d'un an de fils RSS du journal Le Monde 2008 au format XML. Ces flux ont été récupérés tous les jours à 19h

Dans un premier temps, nous filtrons et nous nettoyons des flux RSS. Ceci nous permet d'extraire des parties textuelles qui se trouvent dans la balise <description> du flux RSS des rubriques « Technologies et Médias ». Cette première étape est réalisée grâce au langage de programmation Perl couplé aux expressions régulières (REGEXP).

Les fils ou flux RSS se situent dans une arborescence qui commence par une racine : 2008. Nous avons ensuite les mois et les jours. Dans chaque dossier jour, nous avons un dossier 19-00-00 (19h) qui indique l'heure à laquelle le flux a été récupéré. Enfin, dans chaque dossier 19-00-00, nous trouvons tous les flux qui sont codés. Ces flux sont accessibles au format xml et txt.

Voyons à présent le code de chaque rubrique :

Code = '0,2-3208,1-0,0.xml' = rubrique 'A la une'.

Code = '0,2-3476,1-0,0.xml' = rubrique 'Cinéma'.

Code = '0,2-3246,1-0,0.xml' = rubrique 'Culture'.

Code = '0,2-3234,1-0,0.xml' = rubrique 'Economie'.

Code = '0,2-3244,1-0,0.xml' = rubrique 'Environnement et Sciences'.

Code = '0,2-3214,1-0,0.xml' = rubrique 'Europe'.

⁴⁶ Daille, 1994.

⁴⁷ Treetagger, Cordial.

Code = '0,2-3404,1-0,0.xml' = rubrique 'Examens'.

Code = '0,57-0,64-987718,0.xml' = rubrique 'Fil municipales et cantonales 2008'.

Code = '0,2-3210,1-0,0.xml' = rubrique 'International'.

Code = '0,2-3260,1-0,0.xml' = rubrique 'Livres'.

Code = '0,2-3236,1-0,0.xml' = rubrique 'Médias'.

Code = '0,2-3232,1-0,0.xml' = rubrique 'Opinion'.

Code = '0,57-0,64-823353,0.xml' = rubrique 'Politique'.

Code = '0,2-3238,1-0,0.xml' = rubrique 'Rendez vous'.

Code = '0,2-3224,1-0,0.xml' = rubrique 'Société'.

Code = '0,2-3242,1-0,0.xml' = rubrique 'Sports'.

Code = '0,2-651865,1-0,0.xml' = rubrique 'Technologies'.

Code = '0,2-3546,1-0,0.xml' = rubrique 'Voyages'.

Nous avons choisi d'analyser les rubriques « Technologie » et « Médias ». Dans les paramètres de nos scripts de filtres, nous devons donc préciser les codes correspondants.

3 Description de notre méthodologie

Dans un premier temps, nous filtrons et nous nettoyons des flux RSS (étape 1). Ceci nous permet d'extraire des parties textuelles qui se trouvent dans la balise <description> du flux RSS des rubriques « Technologies » et « Médias ». Cette première étape est réalisée grâce au langage de programmation PERL couplé aux expressions régulières (REGEXP).

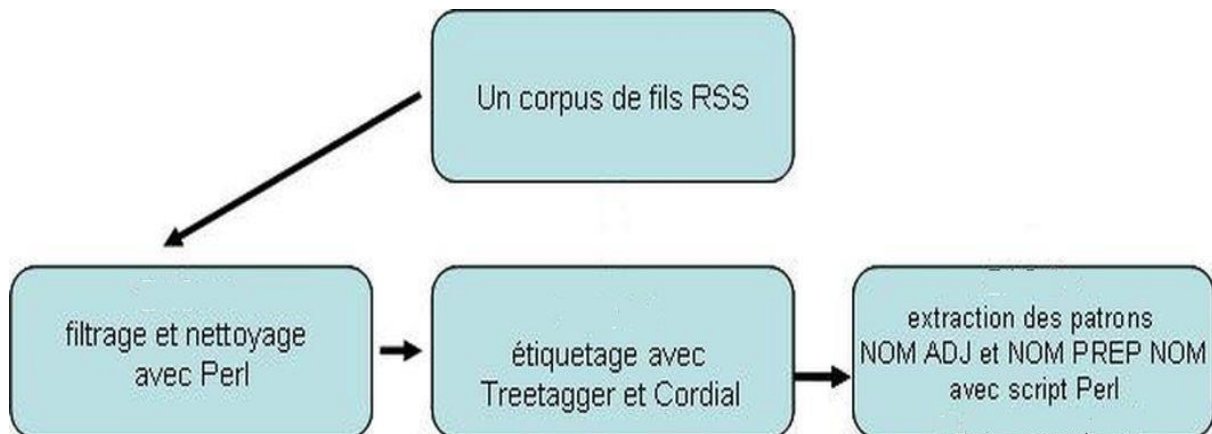
Nous appliquons ensuite un étiquetage morpho-syntaxique (étape 2) à ce contenu textuel à l'aide d'outils spécifiques (Treetager, Cordial), afin de générer des suites composées de trois colonnes :

- partie du discours ;

- lemme ;
- token.

Enfin, nous identifions dans ces suites, des patrons syntaxiques préalablement établis. Nous obtenons alors une liste de candidats termes qu'il est possible d'épurer et de filtrer à l'aide de paramètres statistiques simples.

Nous présentons, ci-dessous, un schéma qui décrit les étapes de ce projet :



3.1 Extraction du contenu de la balise <description>

Dans la première étape, nous allons présenter le script qui nous a permis d'extraire, de filtrer et de nettoyer le contenu de la balise <description>. Mais avant d'obtenir le script final, nous sommes passés par plusieurs étapes intermédiaires que nous allons décrire dans ce qui suit.

Les scripts PERL

1- Le filtreur (phase 1)

```

#!/usr/bin/perl
open(FILEINPUT,"$ARGV[0]");
while ($ligne = <FILEINPUT>){
if ($ligne=~/REGEXP/) {
print $ligne;
}
}
  
```

```
close(FILEINPUT) ;
```

Dans ce premier script, il s'agit d'identifier les chaînes de caractères qui contiennent l'expression régulière que l'on recherche. Pour ce faire, on ouvre un fichier en lecture. Ensuite, on utilise une boucle while pour lire le fichier. Tant qu'il y a une ligne à lire, on entre dans la structure de contrôle "if". Dans la condition de cette structure de contrôle "if", on doit spécifier le motif de l'expression régulière. Et l'instruction de cette boucle représente l'impression de la ligne qui contient cette expression régulière.

2- Le nettoyeur

Premier script :

```
#!/usr/bin/perl
open(FILEINPUT,"$ARGV[0]");
while ($ligne = <FILEINPUT>){
$ligne=~s/RECHERCHE/REMPLACEMENT/g;
print $ligne;
}
close(FILEINPUT);
```

Dans ce deuxième script, il s'agit de nettoyer les lignes qu'on obtient après avoir lu le fichier avec la boucle while. La ligne en gras montre qu'on remplace l'élément qu'on recherche par un autre. Par ailleurs on remarque l'utilisation du modificateur g, qui permet d'effectuer une recherche globale de toutes les occurrences.

3- Le filtreur (phase 2)

```
#!/usr/bin/perl
open(FILEIN,"$ARGV[0]"); # on ouvre le fichier qu'on va lire
open(FILEOUT, ">resultatsMODIF.txt"); # on ouvre le fichier sur lequel on va écrire
$i=1; # compteur qui sert à numérotor les lignes filtrées
while ($ligne = <FILEIN>){ # tant qu'il y a qqch à lire, on rentre dans la boucle, le perl ne
fonctionne que par ligne
if ($ligne=~<description>([^\<]+)<\>description>){ # si on trouve ce motif, on entre dans la
boucle
print FILEOUT "contenu de la ligne n°$i : $1\n";
$i++; # on incrémente le compteur
}
}
```

```
close(FILEIN);
close(FILEOUT);
```

Dans ce troisième script, on remarque, en gras, plusieurs modifications. En effet, le filtreur (phase 2) comporte une ligne qui permet d'ouvrir un fichier en écriture. Dans les précédents scripts, l'impression du résultat se fait directement sur la ligne de commande. Par ailleurs, on remarque l'ajout d'un compteur pour numéroté les lignes.

Le filtreur (phase 2) permet aussi d'imprimer les lignes sans les deux balises <description> (ouvrante et fermante). Pour ce faire, on déspecialise le slash "/" de la balise fermante </description> grâce au métacaractère anti-slash "\". Ensuite, on récupère le motif qui est mis entre parenthèses grâce à une expression régulière et on l'affecte à la variable \$1. Celle-ci est, ensuite, imprimée dans le fichier qui a été ouvert en écriture.

4- Le filtreur-nettoyeur

```
#!/usr/bin/perl
open(FILEIN,"$ARGV[0]");
open(FILEOUT, ">filtpropres.txt");
$i=1; #compteur qui sert à numertorer les lignes filtrées et nettoyées
while ($ligne = <FILEIN>){
if ($ligne=~<description>([^\<]+)<\description>){
my $propre=$1;
$propre=~s/&#233;/é/g;
$propre=~s/&#234;/ê/g;
print FILEOUT "contenu de la ligne n°$i : $propre\n";
$i++;
} } close(FILEIN); close(FILEOUT);
```

Dans ce quatrième script, il s'agit, tout simplement, de fusionner le nettoyeur avec le filtreur (phase 2).

5- Parcours arborescence

```
#!/usr/bin/perl
#-----
my $rep="$ARGV[0]";
# on s'assure que le nom du répertoire ne se termine pas par un "/"
$rep=~ s/[V]$//;
# on initialise une variable contenant le flux de sortie
```

```

my $DUMPFULL1="";
#-----
my $output1="SORTIE.xml";
if (!open (FILEOUT,">$output1")) { die "Pb a l'ouverture du fichier $output1";
#-----
&parcoursarborescencefichiers($rep); #recurse!
#-----
print FILEOUT "<?xml version=\"1.0\" encoding=\"iso-8859-1\" ?>\n";
print FILEOUT "<PARCOURS>\n";
print FILEOUT "<NOM>Votre nom</NOM>\n";
print FILEOUT "<FILTRAGE>".$DUMPFULL1."</FILTRAGE>\n";
print FILEOUT "</PARCOURS>\n";
close(FILEOUT);
exit;
#-----
sub parcoursarborescencefichiers {
my $path = shift(@_);
opendir(DIR, $path) or die "can't open $path: $!\n";
my @files = readdir(DIR);
closedir(DIR);
foreach my $file (@files) {
next if $file =~ /\.\.|\.?$/;
$file = $path."/". $file;
if (-d $file) {
&parcoursarborescencefichiers($file); #recurse!
}
if (-f $file) {
# TRAITEMENT à réaliser sur chaque fichier
# Insérer ici votre code (le filtreur)
print $i++,"\n";
}
}
}
#-----

```

Ce cinquième script représente le programme qu'on utilise pour parcourir l'arborescence des fils RSS (fichiers). Autrement dit, le répertoire qui contient les fils RSS au format XML. Avec les précédents scripts, il n'est pas possible de passer en argument un répertoire au script Perl. Pour parcourir l'arborescence d'un répertoire, on utilise le "Parcours arborescence" qui exécute plusieurs opérations.

Dans la première phase, on commence, d'abord, par affecter à la variable "\$rep" le répertoire passé en argument. On s'assure que le nom du répertoire ne se termine pas par un slash, en substituant le slash en fin de chaîne par rien. Ensuite, on initialise, à vide, une variable scalaire locale (my \$DUMPFULL1), qui va stocker l'ensemble des extractions.

Dans la seconde phase, on initialise une autre variable locale (\$output1). Cette variable correspond au fichier physique, qui va contenir les caractères imprimés, dans lequel on va mettre l'ensemble des parties textuelles concaténées. On remarque également que la sortie est au format XML.

Dans la troisième phase, un appel de sous programme est effectué grâce au symbole &. Nous avons mis ce sous programme en rouge.

Dans la phase qui suit, on prépare la sortie avec plusieurs balises. Parmi ces balises, on a la balise <FILTRAGE>, qui va contenir la variable \$DUMPFULL1.

Dans la dernière phase, on exécute le sous programme, lequel est défini par la commande sub. Il faut, par la suite, vérifier grâce à un next if si le nom du fichier contenu dans \$file contient "..". Si c'est le cas, on passe au suivant, sinon on réécrit le chemin absolu qui pointe vers l'élément voulu. Pour ce faire, on ajoute le nom correct au reste du chemin (\$path). On utilise, pour cette opération, le point, qui permet la concaténation de tous ces éléments. (\$file = \$path."/".\$file;).

Dans la dernière phase, on vérifie si l'élément qui est contenu dans \$file est un répertoire ou un fichier. Cette vérification se fait grâce à deux boucles if. La première utilise la commande -d, qui permet de s'assurer si l'élément est un répertoire. Si c'est le cas, alors on relance le parcours. La deuxième boucle if utilise la commande -f pour voir si l'élément est un fichier. Si la valeur est vraie, alors lance le traitement de filtrage et de nettoyage avec impression du résultat dans la variable \$DUMPFULL1.

6- Script final

```
#!/usr/bin/perl
#-----

my $rep="$ARGV[0]";
# on s'assure que le nom du répertoire ne se termine pas par un "/" si ça se termine par / on
remplace par rien
$rep=~ s/[V]$//;
# on initialise une variable contenant le flux de sortie
my $DUMPFULL1="";
my %tableaudestextes=();
#-----

my $output1="Sortie.xml";
if (!open (FILEOUT,">$output1")) { die "Pb a l'ouverture du fichier $output1";}
```

```

#-----
&parcoursarborescencéfichiers($rep); # on appelle le sous programme
#-----

sub parcoursarborescencéfichiers { # on entre dans le sous programme
my $path = shift(@_); # on met dans la variable $path le contenu de la variable mise en
argument
opendir(DIR, $path) or die "can't open $path: $!\n"; # opendir : ouvre un rep - si on arrive pas
: le pgm s'arrete et on envoie un message d'erreur
my @files = readdir(DIR); # on lit le contenu du répertoire et on renvoie une liste qu'on
stocke dans @files
closedir(DIR);
foreach my $file (@files) { # pour chaque ressource dans @files,on crée $files
next if $file =~ /\^\.\.?$/; # on passe au suivant si on trouve . ou .. car on risque une boucle
infinie
$file = $path."/".$file; # on réécrit tout le chemin du fichier - le point permet la concaténation
if (-d $file) { # on entre dans cette boucle si il y a dans $file un répertoire (-d vérifie que
l'argument est un répertoire)
&parcoursarborescencéfichiers($file); #recurse!
}
if (-f $file) { # on entre dans cette boucle si il y a dans $file un fichier (-f vérifie que
l'argument est un fichier)
if (($file=~0,2-651865,1-0,0.xml/) || ($file=~0,2-3236,1-0,0.xml/)){
open(FILEIN,$file); # on ouvre le fichier qu'on va lire
#$i=1; # compteur
printf "$file\n"; # on voit si ça travaille
while ($ligne = <FILEIN>){ # tant qu'il y a qqch à lire, on rentre dans la boucle
#chomp($ligne);
if ($ligne=~<description>([^\<]+)<\>/description>){
if (($ligne!~/Retrouvez *l.*ensemble *des/) && ($ligne!~/Lisez *l.*int.*gralit.* *de
*l.*article *pour *plus *d.*information./) && ($ligne!~/Toute *l.*actualit.* *au *moment
*de *la *connexion/)) {
my $propre=$1;# on prend le motif et on le met dans $propre
if ($propre=~/\^(.+)&lt;img width='1' height='1'.*$/ ) {$propre=$1};

```

```

$propre=~s/&#38;#39;/\'/g;
$propre=~s/&#38;#34;/\"/g;
$propre=~s/&#233;/é/g;
$propre=~s/&#234;/ê/g;
$propre=~s/&eacute;/é/g;
$propre=~s/&egrave;/è/g;
$propre=~s/&icirc;/î/g;
$propre=~s/&ocirc;/ô/g;
$propre=~s/&ccedil;/ç/g;
$propre=~s/&agrave;/à/g;
$propre=~s/&#38;/&/g;
$propre=~s/&lt;/</g;
$propre=~s/&gt;/>/g;

```

```

if (exists($tableaudestextes{$propre})) {

```

```

    $tableaudestextes{$propre}++;

```

```

}

```

```

else {

```

```

    $DUMPFULL1.="$propre\n";

```

```

    $tableaudestextes{$propre}++;

```

```

}

```

```

} # fin de if

```

```

} # fin de if

```

```

} # fin de while

```

```

close(FILEIN);

```

```

}

```

```

}

```

```

}

```

```

}

```

```

#-----

```

```

print FILEOUT "<?xml version=\"1.0\" encoding=\"iso-8859-1\" ?>\n"; # on dit que la sortie
doit être au format XML

```

```

print FILEOUT "<PARCOURS>\n";

```

```

print FILEOUT "<NOM>Marco et Nassim</NOM>\n";

```

```

print FILEOUT "<RUBRIQUE>0,2-651865,1-0,0.xml : technologies 1 an ; 0,2-3236,1-0,0.xml : médias 1 an </RUBRIQUE>\n";
print FILEOUT "<FILTRAGE>".$DUMPFULL1."</FILTRAGE>\n";
print FILEOUT "</PARCOURS>\n";
close(FILEOUT);
exit;

```

Ce script final intègre un parcours arborescence, un filtreur et un nettoyeur.

Tout d'abord, on remarque en bleu au début du script, l'initialisation à vide d'une nouvelle variable, qui est un tableau associatif précédé d'un %. Ce dernier permet d'éviter des lignes doublons. Rappelons qu'un "tableau associatif" est une structure qui permet de mettre en relation deux éléments de type scalaire. C'est un ensemble de couples de scalaires où l'un des éléments (la clé) référence l'autre (la valeur).

Ce tableau associatif va contenir les lignes qu'on imprime. Plus loin les boucles if et else en bleu permettent de faire un tri pour éviter les doublons. La boucle "if (exists(\$tableaudestextes{\$propre}))" empêche les lignes déjà écrites d'être à nouveau écrites. La boucle else, quant à elle, effectue l'opération inverse, en écrivant les lignes qui n'ont pas encore été écrites. Ces lignes sont stockées dans le tableau associatif pour éviter les éventuels doublons.

Un peu plus loin, on remarque également que la sortie est au format XML (Sortie.xml).

Dans la partie ayant trait au filtrage, une boucle if permet de préciser les rubriques sur lesquelles on travaille (Technologies = 0,2-651865,1-0,0 ; Médias = 0,2-3236,1-0,0.xml).

Un peu plus loin on utilise une autre boucle if (if ((\$ligne!~/Retrouvez *l.*ensemble *des/) && (\$ligne!~/Lisez *l.*int.*gralit.* *de *l.*article *pour *plus *d.*information./) && (\$ligne!~/Toute *l.*actualit.* *au *moment *de *la *connexion/)) {}). Celle-ci est un filtre qui permet de ne pas imprimer certaines lignes. On utilise des expressions régulières (. ; *) pour anticiper les éventuelles erreurs au niveau de certains caractères qui seraient mal encodés (e.g., é, è, ê, '). Juste après, on remarque une autre boucle if (if (\$propre=~/(.+)<img width='1' height='1'.*\$/) {\$propre=\$1});). Cette dernière sert à ne pas imprimer d'éventuels balises images (pavés publicitaires), en récupérant le motif qui est entre parenthèses dans \$1 puis dans \$propre.

Dans la partie ayant trait au nettoyage que nous avons mise en rouge, on remarque qu'il y a beaucoup plus de lignes de remplacement, afin de bien recoder toutes les entités html et décimales que nous avons rencontrés durant le traitement de nos fils RSS. Enfin, dans la dernière partie du script qui concerne la mise en forme de la sortie, on remarque l'ajout de la balise <rubrique>.

En sortie, on obtient un fichier au format XML, comme l'illustre la capture d'écran ci-dessous.

- <RUBRIQUE>

0,2-651865,1-0,0.xml : technologies 1 an ; 0,2-3236,1-0,0.xml : médias 1 an

<RUBRIQUE>

- <FILTRAGES>

L'objectif de la commission Copé est une suppression totale en 2011. Parmi les nouvelles tendances, le bio et l'art de vivre. Les deux consoles trident les jeux classés dans le top 100 des ventes. Cinq nouveaux titres sont en gestation. Le Britannique Tim Hetherington a remporté le grand prix du World Press de la photographie d'actualité. Sacré "Photo de l'année 2007", son cliché d'un soldat américain épuisé dans un bunker en Afghanistan a été publié dans le magazine américain "Vanity Fair". La direction de l'hebdomadaire souligne le caractère exceptionnel, voire inédit, de la procédure pénale pour "faux usage de faux et recel" engagée contre le site. Le résultat de l'appel d'offre de la Ligue de football professionnel (LFP) pour les droits de diffusion des matches de la Ligue 1 est clair : à partir de la saison 2008-2009, il faudra "payer pour voir" le sport le plus populaire. Actions en ligne et rassemblements sont prévus le 10 février. L'éditeur de logiciels devra penser autrement. Après l'exposition "L'esthétique domestique", le Musée des années 1930 de Boulogne-Billancourt (Hauts-de-Seine) propose "L'esthétique médiatique". Pendant la campagne des municipales, le journaliste tend son micro aux anonymes et relaie leurs revendications. Le radiozapping, chaque jour à midi, ce qu'il ne fallait pas rater des matinales radio. Canal + et Orange détenteurs des droits de retransmission. Mouvement dans la presse économique. Suppression de la publicité début 2009. TF1 va se pourvoir en cassation. Ce nouveau site propose des contenus à destination des enseignants et élèves, mais aussi des parents et éducateurs. Au musée des années 1930, à Boulogne-Billancourt, l'exposition "L'esthétique médiatique" met à l'honneur - à partir du 20 février - la collection de Jean-Bernard Hebey qui réunit depuis plus de trente ans des objets de notre quotidien au design étonnant. Un voyage dans l'univers de la communication - de la télévision au projecteur - à travers près de 350 objets venus du monde entier. Jérôme Bouvier ouvre un site participatif pour les municipales. Trois années de baisse en France, pays le plus touché en Europe. La campagne de l'hebdomadaire annulée dans le métro. Montée en puissance des chaînes thématiques et de la TNT. L'Elysée avait rajouté une phrase à la version initiale. Plainte du journaliste Charles Enderlin Une enquête de Médiamétrie sur les médias. Un vent de douceur soufflerait-il dans le milieu du disque ? Le Radiozapping, chaque jour à midi, tout ce qu'il ne fallait pas rater des matinales radio. La photo du président en visite en banlieue à Sartrouville (Yvelines), parue dans plusieurs grands quotidiens, n'a pas été signée d'un journaliste mais d'un gendarme. Le ministère de la culture a ouvert un forum où les internautes s'expriment sur la réforme de France Télévisions. Les ventes de ces CD, DVD et autres livres qui accompagnent les quotidiens et magazines représentent aujourd'hui 100 millions d'euros. Et devraient encore augmenter. Le quotidien a perdu 10,5 millions d'euros en 2007. La direction souhaite économiser 12 millions d'euros, dont 7 millions par un plan de départs volontaires. Le documentaire de Thierry de Lestrade et Sylvie Gilman dénonce la faiblesse des avancées dans le combat contre la maladie. La société des rédacteurs dénonce une "dérive partisane". Elle s'étonne aussi "qu'un plan de réduction d'effectifs soit annoncé, alors que le résultat du groupe est positif pour 2007". La chaîne privée souffre de la montée

Étiquetage morpho-syntaxique

Aperçu général

Dans les années 80, le modèle probabiliste de Markov⁴⁸ est apparu, entre autres, pour l'étiquetage morphosyntaxique puis il a été employé pour l'annotation d'entités nommées (cf. à ce propos, Ezzat, 2008). Ce modèle probabiliste a fait ses preuves dans l'opération d'étiquetage morphosyntaxique, qui consiste à assigner à chaque mot d'une phrase d'un texte, des informations morphosyntaxiques, comme la catégorie grammaticale ou les traits morphologiques. Cette opération commence par une segmentation de la phrase en token (tokenisation). Le texte est découpé en unités lexicales simples. Ensuite, un lexique est utilisé pour effectuer une analyse lexicale ambiguë qui consiste à assigner à chaque token l'ensemble des étiquettes possibles. Les mots inconnus subissent un traitement spécial car on ne dispose d'aucune information lexicale sur eux. Toutes les ambiguïtés des mots d'une phrase seront représentées par un automate acyclique. Enfin, la dernière phase est la phase de levé d'ambiguïté qui recherche le meilleur chemin dans l'automate de la phrase. Cette étape sera résolue grâce à un calcul de probabilités. Plus une séquence d'étiquettes est probable, plus elle a de chance d'être choisie pour la suite de tokens d'une phrase à étiqueter. Autrement dit, pour étiqueter un token, nous allons nous servir des probabilités dites d'« émission » qui déterminent la probabilité d'avoir un token étant donné une étiquette ainsi que des probabilités dites de « transitions » qui considèrent les étiquettes des contextes gauches du token à étiqueter. Le texte à étiqueter est vu comme une séquence de mots qui contient des informations utiles et aussi des informations inutiles. Cette séquence est appelée « séquence

⁴⁸ HMM (Hidden Markov Model).

d'observation ». L'objectif est donc de segmenter cette séquence d'observation en une séquence d'états (étiquettes). Le modèle de Markov caché détermine une probabilité jointe écrite $P(x,y)$. L'ensemble « x » représentera la séquence d'observations et « y » la séquence d'étiquettes. Les états constituent la couche *cachée* de ce modèle qui produit une séquence d'observations. Le modèle cherche alors à trouver la séquence d'états cachés la plus probable « y » en fonction de la séquence d'observation « x », le tout, en se basant sur des exemples étiquetés.

Le programme Treetagger (Schmid⁴⁹, 1994) en est un bon exemple d'illustration. Il évalue la probabilité d'une étiquette en fonction des étiquettes précédente. Il s'agit donc de probabilités conditionnelles. Son architecture nécessite, pour chaque langue traitée, un lexique de formes fléchies, un jeu d'étiquettes et un corpus d'entraînement préannoté. L'apprentissage effectué par TreeTagger consiste ensuite à évaluer la probabilité d'une transition entre un token et un autre token, puis à générer un arbre de décision binaire à partir des probabilités calculées. Le modèle de Markov se base également sur le principe dit « trigramme », c'est-à-dire que les probabilités contextuelles sont calculées à partir d'un contexte gauche de deux étiquettes morphosyntaxiques. En effet, selon Vellis (1999, p. 3-4) ce type de modèle statistique exploite « des contextes très limités, généralement des bigrammes (deux mots consécutifs) ou des trigrammes (trois mots consécutifs), dont les fréquences permettent, de proche en proche, de prédire la séquence d'étiquettes la plus probable ». Cette limitation est très contraignante, lorsqu'il s'agit d'extraire une relation entre entités nommées comme l'a souligné Ezzat : « Jusqu'au début des années 2000, les modèles stochastiques tels que les chaînes de Markov cachées (Hidden Markov Model, HMM), étaient omniprésents et offraient de bons résultats pour l'annotation d'entités nommées et morpho-syntaxique.

L'étiquetage des données textuelles

Dans cette deuxième étape, nous allons présenter les deux outils qui nous ont permis d'avoir des sorties étiquetées. Ces sorties étiquetées vont représenter des données en entrée pour la prochaine étape. Mais pour le moment, nous allons nous focaliser sur les outils d'étiquetage morphosyntaxique.

Deux méthodes différentes ont été utilisées. La première permet de faire un étiquetage au moment où on extrait le texte, grâce au Treetagger qu'on intègre directement à notre script final. L'étiquetage se fait, au fur et à mesure de l'extraction des parties textuelles, flux après flux. Dans la deuxième méthode, il s'agit d'utiliser une interface graphique (Cordial), qui prend en entrée un fichier au format texte brut.

1- Étiquetage via Treetagger (au fil de l'eau)

Comme nous l'avons déjà évoqué plus haut, Treetagger est un outil, qui permet d'étiqueter les contenus textuels des fils RSS juste après leur extraction. On obtient, en sortie, un fichier XML. Treetagger s'utilise en ligne de commande et fournit trois types d'informations : la partie du discours, le lemme et la forme graphique.

Avant d'intégrer cet outil à notre script final, nous avons d'abord commencé à nous familiariser avec ses différents modules. Pour ce faire, nous l'avons utilisé en ligne de commande sur un échantillon de fils RSS. Nous présentons, ci-dessous la ligne de commande utilisée et le résultat de l'étiquetage.

⁴⁹ Université de Stuttgart.

```
$perl ./cmd/tokenise-fr.pl fichierentrée.txt | ./bin/tree-tagger.exe ./lib/french.par -lemma -token -sgml > fichiersortie.txt
```

Parmi PRP parmi

les DET:ART le

nouvelles ADJ nouveau

tendances NOM tendance

, PUN ,

le DET:ART le

bio ADJ biologique

et KON et

l' DET:ART le

art NOM art

de PRP de

vivre VER:infi vivre

haut de page

On remarque surtout, que le chemin employé dans la ligne de commande est de type UNIX. En effet, il est important de signaler qu'il est nécessaire de spécifier le type de chemin, selon qu'on travaille sous Cygwin (émulation de UNIX sous Windows) ou sous l'invite de commande de Windows. Ainsi, sous Cygwin on utilise ce type de chemin : `./treetagger-win32/cmd/treetagger2xml.pl`, alors que sous Windows, c'est ce type de chemin qu'on va utiliser : `\"./treetagger-win32/cmd/tokenise-fr.pl`. Par ailleurs, les fichiers `tree-tagger.exe` et `french.par` doivent être placés au même niveau que le fichier qui doit être étiqueté.

Voyons, à présent, de plus près les différents composants de Treetagger lorsque nous l'avons intégré à notre script. Précisons, avant toute chose, qu'il s'agit seulement d'un extrait du script global, car nous allons décrire la partie du code ayant trait au processus d'étiquetage.

```
if (exists($tableaudestextes{$propre})) {  
  
    $tableaudestextes{$propre}++;  
  
}  
  
else {  
  
    open(file, ">temp.txt");
```

```

print file $propre;

print file;

close(file);

print "début de l'étiquetage\n";

#system("perl ./treetagger-win32/cmd/tokenise-fr.pl temp.txt | ./treetagger-win32/bin/tree-
tagger.exe ./treetagger-win32/lib/french.par -lemma -token -no-unknown > sorttik.txt");

#system("perl ./treetagger-win32/cmd/treetagger2xml.pl sorttik.txt");

system ("perl \"./treetagger-win32/cmd/tokenise-fr.pl\" temp.txt > token.txt");#on lance
tokenisation

system ("tree-tagger.exe french.par -lemma -token -no-unknown token.txt > sorttik.txt");#on
lance l'étiquetage

system ("perl \"./treetagger-win32/cmd/treetagger2xml.pl\" sorttik.txt");# on convertit au
format xml ce qui était au format txt

print "fin de l'étiquetage\n";

open(file, "sorttik.txt.xml");

my $temp=<file>; #permet de lire juste la première ligne, afin d'éviter d'affichier l'entête à
chaque fois

my $result="";

while (my $ligne1=<file>) {

$result.=$ligne1;

}

close(file);

$DUMPFULL2.=$result;

$tableaudestextes{$propre}++;

}

```

Dans ce script, on récupère, tout d'abord, le flux textuel qui est contenu dans la variable \$propre, dans un fichier qu'on ouvre en écriture (temp.txt). Ensuite, on arrive aux lignes qui représentent le coeur de l'étiquetage. En rouge, les lignes de code qui conviennent à Cygwin

et en bleu celles qui sont adaptées à l'invite de commande de Windows. On désactive les lignes en rouge pour se focaliser, dans notre description, sur le mode Windows.

Perl ne sait pas tout faire. Il offre une fonction bien pratique pour faire appel à un programme extérieur : la commande `system()`.

On utilise, à trois reprises, cette commande, afin d'appeler les modules de Treetagger et de mener à bien le processus d'étiquetage. Dans la première ligne `system()`, on utilise le script Perl "tokenise-fr.pl", qui prend, comme paramètre, le fichier `temp.txt`. Ce dernier contient le flux textuel. On obtient alors une liste de mots séparés par un retour à la ligne. Ensuite, la deuxième ligne `system()` utilise le programme d'étiquetage "tree-tagger.exe", qui prend, comme paramètre le résultat de la tokenisation "token.txt" et génère un fichier étiqueté en sortie "sortik.txt". Une fois le fichier tokenisé et étiqueté, nous utilisons le script Perl "treetagger2xml.pl", qui permet de transformer la sortie au format texte brut en une sortie au format XML.

Nous vous proposons, à travers le lien ci-dessous, un extrait de la sortie générée par l'outil Treetagger :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<PARCOURS>
<ETIQUETAGE>
<element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">L'</data></element>
<element><data type="type">NOM</data><data type="lemma">objectif</data><data type="string">objectif</data></element>
<element><data type="type">PRP</data><data type="lemma">de</data><data type="string">de</data></element>
<element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">la</data></element>
<element><data type="type">NOM</data><data type="lemma">commission</data><data type="string">commission</data></element>
<element><data type="type">ADJ</data><data type="lemma">Copé</data><data type="string">Copé</data></element>
<element><data type="type">VER:pres</data><data type="lemma">être</data><data type="string">est</data></element>
<element><data type="type">DET:ART</data><data type="lemma">un</data><data type="string">une</data></element>
<element><data type="type">NOM</data><data type="lemma">suppression</data><data type="string">suppression</data></element>
<element><data type="type">ADJ</data><data type="lemma">total</data><data type="string">totale</data></element>
<element><data type="type">PRP</data><data type="lemma">en</data><data type="string">en</data></element>
<element><data type="type">DET:POS</data><data type="lemma">@ord@</data><data type="string">2011.</data></element>
<element><data type="type">PRP</data><data type="lemma">parmi</data><data type="string">Parmi</data></element>
<element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">les</data></element>
<element><data type="type">ADJ</data><data type="lemma">nouveau</data><data type="string">nouvelles</data></element>
<element><data type="type">NOM</data><data type="lemma">tendance</data><data type="string">tendances</data></element>
<element><data type="type">PUN</data><data type="lemma">,</data><data type="string">,</data></element>
<element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">le</data></element>
<element><data type="type">ADJ</data><data type="lemma">biologique</data><data type="string">bio</data></element>
<element><data type="type">KON</data><data type="lemma">et</data><data type="string">et</data></element>
<element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">l'</data></element>
<element><data type="type">NOM</data><data type="lemma">art</data><data type="string">art</data></element>
<element><data type="type">PRP</data><data type="lemma">de</data><data type="string">de</data></element>
<element><data type="type">VER:infi</data><data type="lemma">vivre</data><data type="string">vivre</data></element>
<element><data type="type">SENT</data><data type="lemma">.</data><data type="string">.</data></element>
```

2- Étiquetage via Cordial

Cordial est un programme développé par la société Synapse Développement. Ce programme permet d'effectuer différentes tâches :

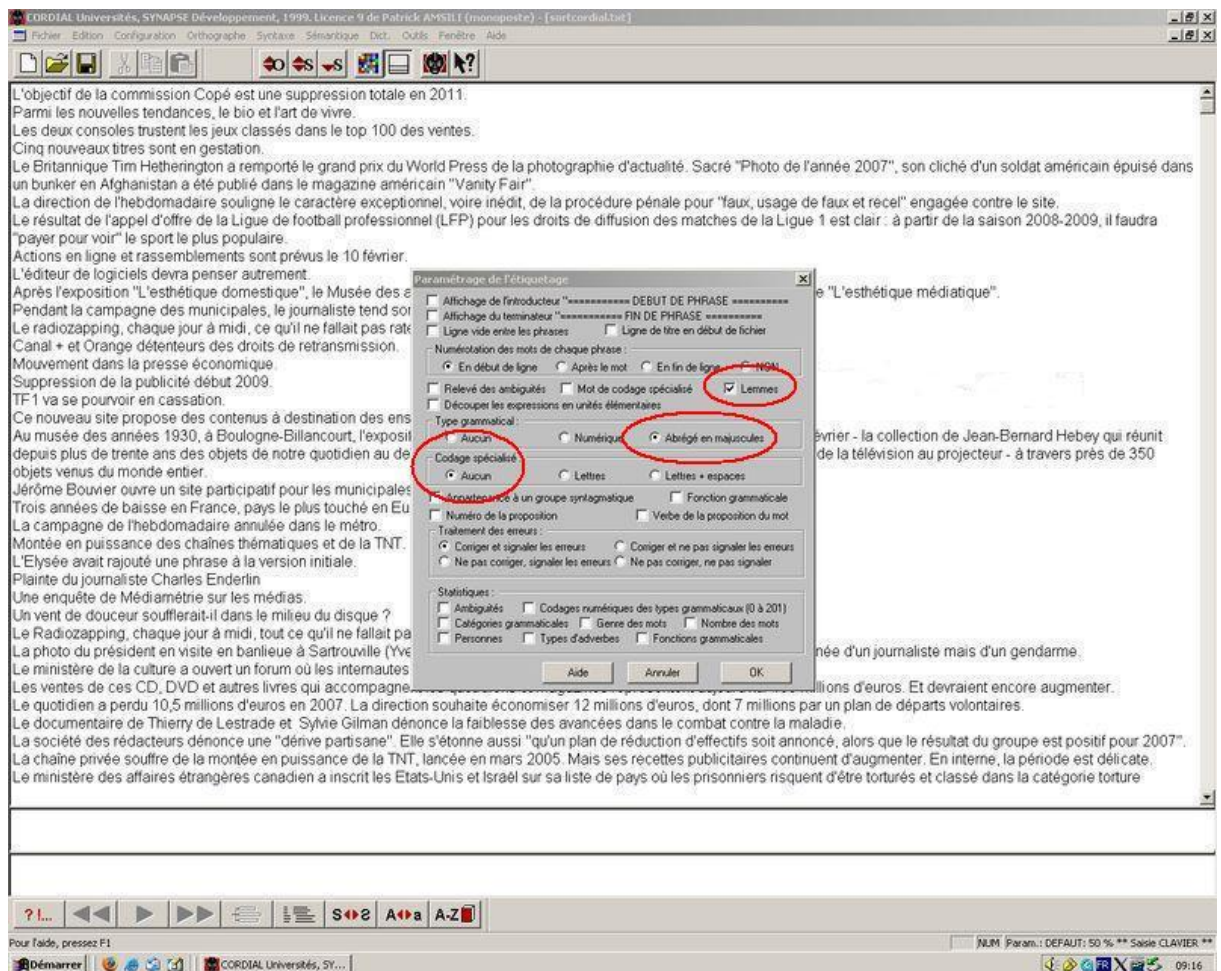
- Étiquetage morphosyntaxique des textes en français ;
- Analyse statistique des caractéristiques stylistiques de textes ;
- Aide à l'analyse terminologique et sémantique de corpus.

Par ailleurs, Cordial prend, entrée, un fichier txt et génère un fichier étiqueté au format cnr. Dans ce fichier, on trouve plusieurs informations : la forme graphique, le lemme et la partie du discours. Cordial indique également le genre et le nombre des parties du discours et les adjectifs numéraux.

L'	le	DETDMS	
objectif		objectif	NCMS
de	de	PREP	
la	le	DETDMS	
commission		commission	NCFS
Copé	Copé	NPI	
est	être	VINDP3S	
une	un	DETIFS	
suppression		suppression	NCFS
totale	total	ADJFS	
en	en	PREP	
2011	2011	NCMIN	
.	.	PCTFORTE	
\r			
Parmi	parmi	PREP	
les	le	DETDPIG	
nouvelles		nouveau	ADJFP
tendances		tendance	NCFP
,	,	PCTFAIB	
le	le	DETDMS	
bio	bio	ADJINV	
et	et	COO	
l'	le	DETDMS	
art de vivre		art de vivre	NCMS
.	.	PCTFORTE	
\r			
Les	le	DETDPIG	
deux	deux	ADJNUM	
consoles		consol	NCFP

Nous vous proposons deux captures d'écran de l'interface graphique de Cordial. Dans la première capture d'écran, le texte à étiqueter est ouvert. Dans la seconde capture d'écran, on procède au paramétrage de l'étiquetage.

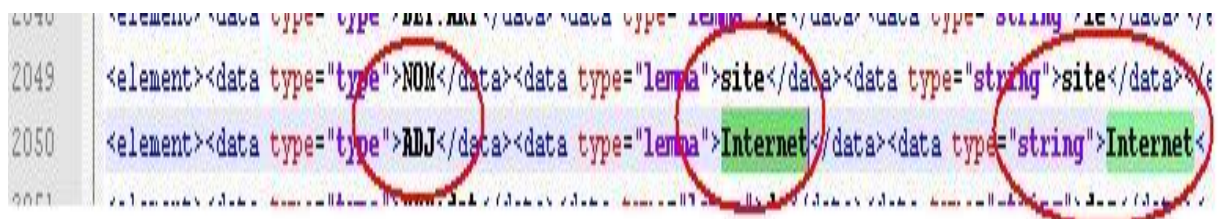




3- Comparaison des résultat d'étiquetage : Treetagger VS Cordial

Dans cette dernière partie de la deuxième étape, nous allons comparer les étiquetages générés par Treetagger et Cordial et nous verrons que la différence d'étiquetage aura des répercussions directes sur l'extraction des patrons "NOM ADJ" et "NOM PREP NOM" (BAO3).

Les deux fichiers étiquetés ont quasiment le même nombre de lignes étiquetées (3628 pour Treetagger et 3640 pour Cordial). En outre, à plusieurs reprises, Treetagger interprète une suite de deux noms comme un nom suivi d'un adjectif. Prenons l'exemple de "site Internet" (lignes 2049 et 2050) que Treetagger étiquette comme un NOM ADJ. L'image, ci-dessous illustre nos propos.



L'image ci-dessous montre que Cordial étiquette "site Internet" (lignes 2050 et 2051) comme un nom suivi d'un autre nom.

2049	le	le	DETDMS
2050	site	site	NCMS
2051	Internet	Internet	NPSIG
2052	des	un	DETDPIG

D'autres exemples de ce type peuvent être mentionnés. Ainsi, Cordial étiquette, "commission Copé" (lignes 5 et 6) comme une suite de deux noms, alors que Treetagger étiquette cette même suite comme un nom suivi d'un adjectif. Tout comme "box ADSL" que Cordial reconnaît comme une suite de deux noms communs (lignes 2241 et 2242), alors que Treetagger étiquette cette suite comme un nom suivi d'un adjectif (ligne 2241 et 2242).

Lorsque Treetagger croit reconnaître une succession de deux noms, celle-ci s'avère être fautive. Dans la phrase : "[...] il aurait proposé à son ex-épouse Cécilia de reprendre leur vie commune, une semaine avant son remariage", commune ne signifie pas : une division territoriale administrée par un maire, mais un adjectif. D'où l'absence de ce patron NOM ADJ dans la liste générée à partir du fichier étiqueté avec Treetagger dans la troisième étape.

Un autre phénomène récurrent peut être constaté concernant la suite prénom/nom que Treetagger étiquette comme un NOM ADJ, alors que Cordial la considère comme une suite de deux noms propres :

*Cordial = Tim Hetherington (lignes 52 et 53) NPI NPI ; Treetagger = NOM ADJ (Lignes 54 et 55).

*Cordial = Laurence Parisot (lignes 1295 et 1296) NPFS NPI ; Treetagger = NOM ADJ (lignes 1289 et 1290).

*Cordial = Jason Bonham (lignes 1063 et 1064) NPMS NPI ; Treetagger = NOM ADJ (lignes 1052 et 1053).

En outre, dans certaines suites prénom/nom, Treetagger étiquette le nom soit comme une abréviation (lignes 1758 et 1759), soit comme un verbe (lignes 1814 et 1815). Ci-dessous, deux captures d'écran qui illustrent nos propos.

1754	type="type">VER:infi</data><data type="lemma">retenir</data><data type="string">retenir
1755	type="type">DET:ART</data><data type="lemma">le</data><data type="string">la</data></el
1756	type="type">NOM</data><data type="lemma">candidature</data><data type="string">candidat
1757	type="type">PRP</data><data type="lemma">de</data><data type="string">d'</data></elemen
1758	type="type">NAM</data><data type="lemma">Eric</data><data type="string">Eric</data></el
1759	type="type">ABR</data><data type="lemma">Fottorino</data><data type="string">Fottorino<
1760	type="type">DML</data><data type="lemma"></data><data type="string"></data></elemen<

```

1812 <element><data type="type">NOM</data><data type="lemma">mort</data><data type="string">mort</data></el
1813 <element><data type="type">SENT</data><data type="lemma">.</data><data type="string">.</data></element>
1814 <element><data type="type">NAM</data><data type="lemma">Cécile</data><data type="string">Cécile</data>
1815 <element><data type="type">VER:simp</data><data type="lemma">Allegra</data><data type="string">Allegra
1816 <element><data type="type">PUN</data><data type="lemma">,</data><data type="string">,</data></element>

```

Cela ne veut pas dire que Cordial ne présente pas certaines anomalies au sujet de l'étiquetage de la suite prénom-nom. En effet, nous pouvons relever deux cas où Cordial étiquette le nom comme un nom commun. Ci-dessous, une capture d'écran pour illustrer nos propos.

```

1042 . . PCTFORTE
1043 Sur scène sur scène ADV
1044 Robert Robert NPMS
1045 Plant plant NCMS
1046 , , PCTFAIB
1047 chant chant NCMS
1048 , , PCTFAIB
1049 Jimmy Jimmy NPMS
1050 Page page NCFS
1051 , , PCTFAIB
1052 guitare guitare NCFS
1053 , , PCTFAIB

```

On constate un autre phénomène concernant les unités polylexicales qui doivent normalement recevoir une seule étiquette grammaticale. On en dénombre quelques cas dans la sortie étiquetée de Cordial (e.g., moteur de recherche (ligne 1224), juge d'instruction (ligne 2158), juge des référés (ligne 2257). Treetagger décompose ces trois noms communs en trois tokens (NOM PREP NOM).

```

1210 <element><data type="type">DET:ART</data><data type="lemma">le</data><data type="string">le</data></el
1211 <element><data type="type">NOM</data><data type="lemma">moteur</data><data type="string">moteur</data>
1212 <element><data type="type">PRP</data><data type="lemma">de</data><data type="string">de</data></element>
1213 <element><data type="type">NOM</data><data type="lemma">recherche</data><data type="string">recherche</data>
1214 <element><data type="type">NOM</data><data type="lemma">moteur</data><data type="string">moteur</data>

```

```

2156 <element><data type="type">DET:ART</data><data type="lemma">un</data><data type="string">Un</data></element>
2157 <element><data type="type">NOM</data><data type="lemma">juge</data><data type="string">juge</data></element>
2158 <element><data type="type">PRP</data><data type="lemma">de</data><data type="string">d'</data></element>
2159 <element><data type="type">NOM</data><data type="lemma">instruction</data><data type="string">instruction</data>
2160 <element><data type="type">PRP:det</data><data type="lemma">du</data><data type="string">du</data></element>

```

Par ailleurs, dans la suite "chaque jour à midi", Cordial étiquette "chaque jour" comme un adverbe (ligne 480) et "à midi" comme un autre adverbe (ligne 481). Pour Cordial, cette suite

est donc composée de deux tokens. Treetagger décompose cette suite en quatre tokens (lignes 249, 250, 251 et 252).

Un dernier point de divergence au niveau des étiquetages concerne la reconnaissance du token "du". Ainsi, dans "holding du groupe" (lignes 2782, 2783 et 2784), "transformation du château" (lignes 2378, 2379 et 2380) et "propriétaire du journal" (lignes 2450, 2451 et 2452), Cordial étiquette du comme un déterminant masculin singulier, alors que Treetagger le reconnaît comme une préposition et déterminant (noté : PRP:det). Ceci aura pour effet de générer plus de patrons NOM PREP NOM à partir de la sortie étiquetée avec Treetagger.

Ainsi, nous verrons que la décomposition en plusieurs tokens de certaines suites ainsi que l'étiquetage du token « du » aura des répercussions directes dans l'extraction du patron NOM PREP NOM.

Extraction de patrons

Dans cette étape, nous avons extrait les patrons NOM ADJ et NOM PREP NOM à partir des sorties étiquetées via Treetagger et Cordial. Nous allons présenter, dans un premier temps, l'extraction des patrons NOM ADJ et NOM PREP NOM, à partir de la sortie générée par Treetagger. Pour ce faire, nous avons écrit des script en langage PERL.

1- Extraction avec un script PERL à partir de la sortie étiquetée avec Treetagger Le script Perl qui sert à extraire le patron NOM ADJ

```
#!/usr/bin/perl

open(FILE,"$ARGV[0]");

open (FILEOUT, ">sortie_SF_NOMADJ_tree.txt");

my @lignes=<FILE>;

close(FILE);

while (@lignes) {

my $ligne=shift(@lignes);

chomp $ligne;

my $sequence="";

my $longueur=0;

if ( $ligne =~ /<element><data type=\"type\">NOM<\data><data type=\"lemma\">[^\<]+<\data><data type=\"string\">([^\<]+)<\data><\element>/) {

my $forme=$1;

$sequence.=$forme;
```

```

$longueur=1;

my $nextligne=$lignes[0];

if ( $nextligne =~ /<element><data type=\"type\">ADJ<\data><data
type=\"lemma\">[^\<]+<\data><data type=\"string\"><([^\<]+)<\data><\element>/) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=2;

}

}

if ($longueur == 2) {

print FILEOUT "$sequence\n";

}

}

close(FILE);

close(FILEOUT);

```

Dans ce script on met, dans un premier temps, toutes les lignes du fichier XML (Treetagger), dans un tableau (@lignes). Ensuite, on prend le premier indice du tableau grâce à la fonction shift et on l'affecte à la variable my \$ligne. On enlève les sauts de ligne et on initialise à vide la variable \$sequence, qui va contenir le patron NOM ADJ.

On utilise, par la suite, deux boucles if imbriquées, qui vont filtrer le patron que l'on recherche. Grâce à une expression régulière en bleu (tout sauf chevron), on récupère, dans chaque boucle if, le motif (forme graphique = string) dans la variable prédéfinie \$1. Si la variable \$longueur équivaut à la valeur == 2, alors on imprime le contenu de la variable \$sequence.

```

suppression totale
Tim Hetherington
World Press
soldat américain
magazine américain
caractère exceptionnel
procédure pénale
football professionnel
esthétique domestique
esthétique médiatique
presse économique
esthétique médiatique
design étonnant
monde entier
site participatif
chaînes thématiques
version initiale
départs volontaires
dérive partisane
recettes publicitaires
affaires étrangères
sites Internet
groupe imaginaire
chaîne cryptée
site nouvelobs
journalistes français
histoires croisées
difficulté croissante
mixage digital

```

Le script Perl qui sert à extraire le patron NOM PREP NOM

```

#!/usr/bin/perl
open(FILE,"$ARGV[0]");

open (FILEOUT, ">sortie_SF_NOMPRPNOM_tree.txt");

my @lignes=<FILE>;

close(FILE);

while (@lignes) {

my $ligne=shift(@lignes);

chomp $ligne;

my $sequence="";

my $longueur=0;

if ( $ligne =~ /<element><data type=\"type\">NOM<\data><data
type=\"lemma\">[^\<]+<\data><data type=\"string\">([^\<]+)<\data><\element>/) {

my $forme=$1;

$sequence.=$forme;

```

```

$longueur=1;

my $nextligne=$lignes[0];

if (    $nextligne    =~    /<element><data    type=\"type\">PRP.*<\data><data
type=\"lemma\">[^<]+<\data><data type=\"string\">([^\<]+)<\data><\element>/) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=2;

my $nextligne=$lignes[1];

if (    $nextligne    =~    /<element><data    type=\"type\">NOM<\data><data
type=\"lemma\">[^<]+<\data><data type=\"string\">([^\<]+)<\data><\element>/) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=3;

}

}

}

if ($longueur == 3) {

print FILEOUT "$sequence\n";

}

}

close(FILE);

close(FILEOUT);

```

photographie d' actualité
 bunker en Afghanistan
 appel d' offre
 droits de diffusion
 diffusion des matches
 Actions en ligne
 éditeur de logiciels
 Musée des années
 micro aux anonymes
 jour à midi
 détenteurs des droits
 droits de retransmission
 pouvoir en cassation
 contenus à destination
 destination des enseignants
 musée des années
 ans des objets
 quotidien au design
 télévision au projecteur
 années de baisse
 puissance des chaînes
 Plainte du journaliste
 enquête de Médiamétrie
 vent de douceur
 milieu du disque
 jour à midi
 photo du président
 président en visite
 visite en banlieue

Ce script a le même fonctionnement que le premier, qui extrait le patron NOM ADJ. Cependant, nous l'avons modifié dans la deuxième boucle if (en rouge), en utilisant une expression régulière (.) pour pouvoir prendre en compte certains patrons NOM PREP NOM, dont le second élément peut être une "PRP:det", comme on peut le voir dans l'image ci-dessous. Grâce à cette modification, on récupère 186 patrons NOM PREP NOM, au lieu de 107 patrons sans cette modification.

```

\data type= type /ver.pics\data\data type= lemma /attaquer\data\data type= string /
<data type="type">PRP:det</data><data type="lemma">au</data><data type="string">au</dat
<data type="type">NOM</data><data type="lemma">marché</data><data type="string">marché<
<data type="type">PRP:det</data><data type="lemma">du</data><data type="string">du</dat
<data type="type">NOM</data><data type="lemma">vin</data><data type="string">vin</data>
  
```

À la suite de l'extraction du patron NOM ADJ et NOM PREP NOM avec un script Perl, nous avons constaté que le nombre de patrons NOM ADJ (127 patrons) et NOM PREP NOM (186 patrons).

Certaines anomalies sont observées avec cette méthode d'extraction. On trouve, par exemple, les mauvaises extractions "site Internet" et "box ADSL" qui ne correspondent pas au patron NOM ADJ. Cependant, il est possible de considérer que dans certains cas, un nom peut se trouver en situation adjectivale et servir à qualifier. On peut dès lors le considérer comme épithète. Ceci expliquerait pourquoi les tokens « Internet » et « ADSL » ont reçu l'étiquette ADJ.

Dans l'extraction « Laurence Parisot » (Parisot n'est pas un adjectif) et l'absence du patron "vie commune" pour les raisons que nous avons déjà évoquées dans la deuxième étape. On trouve aussi le patron NOM PREP NOM "début 2009 C". Cette extraction est le résultat d'une erreur étiquetage.

Concernant le patron NOM PREP NOM, notre méthode d'extraction génère, à partir de la sortie de Treetagger, une sortie assez volumineuse comparée à celle que on a obtenue avec l'étiquetage de Cordial. Ceci s'explique, entre autres, par le fait que Treetagger étiquette le token du dans des patrons comme "propriétaire du journal" ou "holding du groupe", comme une préposition:déterminant. On verra que dans la sortie de Cordial du patron NOM PREP NOM, ce type de patrons est absent, car du est reconnu comme un déterminant.

Extraction de patrons à partir de la sortie étiquetée avec Cordial

1- Extraction du patron NOM ADJ

* Extraction du patron NOM ADJ avec le premier script

Le script, ci-dessous, prend en entrée deux paramètres. Le premier paramètre est le fichier étiqueté avec Cordial au format cnr. Le second paramètre est un fichier au format texte, qui contient la liste des patrons que l'on recherche.

```
#!/usr/bin/perl

# use locale;

open(FICCORDIAL, "$ARGV[0]");

open (OUT, ">sortiecordNOMADJ_JMD.txt");

while ($ligne= <FICCORDIAL>) {

next if ($ligne!~/\t/);

chomp $ligne;

if ($ligne !~/PCT/){ #on s'arrete à la ponctuation.

@decoup=split(/\t/, $ligne);

push (@token, $decoup[0]);

push (@lemme, $decoup[1]);

push (@partof, $decoup[2]);

$i++;#comptage des parties du discours.

}

else {

open(FICPATRONS, "$ARGV[1]");

while ($patrons=<FICPATRONS>) {

chomp $patrons;

my $compare="";

foreach my $element(@partof) {
```

```

#$compare=$compare."\t".$selement;

#}

$compare=join("\t", @partof);

my $j=0;

my $k=0;

while ($compare=~/$patrons/g) {

my $avant = `;

my $apres = `;

while ($avant=~/\t/g) {$j++};

while ($apres=~/\t/g) {$k++};

print OUT "@token[$j..$i-$k-1]\n";

$j=0;

$k=0;

}

}

close (FICPATRONS);

$i=0;

@token=();

@lemme=();

@partof=();

}

}

```

Dans ce script, on commence d'abord par lire chaque ligne du fichier étiqueté avec une boucle while. Avec un next if, on saute les lignes, qui ne contiennent pas une tabulation. On supprime, avec la fonction chomp les éventuels retour à la ligne. Si la ligne ne contient pas une ponctuation (PCT), on fait le traitement en rouge. Dans ce traitement, il s'agit de segmenter, grâce à la commande split, chaque ligne du fichier en entrée en trois colonnes, avec comme séparateur une tabulation (\t).

On obtient trois colonnes d'éléments. Chaque colonne du fichier est placée dans un tableau (@token, @lemme, @partof), grâce à la commande puch. Si la ligne contient une ponctuation, on passe au traitement de la boucle else. Dans cette dernière, on lit le deuxième paramètre et on enlève les éventuels retours à la ligne. On initialise à vide la variable \$compare, qui va contenir les éléments du tableau @partof séparés par une tabulation. On utilise la commande join pour transférer le contenu du tableau @partof dans la variable scalaire \$compare. Pour effectuer cette opération, on peut aussi utiliser une boucle foreach (en bleu), qui concatène les parties du discours avec une tabulation (\t) avant de trouver un PCT (ponctuation). Rappelons qu'avec cette méthode, on doit spécifier dans cette ligne : print OUT "@token[\$j-1..\$i-\$k-1]\n";.

L'étape qui suit est la création de deux variables (\$j et \$k), auxquelles on affecte la valeur "0". Une boucle while permet, ensuite, de comparer la liste des patrons que l'on recherche stockée dans la variable \$patrons avec les parties du discours, qui se trouvent dans la variable \$compare. S'il y a correspondance entre ces deux variables, alors on identifie et on imprime les tokens (mots) qui correspondent aux parties du discours du patron recherché. Pour ce faire, on utilise un paramétrage des indices du tableau @token (@token[\$j..\$i-\$k-1]) pour faire une bijection (cf. la ligne en vert) entre les parties du discours de \$compare et les éléments de @token.

Pour lancer ce premier script, nous avons utilisé une liste de 33 patrons NOM ADJ. Celle-ci contient toutes les combinaisons possibles, sauf celles qui contiennent des noms propres. Autrement dit, on extrait tous les noms communs quelque soit le genre et le nombre du nom et de l'adjectif. On prend également en compte les adjectifs numéraux, comme on peut le voir ci-dessous.

NCFS ADJFS

NCMS ADJMS

NCMS ADJINV

NCMS ADJFS

NCFIN ADJFS

NCI ADJHFS

NCMS ADJIND

NCFS ADJSIG

NCFS ADJNUM

NCFP ADJPIG

NCFP ADJIND

NCMP ADJPIG

NCMP ADJIND

NCMP ADJMS
 NCFS ADJIND
 NCFP ADJFP
 NCMS ADJSIG
 NCPIG ADJMIN
 NCPIG ADJIND
 NCPIG ADJMP
 NCMS ADJMIN
 NCMIN ADJMP
 NCMIN ADJINV
 NCMIN ADJIND
 NCMS ADJMP
 NCSIG ADJSIG
 NCFS ADJMP
 NCSIG ADJFS
 NCMIN ADJMS
 NCMP ADJMIN
 NCMP ADJMP
 NCMP ADJSIG
 NCFS ADJMS

On obtient alors 132 patrons NOM ADJ. Parmi ces patrons, on trouve trois patrons ayant comme deuxième éléments un adjectif numéral, selon l'étiquetage de Cordial :

- Exposition L' ;
- Exposition L' ;
- tranche 13-24.

On remarque que les deux premiers patrons sont extraits suite à étiquetage erroné. En outre, on relève d'autres patrons, qui sont issus d'un mauvais étiquetage :

- DVD haute (haute ne modifie pas DVD, car il réfère au mot "définition" qui suit) ;
- pays le plus (le plus n'est pas un adjectif) ;
- écrans plus de (plus de n'est pas un adjectif) ;
- départ d'ici (d'ici n'est pas un adjectif).

```

suppression totale
soldat américain
magazine américain
caractère exceptionnel
procédure pénale
football professionnel
sport le plus
esthétique domestique
exposition L'
esthétique médiatique
presse économique
esthétique médiatique
exposition L'
design étonnant
monde entier
site participatif
pays le plus
hebdomadaire annulée
chaînes thématiques
version initiale
départs volontaires
dérive partisane
chaîne privée
recettes publicitaires
ministère des affaires étrangères canadien
torture certaines
techniques américaines
militants antisectes
groupe imaginaire
vie commune
journalistes français

```

*** Extraction du patron NOM ADJ avec le deuxième script Perl**

Nous présentons, ci-dessous, un deuxième script Perl pour extraire le patron NOM ADJ.

```
#!/usr/bin/perl
```

```
open(FILE,"$ARGV[0]");
```

```
open (FILEOUT, ">sortie_SF_NOMADJ.txt");
```

```
my @lignes=<FILE>;
```

```
close(FILE);
```

```
while (@lignes) {
```

```
my $ligne=shift(@lignes);
```

```

chomp $ligne;

my $sequence="";

my $longueur=0;

if ( $ligne =~ /^(^[^t]+)\t^[^t]+\tNC.*/) {

my $forme=$1;

$sequence.=$forme;

$longueur=1;

my $nextligne=$lignes[0];

if ( $nextligne =~ /^(^[^t]+)\t^[^t]+\tADJ.*/) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=2;

}

}

if ($longueur == 2) {

print FILEOUT "$sequence\n";

}

}

close(FILE);

close(FILEOUT);

```

On remarque que ce script a le même mode de fonctionnement que le script que nous avons utilisé plus haut pour extraire le patron NOM ADJ à partir de la sortie étiquetée avec Treetagger. Cependant, une légère modification (en rouge) a été apportée. En effet, dans la sortie étiquetée avec Cordial, la spécificité du genre et du nombre doit être prise en compte. Une expression régulière (.*) est donc nécessaire pour extraire les patrons.

Par ailleurs, on obtient une liste composée de 128 patrons NOM ADJ. On constate qu'il y a une différence de quatre patrons, si on compare cette liste avec celle qu'on nous avons obtenue, plus haut, avec le premier script Perl. Ces quatre patrons sont les suivants :

- Exposition L' ;
- Exposition L' ;
- Bouclettes locales ;
- écran plus de.

Dans les trois premiers patrons, il y a des guillemets " , entre le nom et l'adjectif. Dans le dernier patron, on trouve un \r entre le nom et l'adjectif, comme on peut le voir dans les captures d'écran ci-dessous :

322	1930	1930	NCMIN	1553	de	de	PREP
323	,	,	PCTFAIB	1554	Las Vegas	Las Vegas	NPI
324	à	à	PREP	1555	ont	avoir	VINDP3P
325	Boulogne-Billancourt	Boulogne-Bil:		1556	concerné	concerner	VPARPMS
326	,	,	PCTFAIB	1557	les	le	DETDPIG
327	le	le	DETDIFS	1558	écrans	écran	NCMP
328	exposition	exposition	NCFS	1559	\r		
329	"			1560	Plus de	plus de	ADJIND
330	L' 1		ADJNUM	1561	200	200	NCMIN
331	esthétique	esthétique	NCFS	1562	personnes	personne	NCFP
332	médiatique	médiatique	ADJSIG	1563	se	se	PPER3S
333	"						
1431	de	de	PREP				
1432	"						
1433	bouclettes	bouclette	NCFP				
1434	"						
1435	locales	locale	ADJFP				
1436	de	de	PREP				
1437	elles	elles	NCFS				

Les expressions régulières qu'utilise ce deuxième script ne prennent pas en compte les cas où il y a des guillemets ou un \r entre la première et la deuxième séquence du patron NOM ADJ. Cependant, ce script est facilement adaptable pour extraire d'autres patrons.

suppression totale
 soldat américain
 magazine américain
 caractère exceptionnel
 procédure pénale
 football professionnel
 sport le plus
 esthétique domestique
 esthétique médiatique
 presse économique
 esthétique médiatique
 design étonnant
 monde entier
 site participatif
 pays le plus
 hebdomadaire annulée
 chaînes thématiques
 version initiale
 départs volontaires
 dérive partisane
 chaîne privée
 recettes publicitaires
 ministère des affaires étrangères canadien
 torture certaines
 techniques américaines
 militants antisectes
 groupe imaginaire
 vie commune
 journalistes français
 Point commun
 histoires croisées
 difficulté croissante
 voix nue
 mixage digital
 musique classique
 formation britannique

2- Extraction du patron NOM PREP NOM

Les deux scripts que nous allons voir, à présent, sont les mêmes que nous avons employés dans la partie précédente. La seule différence réside dans la spécification de la liste de patrons NOM PREP NOM pour le premier script Perl et la modification de l'expression régulière dans le deuxième script Perl.

* Extraction du patron NOM PREP NOM avec le premier script

Comme ce script n'a pas été modifié, nous présentons seulement la liste des patrons (53) NOM PREP NOM que l'on recherche.

NCFS PREP NCFS

NCMS PREP NCMS

NCMS PREP NCFS

NCFS PREP NCMS

NPFS PREP NCMS

NCMP PREP NCFS

NCFP PREP NCFS

NCMS PREP NCMP
NCMIN PREP NPSIG
NCMP PREP NCMP
NCFS PREP NPMS
NCFS PREP NPFS
NCFS PREP NPI
NCFS PREP NPSIG
NCMP PREP NCMIN
NCMS PREP NPMS
NPMS PREP NPI
NCFS PREP NCMP
NCFS PREP NCMIN
NCMS PREP NPFS
NCMS PREP NCFP
NPI PREP NCFS
NCMS PREP NCPIG
NCSIG PREP NCFS
NCFP PREP NCMP
NCMIN PREP NPMS
NCMP PREP NCI
NCMS PREP NPSIG
NPMS PREP NPSIG
NCMIN PREP NCFS
NCMP PREP NCFP
NCFP PREP NPI
NPI PREP NPMIN

NCFS PREP NPMIN
NCMS PREP NCI
NCMS PREP NCMIN
NCMIN PREP NPI
NCPIG PREP NPFS
NCMS PREP NPI
NCFP PREP NCPIG
NCMP PREP NPFS
NCFP PREP NCMIN
NCMP PREP NCMS
NCI PREP NPSIG
NCPIG PREP NCFP
NCFP PREP NPFS
NCI PREP NCMP
NCFP PREP NPMS
NCFS PREP NCFP
NCI PREP NCMIN
NCPIG PREP NCMIN
NPMS PREP NCMS
NPI PREP NCI

À travers la lecture de cette liste, nous pouvons remarquer que nous avons pris en compte toutes les combinaisons de NOM PREP NOM, y compris celles qui sont composées de noms propres. On obtient 129 patrons NOM PREP NOM.

Signalons, quand même, une anomalie au niveau de l'étiquetage de Cordial à la ligne 1875, laquelle a généré un mauvais patron NOM PREP NOM, comme le montre l'image ci-dessous. En effet, "Evoqué" n'est pas un NOM. Treetagger réagit, d'ailleurs, de la même manière en considérant "Evoqué" comme un NOM (ligne 1871).

```

1872 Dailymotion Dailymotion NPI
1873 . . PCTFORTE
1874 \r
1875 Evoquée Evoquée NPI => Erreur orthographique: Evoquée => Évoquée
1876 par par PREP
1877 Nicolas Nicolas NPMIN
1878 Sarkozy Sarkozy NPMS

```

Cette anomalie est due, selon Cordial, à une erreur d'orthographe.

On note également l'absence des patrons "holding du groupe", "propriétaire du journal" et "transformation du château", car du est reconnu comme un déterminant, alors que Treetagger l'étiquette comme un PRP:det.

```

moteur de recherche en ligne
millions d' euros
formats de DVD
carnet en ligne
plateaux de télévision
reportage pour ARTE
Trocadéro à Paris
comité de soutien
ligne depuis lundi
réseaux de convivialité
candidature d' Eric
2007 en compagnie
comité de soutien
abus de position
mairie d' Asnières
jours en tête des vidéos
vues sur Dailymotion
Evoquée par Nicolas
journal télévisé en ligne
société d' électronique
marque de luxe
candidature d' Eric
opérateur de téléphonie
remise en liberté
tribunal de grande instance de Paris
demande de suspension
Ligue de football
droits de diffusion

```

* Extraction du patron NOM PREP NOM avec le deuxième script Perl

```
#!/usr/bin/perl
```

```
open(FILE,"$ARGV[0]");
```

```
open (FILEOUT, ">sortie_SF_NOMPRPNOM.txt");
```

```
my @lignes=<FILE>;
```

```

close(FILE);

while (@lignes) {

my $ligne=shift(@lignes);

chomp $ligne;

my $sequence="";

my $longueur=0;

if ( $ligne =~ /^(^[^t]+)\t([^\t]+\tN.*)/ ) {

my $forme=$1;

$sequence.=$forme;

$longueur=1;

my $nextligne=$lignes[0];

if ( $nextligne =~ /^(^[^t]+)\t([^\t]+\tPREP)/ ) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=2;

my $nextligne=$lignes[1];

if ( $nextligne =~ /^(^[^t]+)\t([^\t]+\tN.*)/ ) {

my $forme=$1;

$sequence.=" ".$forme;

$longueur=3;

}

}

}

if ($longueur == 3) {

print FILEOUT "$sequence\n";

}

}

```

```

}

close(FILE);

close(FILEOUT);

```

Ce script prend en entrée le fichier étiqueté avec Cordial et génère une liste de patrons NOM PREP NOM. Trois boucles if sont utilisées pour aboutir à ce résultat. On peut constater, en rouge dans la première et troisième boucle if, l'expression régulière (.**) pour récupérer toutes les combinaisons possibles. Rappelons que le métacaractère signifie un caractère quelconque, sauf \n (comportement par défaut, modifiable). Le métacaractère * marque la possible répétition du caractère précédent (ou de l'expression précédente entre parenthèses). On obtient alors 129 patrons NOM PREP NOM.

```

abus de convivialité
candidature d' Eric
2007 en compagnie
comité de soutien
abus de position
mairie d' Asnières
jours en tête des vidéos
vues sur Dailymotion
Evoquée par Nicolas
journal télévisé en ligne
société d' électronique
marque de luxe
candidature d' Eric
opérateur de téléphonie
remise en liberté
tribunal de grande instance de Paris
demande de suspension
Ligue de football
droits de diffusion
millions d' abonnés
château en lieu
lieu de pouvoir
annonce de Nicolas
site d' enchères
enchères en ligne
marché de gros
industrie en 2002
directoire de M6
remise à plat
financement de France
millions d' euros
euros de recettes
intersyndicale de France
projet de M.
M. de Cordia

```

Ce résultat montre que pour le patron NOM PREP NOM, l'utilisation de deux scripts différents permet d'avoir le même nombre de patrons, sans doute à cause de l'utilisation d'un même fichier étiqueté. Ce qui n'est pas le cas pour le patron NOM ADJ.

Rappelons qu'il est toujours possible d'appliquer des filtres aux sorties, afin d'optimiser et d'affiner les résultats. On augmente ainsi la précision et on diminue le bruit. On peut, par

exemple, exclure certaines extractions qui contiennent des chiffres ou des éléments contenus dans des listes d'exclusion (stop-list)⁵⁰.

Conclusion

Dans cet article, nous avons présenté une méthodologie d'extraction de termes à partir d'un corpus. Elle revendique son caractère semi-automatique, car la validation des candidats termes par l'expert est nécessaire pour ne garder que les termes pertinents. L'approche adoptée est symbolique et se base sur des connaissances linguistiques. Le résultat obtenu peut être amélioré moyennant quelques traitements supplémentaires (filtrage).

Les techniques de la linguistique de corpus et les outils du TAL apportent un gain de temps considérable au terminologue dans son travail de dépouillement de corpus, à des fins de constitution de dictionnaires terminologiques.

Bibliographie

- 1- Bourigault (Didier), « LEXTER, un Logiciel d'Extraction de TERminologie », Actes du colloque sur le Repérage de l'information textuelle, Ministère des Communications du Gouvernement du Québec et Hydro-Québec, Montréal, 1991, p. 15-25.
- 2- Daille (Béatrice), *Approche mixte pour l'extraction de terminologie : statistique lexicale et filtres linguistiques*, thèse en Informatique Fondamentale, Université de Paris 7, Paris, 1994.
- 3- Slodzian (Monique), « L'émergence d'une terminologie textuelle et le retour du sens », Le sens en terminologie, (sous la dir. de H. Béjoint et P. Thoirion, Travaux du C.R.T.T), Presses Universitaires de Lyon, Lyon, 2000, p. 61-85.

⁵⁰ Ces listes appelées stop-list ou listes d'exclusion sont faciles à élaborer attendu qu'elles recèlent principalement les mots grammaticaux (stop word ou mot vide) de la langue utilisée, termes en nombre fini et exhaustivement connus. Ces listes doivent être adaptées aux types de patrons extraits.