[SL4BY020]

Corpus : outils d'analyse automatique

Séance 5/6 − 8 avril

Introduction

Pour ce premier de nos deux TDs, nous allons simplement utiliser TreeTagger en *ligne de commande*, c’est à dire sans recourir à une enveloppe/*wrapper* comme celle qui est disponible pour Python. Ceci nous permettra de :

- Bien décomposer les étapes du traitement, notamment ce qui est fait par TreeTagger, et ce qu’il faut préparer en amont.
- Décortiquer le fonctionnement de TreeTagger.
- Ouvrir la voie à une utilisation avancée, pour faire apprendre de nouvelles choses à TreeTagger (ce qu’on verra lors du 2e TD).

Organisation du TD :

- Pages 3 à 9: c'est du "cours" vous pouvez l'imprimer et le lire où vous voulez, il n'y a pas besoin d'ordi. Il faut bien l'avoir assimilé avant d'attaquer la pratique. Temps de travail estimé: ~30min.
- Pages 10 à 18: c'est de l'installation. Je détaille beaucoup, avec plein de captures d'écran, c'est pour ça que c'est long ! Si tout se passe bien vous en aurez pour ~20 min. Attention toutefois, il y a autour de 500 Mo à télécharger si vous êtes sur Windows, et là ça dépend beaucoup de votre connexion...
- Pages 19 à 27: c'est la pratique ! Là aussi, il y a beaucoup d'explications et de captures d'écran; en pratique, ça ne devrait pas être très long ! Temps de travail estimé: ~1h.

Pourquoi TreeTagger ?

TreeTagger n'est pas le logiciel d'annotation automatique (*tagger*) le plus efficace, mais c'est en quelque sorte *le* logiciel d'annotation *classique* (créé en 1995). C'est l'un des plus simples à comprendre et utiliser, et bien que n'étant pas le meilleur, il est « suffisamment bon » pour de nombreux travaux de recherche en linguistique ; tout le monde le connaît, et il est toujours couramment utilisé.

Origine du nom TreeTagger. TreeTagger est un logiciel d'annotation automatique (*tagger*), dont l'algorithme est basé sur des arbres de décision, une approche connue pour sa rapidité.

Fonctionnement général de TreeTagger

Comme la plupart des *taggers*, TreeTagger part d'un séquence de tokens :

Tokens
Chaque
être
humain
naît
libre
.

… et produit une annotation de type forme/POS[1]/lemme :

Forme	POS	Lemme
Chaque	PRO:IND	chaque
être	NOM	être
humain	ADJ	humain
naît	VER:pres	naître
libre	ADJ	libre
.	SENT	.

On remarquera dès à présent deux choses (cf. ci-après tokenisation et désambiguïsation).

1 POS : *Part of Speech*, ou Partie du Discours

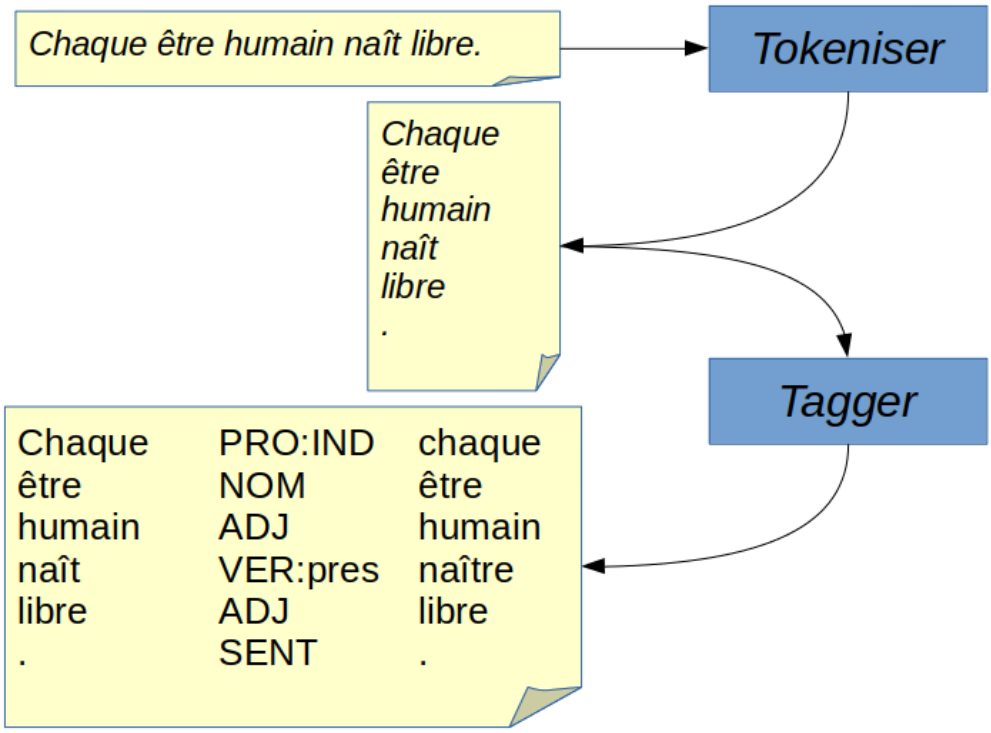
Tokenisation

Le texte doit être préalablement découpé en *tokens* (on parle de tokénisation − voir schéma ci-dessous) avant d'être donné à TreeTagger, qui ne sait pas faire ce découpage. En pratique, TreeTagger est distribué avec un petit script de tokénisation très simple : il découpe les tokens en fonction des espaces typographiques, des signes de ponctuation, et de quelques règles spécifiques à la langue ; par exemple, pour le français :

```
# Vous n'avez pas besoin de comprendre ces règles… juste qu'elles sont très courtes !
$PClitic = "(?:[dcjlmnstDCJLNMST]|[Qq]u|[Jj]usqu|[Ll]orsqu)['’´]";
$FClitic = "-t-elles?|-t-ils?|-t-on|-ce|-elles?|-ils?|-je|-la|-les?|-
leur|-lui|-mmes?|-m['’´]|-moi|-nous|-on|-toi|-tu|-t['’´]|-vous|-en|-y|-
ci|-là";
```

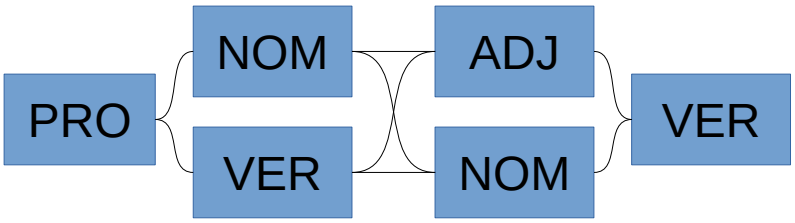
Ceci explique que TreeTagger soit facilement utilisable sur les langues écrites avec des espaces entre les mots, puisque la tokénisation n'est pas difficile et peut se faire avec un simple petit script. Par contre, sur des langues qui n'utilisent pas de séparateur entre les mots (p. ex. chinois : pas de séparateur entre les mots ou à l'inverse le vietnamien : il y a des blancs, mais pas entre les mots, mais entre les… syllabes !), la tokénisation devient problématique, et TreeTagger est moins utilisé pour ces langues.

Schématiquement, la tokenisation (effectuée par un script) et le *tagging* (effectué par *TreeTagger*) s'enchaînent comme suit :



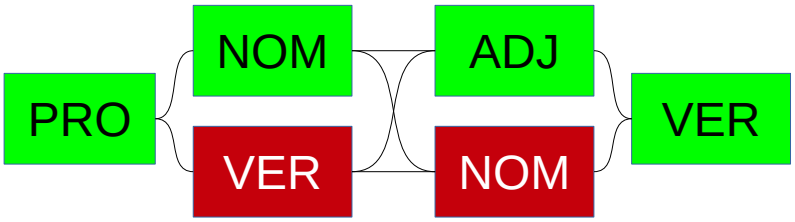
Désambiguïsation

Comme on le voit dans l'exemple précédent, le token *être* peut être à la fois un nom ou un verbe, tandis que le token suivant, *humain*, peut être un nom ou un adjectif : on parle d'*ambiguïtés*. Par contre, *chaque* et *naît* ne sont pas ambigus. TreeTagger s'appuie sur un dictionnaire du français pour déterminer si il y a ambiguïté, et son travail va consister à lever ces ambiguïtés. Schématiquement, grâce à son dictionnaire du français, TreeTagger sait qu'il doit choisir un « chemin » dans le graphe suivant :



(pour simplifier, je note PRO à la place de PRO:IND, et VER à la place de VER:pres)

Pour cela, *TreeTagger* va s'appuyer sur (1) le contexte des parties non ambiguës et (2) un modèle de langage (nous en reparlerons) pour déterminer le chemin le plus probable :



Le modèle de langage (*parameter file* dans le jargon de TreeTagger), qui permet de calculer la probabilité d'un chemin, est basé sur un dictionnaire et un **corpus d'apprentissage**. Il s'agit d'un petit corpus (~50 000 tokens), qui a été annoté « à la main » par des annotateurs, et dont TreeTagger va extraire des probabilités.

Exemple de corpus d'apprentissage :

Forme	POS	Lemme
Quelque	PRO:IND	quelque
animal	NOM	animal
marin	ADJ	marin
vient	VER:pres	venir

(c'est un extrait, un vrai corpus d'apprentissage fait environ 50 000 tokens, soit quelques mois de boulot !)

Si vous trouvez que ce *corpus d'apprentissage* ressemble beaucoup au type d'*annotation* produit par TreeTagger, c'est normal ! C'est exactement la même chose. Ce corpus d'apprentissage, vous pouvez le voir comme un « exemple » que TreeTagger va « apprendre ».

En l'occurrence, disons qu'à partir de son corpus d'apprentissage, TreeTagger a appris les probabilités suivantes (les chiffres sont fictifs) :

9	PRO	NOM	ADJ	VER
0	PRO	VER	ADJ	VER
0	PRO	NOM	NOM	VER
3	PRO	VER	NOM	VER

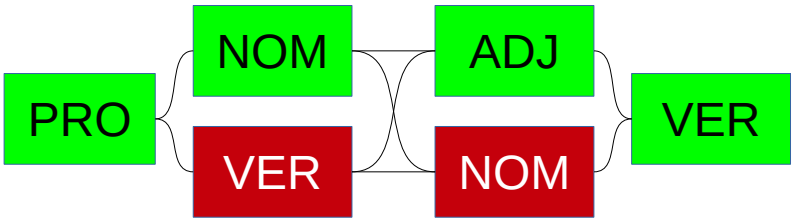
C'est à dire que dans ce corpus d'apprentissage (qui, rappelons-le, n'est *pas* le corpus qu'on cherche à annoter, mais un corpus déjà existant, annoté « manuellement » par des experts), TreeTagger a comptabilisé :

- 9 séquences PRO, NOM, ADJ, VER
- 3 séquences PRO, VER, NOM, VER
- et jamais les deux autres chemins possibles.

Il conclut donc naturellement que la séquence PRO, NOM, ADJ, VER est la plus probable :

	PRO	NOM	ADJ	VER
	PRO	VER	ADJ	VER
	PRO	NOM	NOM	VER
	PRO	VER	NOM	VER

… et donc que c’est le « chemin » suivant qui est le « bon » :

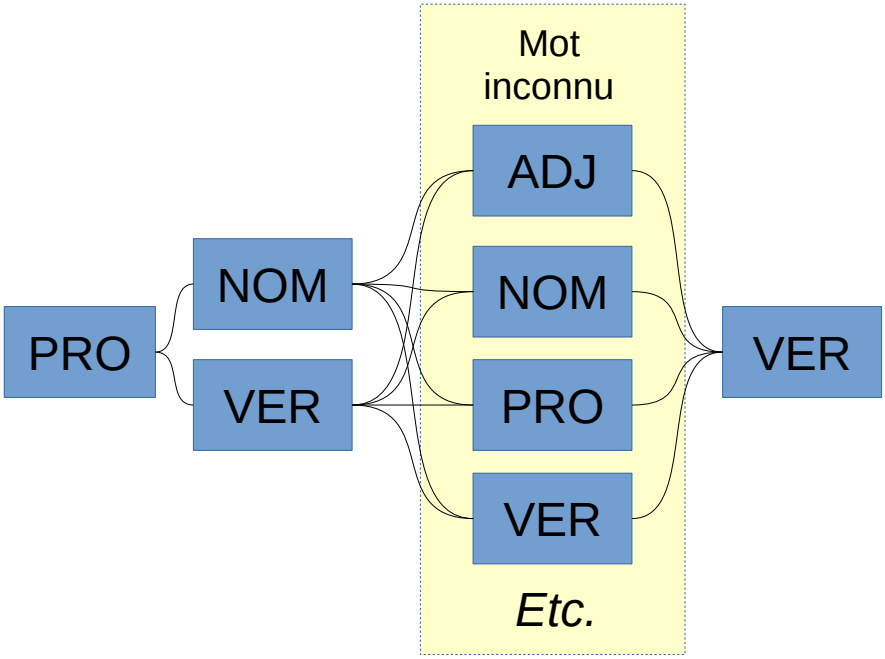


Notez-bien qu’il s’agit là d’une explication simplifiée du fonctionnement de TreeTagger, mais elle est suffisante pour appréhender le fonctionnement et les plupart des erreurs que TreeTagger peut commettre.

Les cas des mots inconnus

Que se passe-t-il quand un mot n'est pas dans le dictionnaire ? Avec les noms propres par exemple, c'est assez fréquent !

Dans ce cas, TreeTagger va simplement chercher un chemin entre toutes les POS possibles, en fonction des probabilités qui ressortent du modèle de langage :



En réalité, TreeTagger va aussi tenir compte des préfixes/suffixes qu'il a pu observer dans le corpus d'apprentissage. Par exemple, un mot se terminant en *ant* aura un peu plus de chances d'être un verbe participe présent… mais c'est toujours le contexte qui a le dernier mot, comme on l'a illustré précédemment.

Le cas des lemmes

On notera que dans l'exemple précédent, on ne parle que des POS. Que ce passe-t-il en cas d'ambiguïté entre des lemmes ?

Dans la plupart des cas, l'ambiguïté peut être levée par la POS : *est*+VER a forcément le lemme *être*, et *est*+NOM a le lemme *est*.

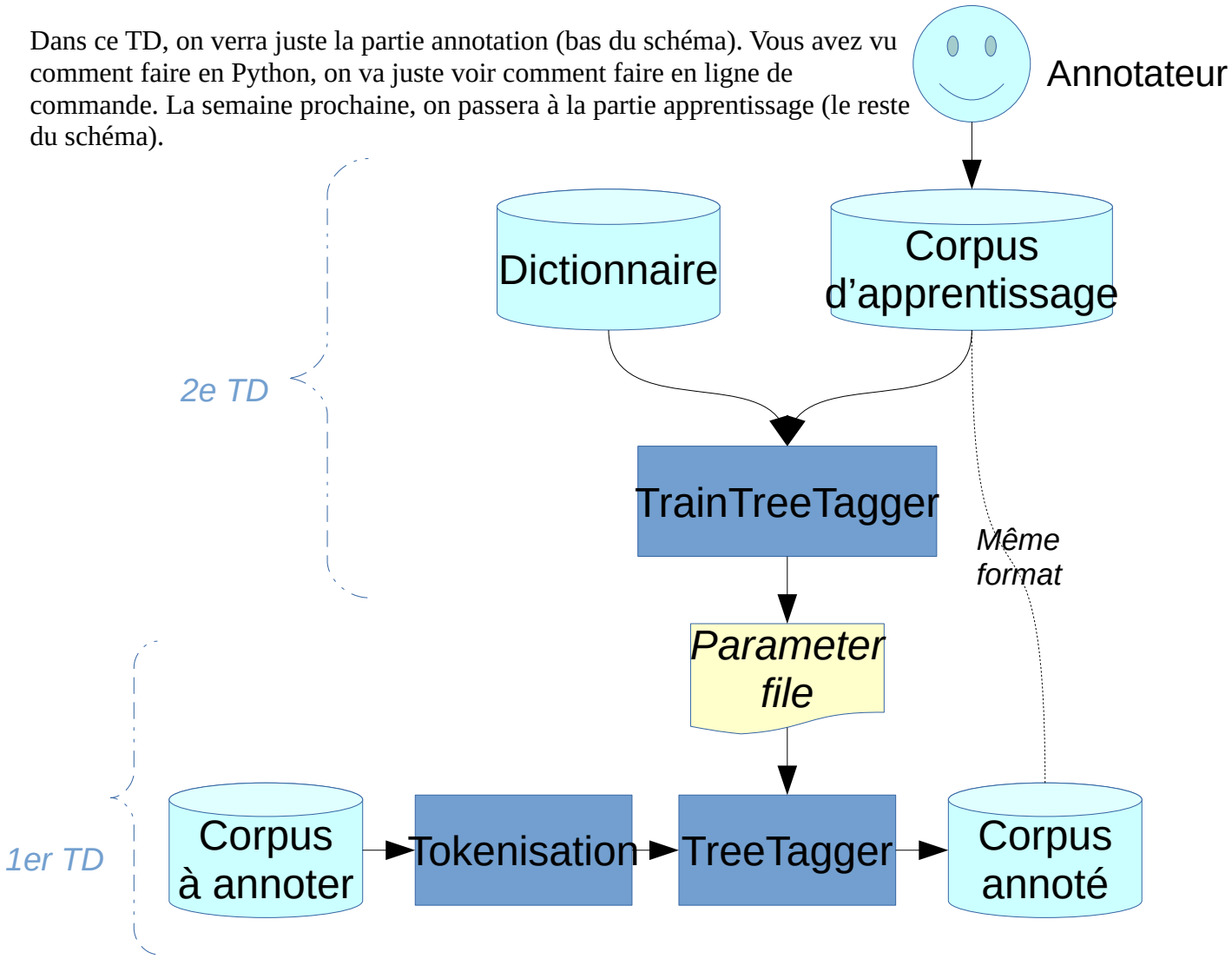
Mais dans le cas où ce n'est pas possible, par exemple pour *fils*+NOM (des *fils de laine* −lemme *fil*−, ou bien des *fils à papa* −lemme *fils*− ? Ce sont deux lemmes différents, mais tous deux des noms !), TreeTagger ne choisit pas ! Ainsi pour la forme *fils* avec la POS NOM, il va regarder dans son dictionnaire, voir que cela peut correspondre aux lemmes *fil* et *fils*, et noter comme lemme *fil*|*fils* (avec une barre verticale). TreeTagger ne calcule des probabilités que sur les POS, les lemmes sont justes récupérés dans le dictionnaire à la fin du traitement. En cas d'ambiguïté, pas grave ! on colle tout à la suite, avec des barres verticales comme séparateur.

Cela pose assez peu de problèmes pour les langues dont l'orthographe permet de lever la plupart des ambiguïtés, comme le français.

Récapitulons

Pour récapituler, une chaîne de traitement TreeTagger complète (comprenant l'apprentissage) ressemble à ceci :

Dans ce TD, on verra juste la partie annotation (bas du schéma). Vous avez vu comment faire en Python, on va juste voir comment faire en ligne de commande. La semaine prochaine, on passera à la partie apprentissage (le reste du schéma).

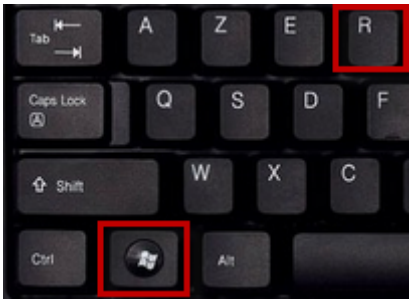


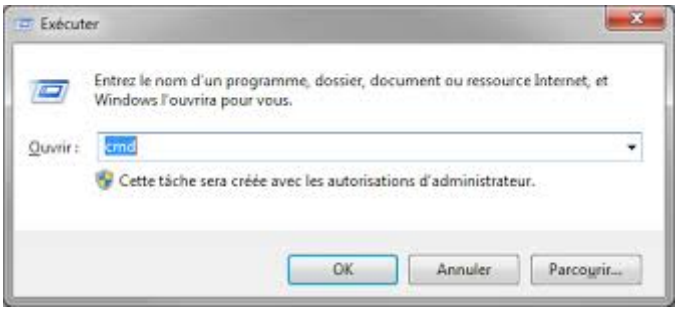
Utilisation de la ligne de commande

Démarrage de la ligne de commande (« invite de commande », « terminal »)

La façon d’accéder à la ligne de commande varie selon votre système d’exploitation.

Sur Windows 7, 8 et 10, utilisez la combinaison de touches Win+R. Une fenêtre apparaît, dans laquelle vous tapez « cmd » (sans le guillemets) et appuyez sur la touche OK.

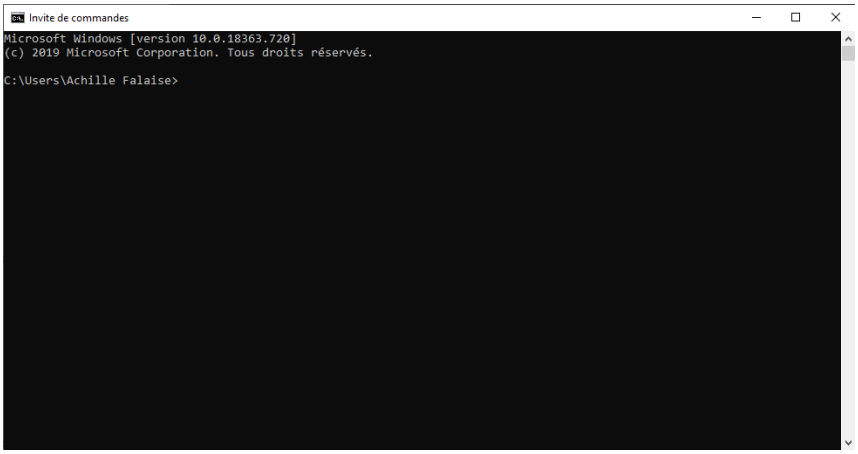


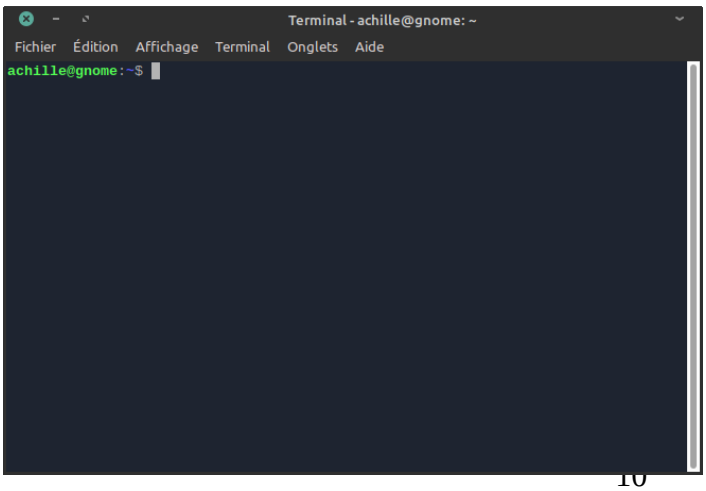


Sur MacOS, rendez-vous dans le dossier « Applications », puis allez dans « Utilitaires ». Vous n'aurez plus qu'à l'ouvrir, comme n'importe quelle autre application.

Sur Linux vous avez forcément un truc qui s’appelle Terminal dans votre menu principal.

Normalement, vous arrivez sur un logiciel en mode texte, qui affiche quelque chose du genre :





Aller sur le Bureau

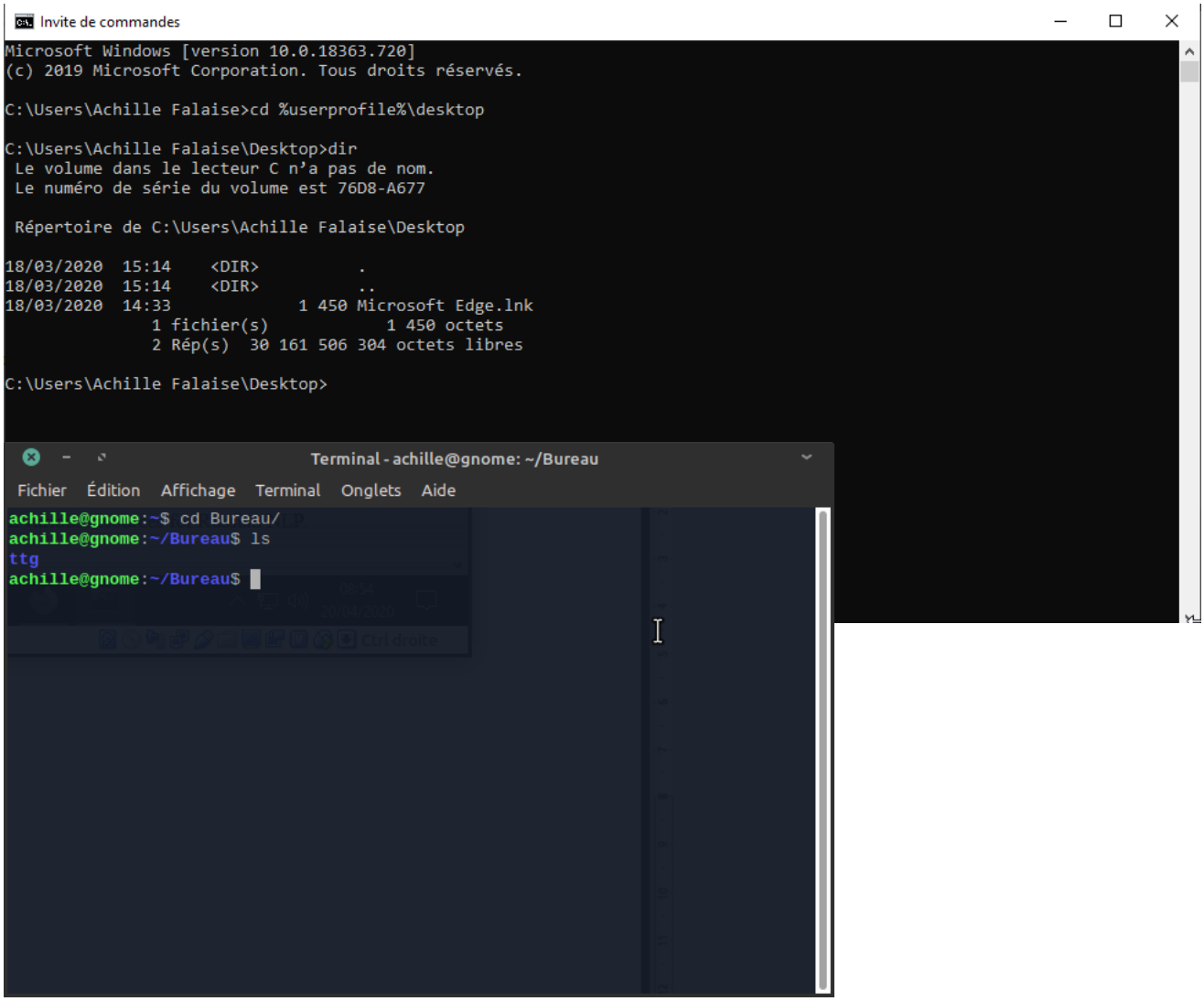
Ensuite, on va naviguer dans cette ligne de commande jusqu'à arriver sur le Bureau.

- Sur Windows une de ces deux commandes devrait fonctionner (remplacer username par votre nom d'utilisateur) :
  - `cd %userprofile%\desktop`
  - `c:\Users\username\Desktop`
- Sur MacOS et Linux, selon que votre Bureau s'appelle Bureau ou Desktop (à adapter en fonction de la langue) :
  - `cd ~/Bureau`
  - `cd ~/Desktop`

Pour vérifier que vous êtes bien sur votre Bureau, tapez ensuite la commande :

- Sur Windows
  - `dir`
- Sur MacOS et Linux
  - `ls`

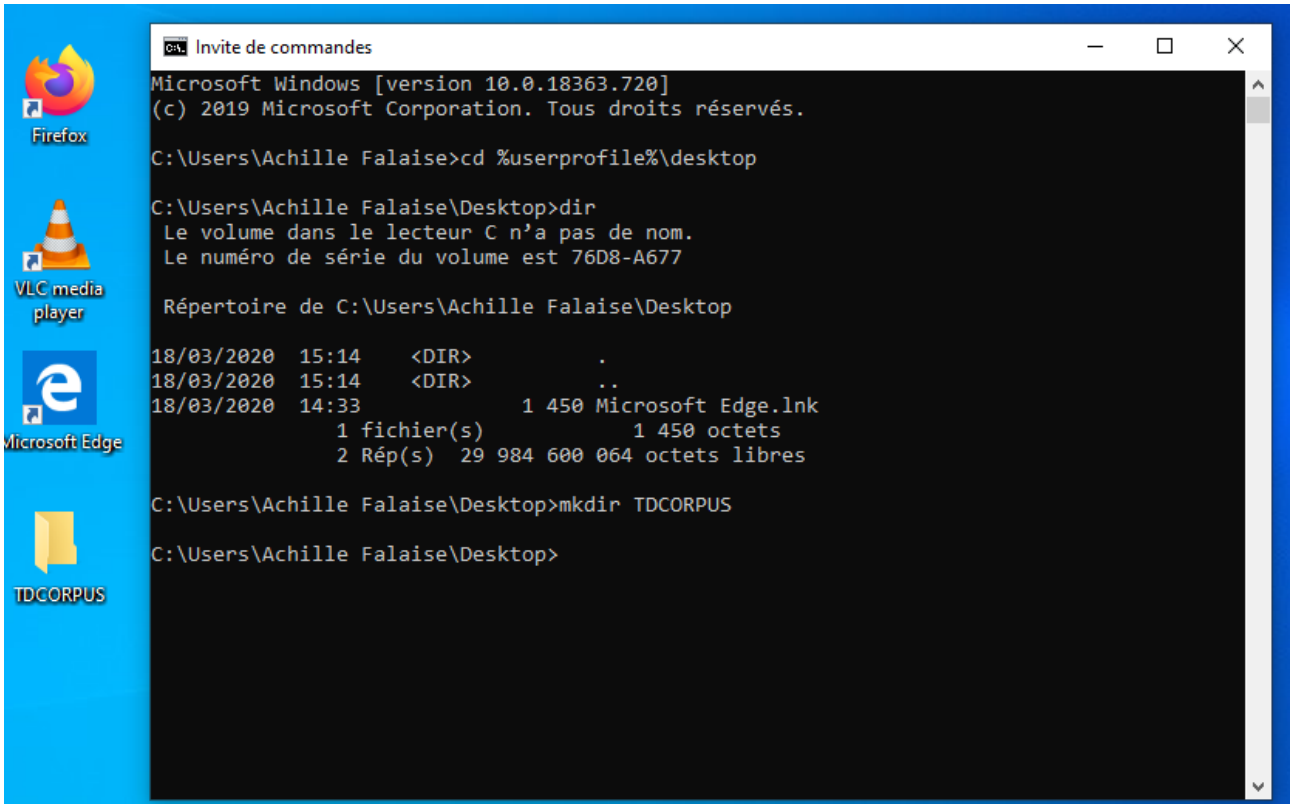
… cette commande affiche la liste des fichiers présents sur votre Bureau.

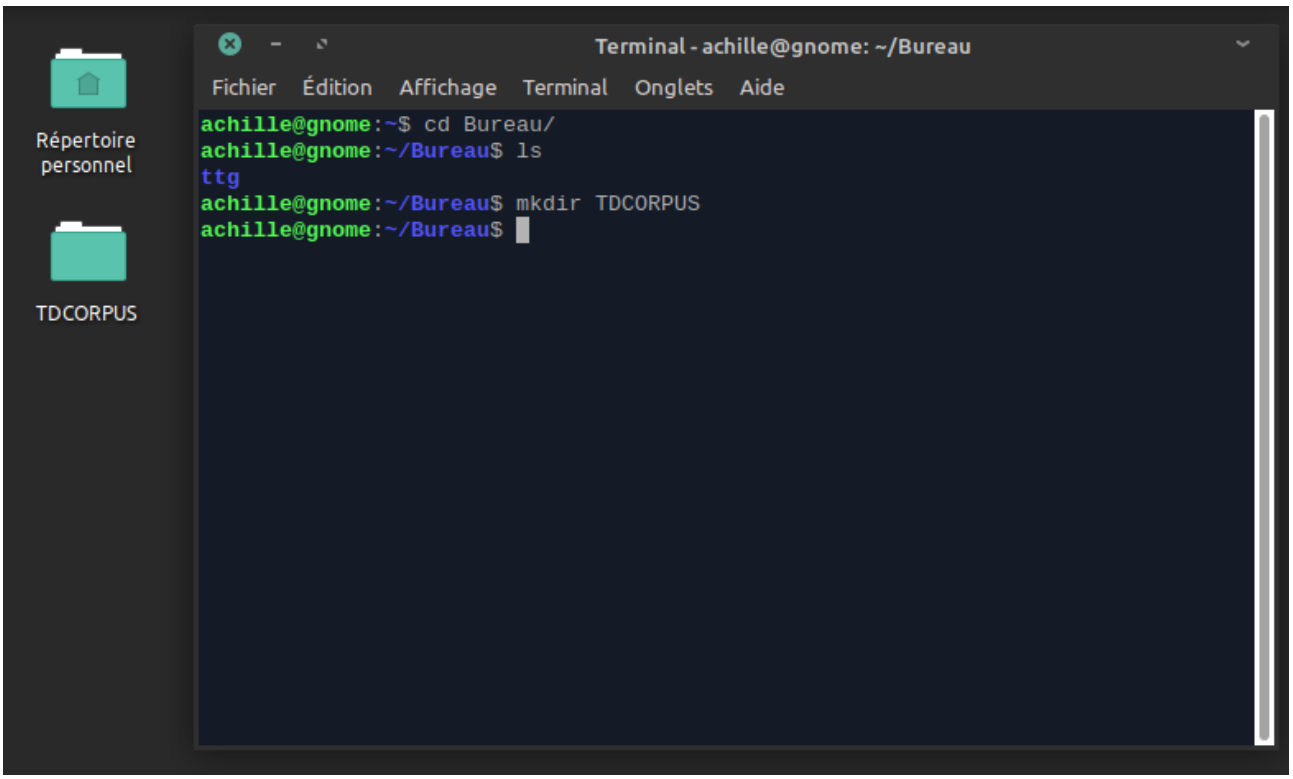


Maintenant, on va créer un dossier pour le TD. On va l'appeler TDCORPUS, et pour être sûr que tout fonctionne, *on va le créer en ligne de commande*. C'est la même commande sur tous les systèmes :

- `mkdir TDCORPUS`

Un dossier TDCORPUS devrait alors apparaître sur votre Bureau ! Voilà, nous sommes prêts à travailler.





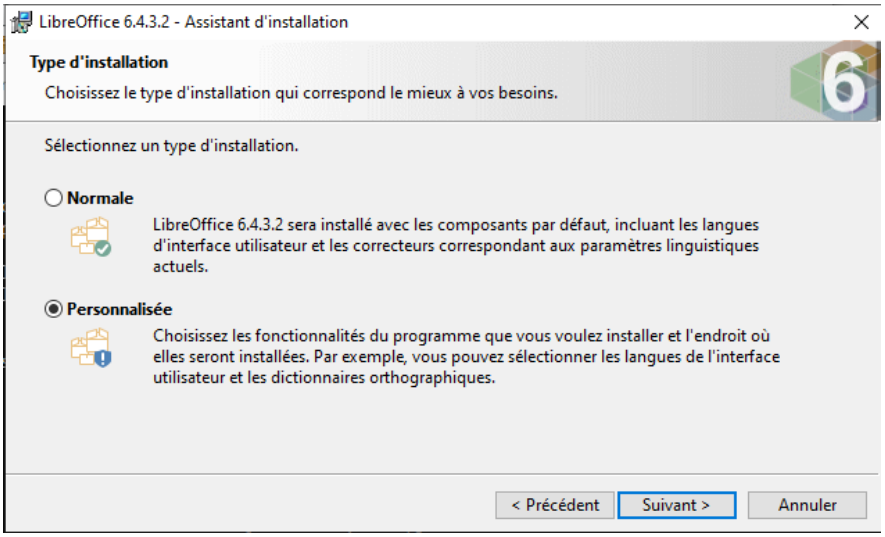
Installer LibreOffice

LibreOffice va surtout servir pour le TD 2. Ce n’est pas (trop) bloquant si vous ne l’avez pas pour le TD 1.

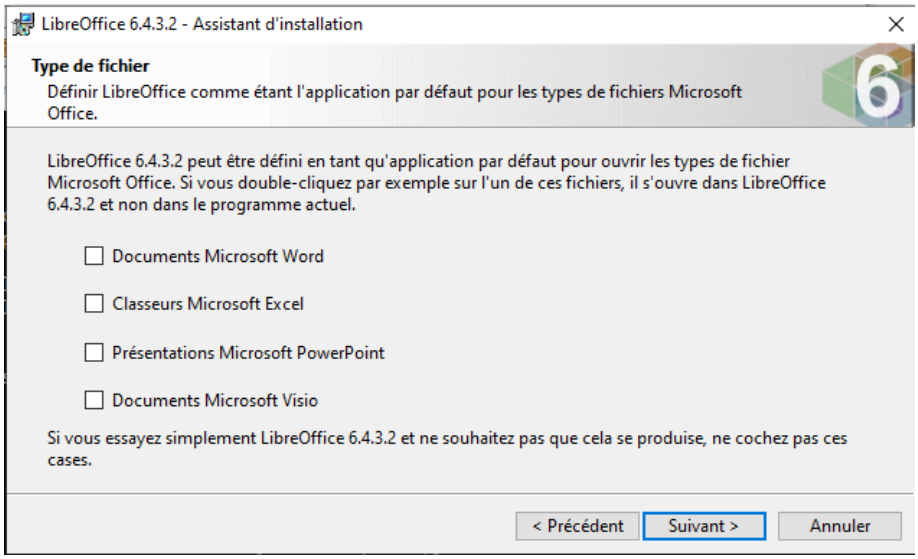
À télécharger ici :

https://fr.libreoffice.org/download/telecharger-libreoffice/

Si vous avez déjà Microsoft Office, ce n’est pas suffisant ! Nous allons avoir besoin d’une fonction qui marche très mal sur Microsoft Office. Si vous ne voulez pas associer tous les fichiers Microsoft Office avec LibreOffice, choisissez l’installation personnalisée :



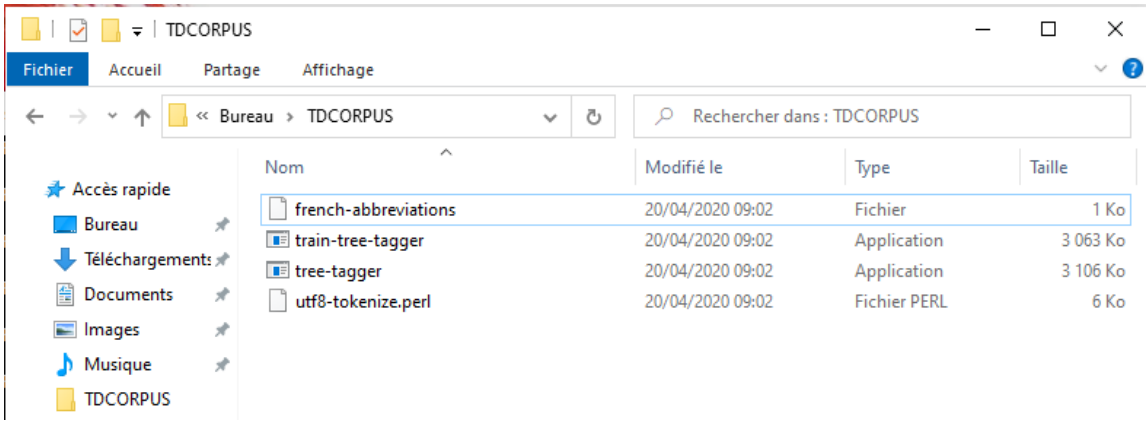
Puis décochez les cases d’association :



Installer TreeTagger

Sur Windows

1. Télécharger et installer Perl :
   - http://strawberryperl.com/
   - Test : ouvrez une ligne de commande (une nouvelle fenêtre ; si vous avez une fenêtre déjà ouverte, elle ne tient pas compte de l'installation), et tapez la commande :
     - `perl -e "print 'Test';"`
2. Télécharger et installer TreeTagger (chercher « Windows version ») :
   - https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/
   - Dans l'archive, récupérez les fichiers suivants, et placez-les dans votre dossier TDCORPUS :
     - bin/train-tree-tagger ← servira pour le TD n° 2
     - bin/tree-tagger
     - cmd/utf8-tokenize.perl
     - lib/french-abbreviations ← contient des erreurs d'encodage, mais on fera avec…
     - votre dossier TDCORPUS doit ressembler à ça :

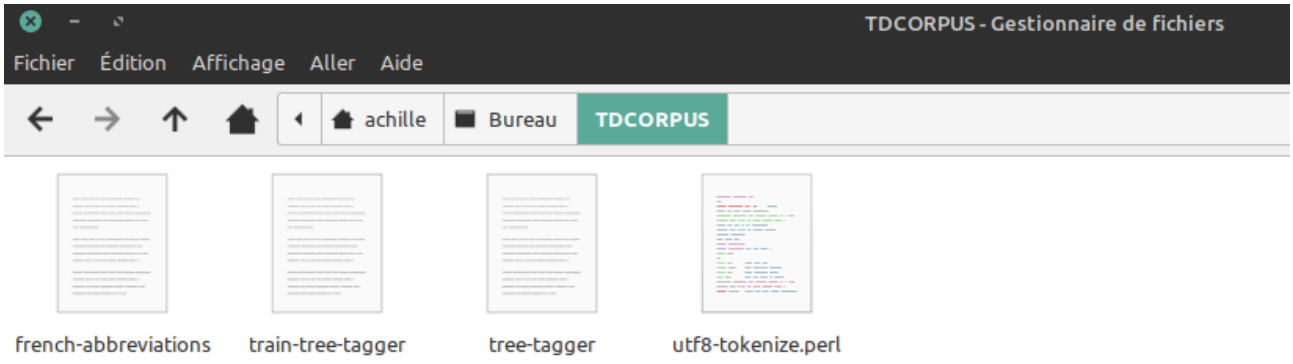


Sur MacOS et Linux

1. Normalement vous avez déjà Perl. Testez en ligne de commande avec :
    - `perl -e "print 'Test';"`
2. Télécharger et installer TreeTagger (« Download the tagger package for your system » : « Mac OS-X » ou « PC-Linux ») et les « tagging scripts ».
    - https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/
    - Seulement TreeTagger et les tagging scripts (étapes n°1 et 2 sur le site), on ne veut pas du reste :

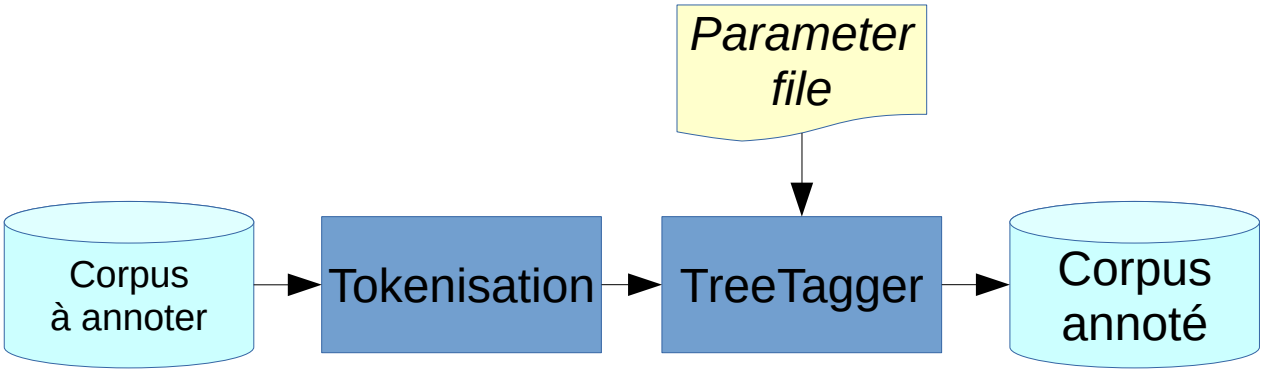
1. Download the tagger package for your system (PC-Linux, Mac OS-X, ARM64, ARMHF, ARM-Android, PPC64le-Linux).
   If you have problems with your Linux kernel version, download this older Linux version and rename it to tree-tagger-linux-3.2.2.tar.gz.
2. Download the tagging scripts into the same directory.

    - Dans les archives (2 archives : TreeTagger et les tagging scripts), récupérez les fichiers suivants, et placez-les dans votre dossier TDCORPUS :
        - bin/train-tree-tagger ← servira pour le TD n° 2
        - bin/tree-tagger
        - cmd/utf8-tokenize.perl
        - lib/french-abbreviations ← contient des erreurs d'encodage, mais on fera avec…
        - votre dossier TDCORPUS doit ressembler à ça :



Télécharger les ressources

Nous avons le script de Tokenisation et TreeTagger. Maintenant, il nous faut *un parameter file* et un corpus à annoter !



Pour pouvoir faire des comparaisons, nous allons télécharger 3 modèles de langage (*parameter files*), avec la description de leur jeu d'étiquettes :

1. Le modèle distribué sur le site de TreeTagger. Il provient d'un apprentissage sur le journal Le Monde:
    - Modèle : https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french.par.gz
        - Décompressez-le pour avoir un fichier avec l'extension .par !
        - Sur Windows : si Windows ne comprend pas que l'extension .gz est une archive, vous aurez besoin de l'utilitaire https://peazip.org
    - Doc : https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french-tagset.html
2. Le modèle Presto, entraîné sur des textes littéraires français du XVIe-XVIIIe siècle.
    - Modèle : http://presto.aiakide.net/downloads/models/ttg/2014-10-31/Mod%C3%A8le_Presto-2014-10-31.par
    - Doc : http://presto.ens-lyon.fr/wp-content/uploads/2014/05/%C3%89tiquettes_Presto-2014-10-13.pdf
3. Le modèle Perceo, entraîné sur des transcriptions de français parlé.
    - Modèle et doc : https://www.ortolang.fr/market/corpora/perceo
        - Cliquer sur « Télécharger », et après le téléchargement, dans l'archive, que vous obtiendrez, on veut juste perceo_oral/spoken-french.par
        - Cliquer sur « Documentation » pour télécharger la doc en pdf.

Il n'existe pas de définition universelle de ce qu'est un *token*. Par conséquent, la manière exacte de tokeniser varie en fonction du modèle de langage : par exemple, la locution adverbiale *par conséquent* va être considérée comme un seul token par certains modèles, et comme deux tokens par d'autres modèles.

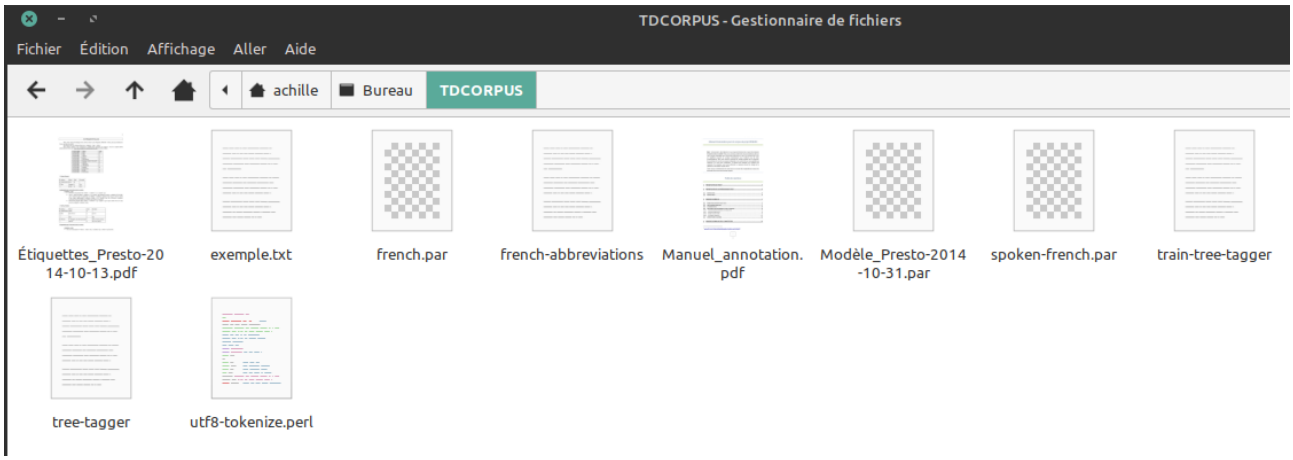
Si la tokenisation et le modèle de langage ne correspondent pas parfaitement, on va avoir des erreurs provenant du fait que les tokens ne sont pas trouvés dans le dictionnaire. Notamment, des mots « inconnus ».

Le script de tokenisation que nous avons est conçu pour le modèle de langage distribué avec TreeTagger. Il ne fonctionnera pas parfaitement avec les autres modèles. Toutefois, nous n'allons pas installer les scripts de tokenisation pour les autres modèles, parce que cela prendrait trop de temps de les installer sur vos machines ! Ça ira bien pour un TD, mais si un jour vous avez besoin d'utiliser « sérieusement » ces modèles, vous aurez besoin d'utiliser le script de tokenisation qui va avec.

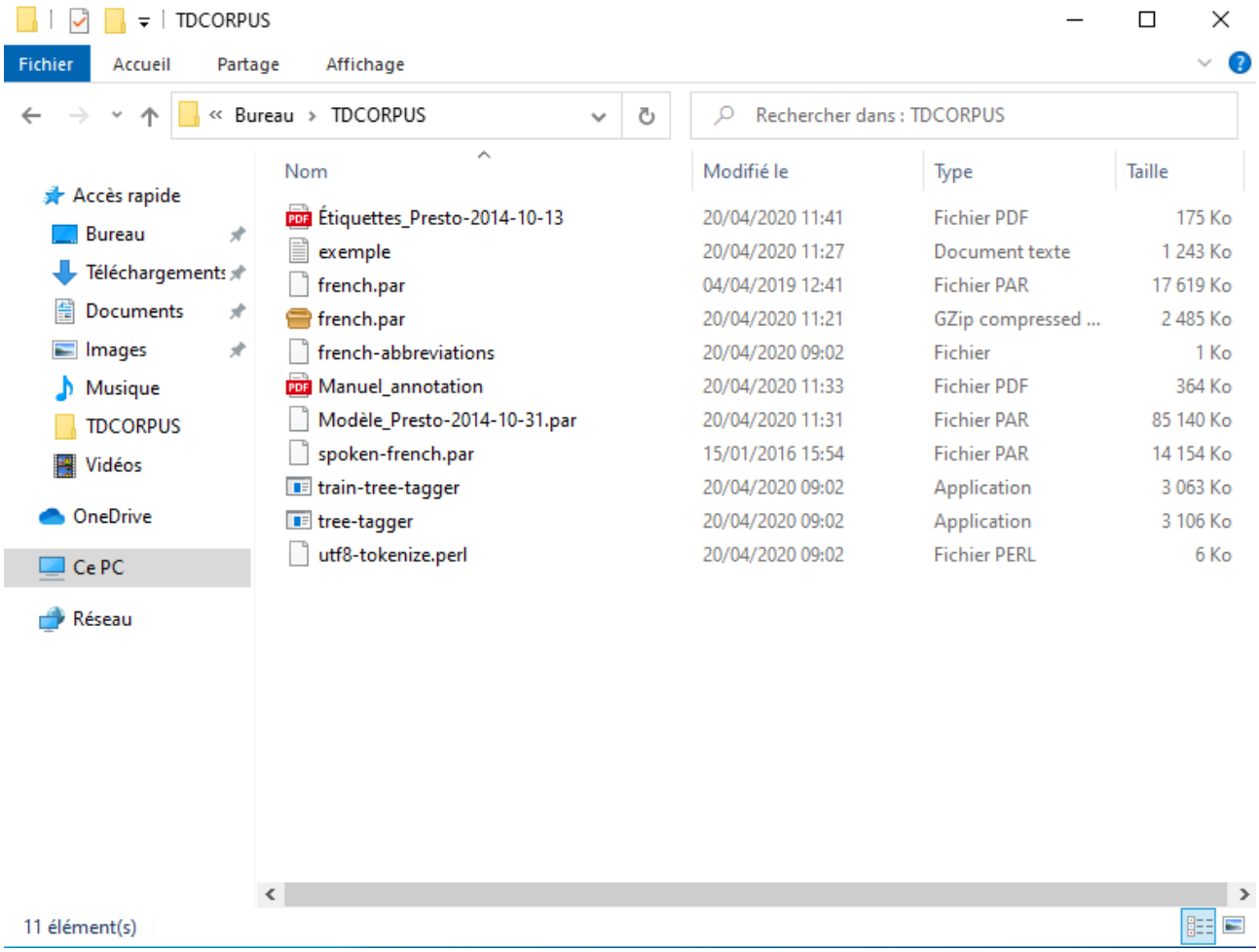
Comme corpus à annoter, nous allons prendre *L'Île mystérieuse* de Jules Verne :

- https://pro.aiakide.net/cours/L2016/exemple.txt

Votre dossier TDCORPUS doit ressembler à ça :



Sur Windows : notez bien que par défaut, Windows masque certaines extensions de fichiers (mais pas toutes !). Donc dans la capture ci-dessous, vous avec un fichier *french.par* décompressé, et aussi une archive *french.par*, qui est en fait *french.par.gz*, mais dont le *.gz* est masqué.



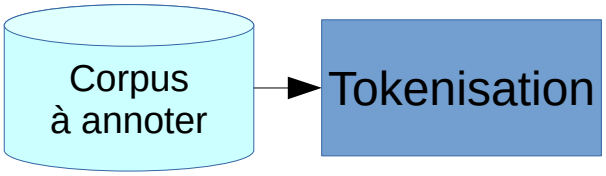
Seule la ligne de commande vous dit ce que vous avez *vraiment* dans le dossier.

```
C:\Users\Achille Falaise\Desktop\TDCORPUS>dir
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est 76D8-A677

 Répertoire de C:\Users\Achille Falaise\Desktop\TDCORPUS

20/04/2020  11:41    <DIR>          .
20/04/2020  11:41    <DIR>          ..
20/04/2020  11:27         1 272 224 exemple.txt
20/04/2020  09:02               388 french-abbreviations
04/04/2019  12:41        18 040 928 french.par
20/04/2020  11:21         2 544 479 french.par.gz
20/04/2020  11:33           372 600 Manuel_annotation.pdf
20/04/2020  11:31        87 182 594 Modèle_Presto-2014-10-31.par
15/01/2016  16:54        14 493 539 spoken-french.par
20/04/2020  09:02         3 136 348 train-tree-tagger.exe
20/04/2020  09:02         3 180 249 tree-tagger.exe
20/04/2020  09:02             5 308 utf8-tokenize.perl
20/04/2020  11:41           178 664 Étiquettes_Presto-2014-10-13.pdf
              11 fichier(s)      130 407 321 octets
               2 Rép(s)  29 249 089 536 octets libres
```

Utilisation de TreeTagger en ligne de commande

Je donne des captures d'écran pour Linux, mais vous aurez les mêmes résultats sur les autres systèmes. Juste des couleurs un peu différentes.

Tokéniser



Nous allons d'abord commencer par afficher du texte en ligne de commande :

- `echo "Ceci est un test."`

```
achille@gnome:~/Bureau/TDCORPUS$ echo "Ceci est un test."
Ceci est un test.
```

Ensuite, au lieu d'afficher ce texte, nous allons le tokéniser, et afficher le résultat :

- `echo "Ceci est un test." | perl utf8-tokenize.perl`
- Notez l'utilisation du caractère « | » (« *pipe* », c'est à dire « tuyau » en anglais) qui sert à connecter la sortie de la commande *echo* sur l'entrée de la commande *perl*.

```
achille@gnome:~/Bureau/TDCORPUS$ echo "Ceci est un test." | perl utf8-tokenize.perl
Ceci
est
un
test
.
```

Ici, le résultat n'est pas très intéressant : on a un mot par ligne. C'est certes le cas le plus fréquent, mais le texte *C'est aujourd'hui.* devrait offrir un résultat plus intéressant :

- `echo "C'est aujourd'hui." | perl utf8-tokenize.perl`

```
achille@gnome:~/Bureau/TDCORPUS$ echo "C'est aujourd'hui." | perl utf8-tokenize.perl
C'est
aujourd'hui
.
```

C'est tout de même curieux que *C'est* soit tokenisé comme un seul mot… En fait, il faut préciser au script de tokénisation qu'on est en français avec l'option -f.

- `echo "C'est aujourd'hui." | perl utf8-tokenize.perl -f`

```
achille@gnome:~/Bureau/TDCORPUS$ echo "C'est aujourd'hui." | perl utf8-tokenize.perl -f
C'
est
aujourd'hui
.
```

On se retrouve alors avec une tokénisation qui ressemble plus à ce qu'on attend pour du français.

Mais ce n’est pas tout ! Essayons de tokéniser *Rendez-vous aujourd’hui*.

- `echo "Rendez-vous aujourd'hui." | perl utf8-tokenize.perl -f`

```
achille@gnome:~/Bureau/TDCORPUS$ echo "Rendez-vous aujourd'hui." | perl utf8-tokenize.perl -f
Rendez
-vous
aujourd'hui
.
```

Pas terrible…

Pour dire au tokéniseur de ne pas couper certains mots, on peut utiliser un fichier « d’abréviation », qui contient souvent bien autre chose que des abréviations (jetez donc un œil au fichier *french-abbreviations*).

- `echo "Rendez-vous aujourd'hui." | perl utf8-tokenize.perl -f -a french-abbreviations`

```
achille@gnome:~/Bureau/TDCORPUS$ echo "Rendez-vous aujourd'hui." | perl utf8-tokenize.perl -f -a french-abbreviations
Rendez-vous
aujourd'hui
.
```

C’est mieux !

Pour terminer, nous allons tokéniser un texte entier ! Notez que cette fois on n’utilise plus de *pipe*, mais qu’on indique le nom du fichier à analyser comme dernier paramètre de la commande *perl* :

- cat `exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations`

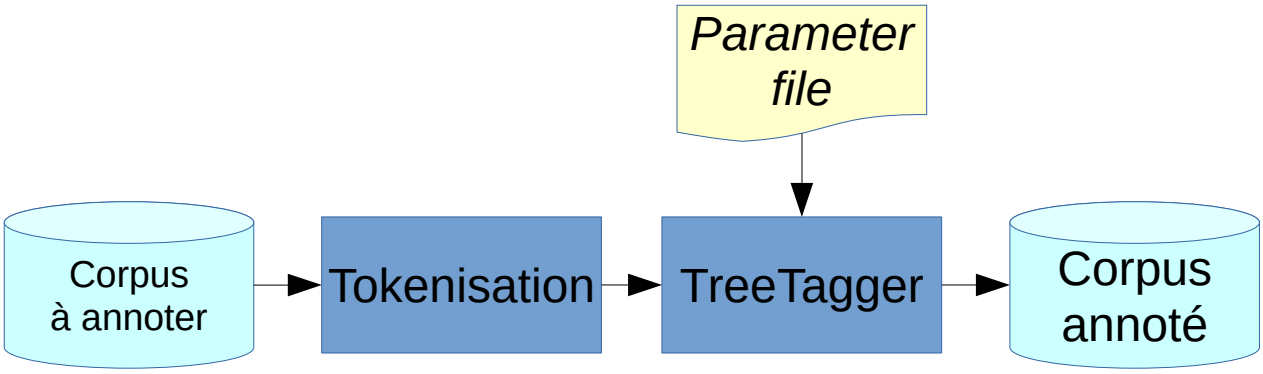
Cela donne une très longue liste de tokens, qui se termine par :

```
,
et
dont
il
ne
restait
plus
qu'
un
morceau
de
granit
battu
par
les
lames
du
Pacifique
,
tombe
de
celui
qui
fut
le
capitaine
Nemo
.
```

Ce sera tout pour la tokénisation. Ce script accepte aussi les options -e pour l’anglais, -i pour l’italien, et -z pour le galicien.

Quoi qu'il en soit, il faut bien comprendre qu'il tokénise d'une certaine manière, et qu'un autre script pourrait tokéniser différemment. Il s'agit ici d'une tokénisation pensée pour le modèle de langage *french.par*.

Annoter



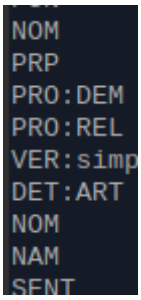
Les commandes données ci-dessous sont pour MacOS/Linux. Pour Windows il faut remplacer `./tree-tagger` par `tree-tagger.exe` . À part ça tout est identique.

Si votre ordinateur est lent, l'annotation automatique peut prendre un peu de temps… Pour ce TD, vous pouvez parfaitement ouvrir le fichier exemple.txt avec un Bloc-notes et effacer tout sauf la dernière ligne ! Ce sera moins « réaliste », mais ce sera plus rapide !

Maintenant, nous allons essayer de transférer cette longue liste de tokens à TreeTagger. Pour cela, nous allons à nouveau utiliser un *pipe*. Notez que la commande ./tree-tagger a aussi un paramètre : le nom du *parameter file* (modèle de langage) qu'on veut utiliser, ici *french.par*.

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger french.par`

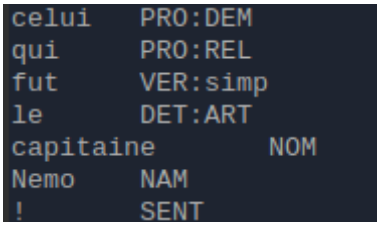
La commande nous sort une très longue liste de POS ; vous pouvez vérifier, elles correspondent au texte qu'on a tokénisé à l'étape précédente.



Les étiquettes utilisées sont documentées dans:
https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french-tagset.html

Maintenant, pour afficher quelque chose de plus lisible, on va ajouter un paramètre -token à la commande ./treetagger. Notez qu'elle se place avant le nom du modèle de langage :

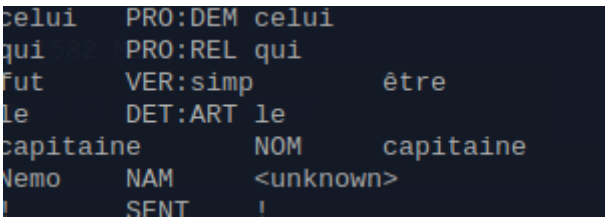
- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token french.par`



Mais on peut aussi afficher une troisième colonne, avec un lemme ! Pour cela, il faut utiliser l'option -lemma , en plus de l'option token :

- Cette fois je ne vous donne pas la commande ! À vous de trouver ! Mais si vous avez compris l'étape précédente avec -token, ça ne devrait pas vous poser de problème pour ajouter -lemma [solution à la fin du TD].

On obtient alors trois colonnes : token, POS et lemme :



Notez le token *Nemo*, qui n'est pas reconnu par le modèle de langage (il n'était pas dans le dictionnaire que TreeTagger a appris). Ici, TreeTagger a tout de même réussi à l'identifier grâce au contexte (DET:ART — NOM — ? — SENT), et grâce au préfixe (une majuscule) qui a permis de trouver qu'il s'agissait d'un NAM (nom propre).

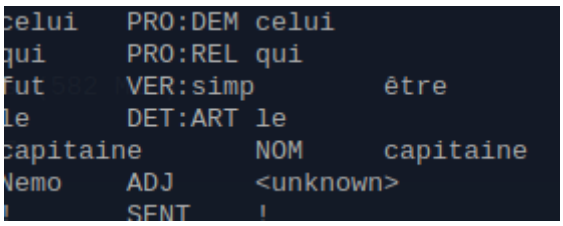
Dans les entrailles de TreeTagger

Nous allons nous pencher un peu sur cette identification de *Nemo*.

Déjà, comme je l'ai dit plus haut, TreeTagger s'est aidé du préfixe… Que se serait-il passé sans cette aide ? TreeTagger aurait uniquement regardé le contexte. Et dans la séquence de POS DET:ART — NOM — ? — SENT, à votre avis, quelle est la POS la plus probable ?

On va regarder ce qu'en pense TreeTagger en utilisant l'option *-ignore-prefix* qui désactive l'utilisation des préfixes :

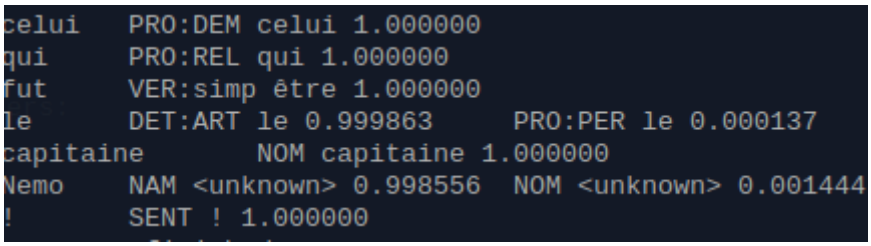
- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma -ignore-prefix french.par`



Cette fois on trouve un ADJ ! Ça semble logique : dans ce contexte, sans tenir compte du préfixe, c'est effectivement le plus probable.

Creusons encore un peu… avec les options *-threshold n* (qui affiche toutes les possibilités dont la probabilité est supérieure à *n*, *n* étant compris sur]0,1]) et *-prob*, qui affiche les probabilités (idem, sur]0,1]). Nous allons utiliser un *threshold* très très bas, à 0.0000001, pour bien voir ce que TreeTagger a dans le ventre. Cette fois on n'utilise pas l'option *-ignore-prefix* : on veut voir ce qui se passe dans le cas normal où TreeTagger utilise le préfixe.

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma -prob -threshold 0.0000001 french.par`



On voit que TreeTagger hésite un peu sur certains tokens, mais qu'il est quand même très sûr de lui : il donne des probabilités supérieures à 99 % sur chaque token.

Remontez quand même quelques tokens au-dessus dans votre terminal (la capture d'écran ne va pas jusque là) ; il y a quand même des cas où il est moins sûr de lui n'est-ce pas ? Regardez par exemple *tombe*. Étant donné le contexte, la (relative) perplexité de TreeTagger n'est pas étonnante n'est-ce pas ? Heureusement, il utilise un contexte de plusieurs tokens de chaque côté, ce qui lui permet d'éviter les erreurs les plus grossières.

Maintenant, si on reprend l'exemple sans les préfixes, on voit qu'il a quand même de gros doutes sur *Nemo* comme ADJ. Il envisage d'ailleurs que ce puisse être un NAM, mais avec une probabilité faible (~12,97%).

- `exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma -prob -threshold 0.0000001 -ignore-prefix french.par`

```
celui   PRO:DEM celui 1.000000
qui     PRO:REL qui 1.000000
fut     VER:simp être 1.000000
le      DET:ART le 0.999863     PRO:PER le 0.000137
capitaine       NOM capitaine 1.000000
Nemo    ADJ <unknown> 0.682670  NOM <unknown> 0.187619  NAM <unknown> 0.129711
!       SENT ! 1.000000
        finished.
```

Pour terminer sur les probabilités, essayons avec une phrase vraiment ambiguë : *La petite ferme la porte*. Notez que cette fois, on ne donne plus un fichier à « manger » au script de tokenisation, mais juste une phrase ; donc la commande ressemble à celles qu'on avait écrites au début de l'exercice sur la tokénisation. Il y a tellement de possibilités que je passe le *threshold* à 0.001 (vous pouvez essayer +, moi je suis limité par la largeur de la page ;-).

- `echo "La petite ferme la porte." | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma -prob -threshold 0.001 -ignore-prefix french.par`

```
La      DET:ART le 0.999798
petite  ADJ petit 0.859636      NOM petit 0.140364
ferme   ADV ferme 0.803220      VER:pres fermer 0.102970        ADJ ferme 0.067877      NOM ferme 0.015552      VER:subp fermer 0.010283
la      DET:ART le 0.997756     PRO:PER la 0.002238
porte   NOM porte 0.743456      ADJ porte 0.253930      VER:pres porter 0.002575
.       SENT . 1.000000
```

Ici, TreeTagger fait une erreur n'est-ce pas ? Mais il faut dire que la phrase n'est pas facile !

Varier les modèles

À présent, nous allons essayer d'analyser ce texte avec d'autres modèles. Vous vous souvenez comment analyser le fichier exemple.txt en affichant les tokens, POS et lemme n'est-ce pas ? (essayez de le faire sans relire le début de cette partie !)

Essayons en utilisant *Modèle_Presto-2014-10-31.par* à la place de *french.par*.

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma french.par`

```
tombe   Vvc     TOMBER
de      S       DE
celui   Pd      CELUI
qui     Pr      QUI
fut     Vuc     ÊTRE
le      Da      LE
capitaine       Nc      CAPITAINE
Nemo    Nc      <unknown>
!       Fs      !
```

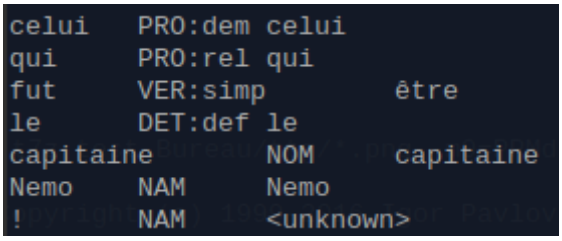
On voit qu'il y a pas mal de différences :

- Les POS sont totalement différentes (normal, le corpus d'apprentissage ne contenait pas les mêmes POS).
- Les lemmes sont en majuscule (normal, dans le dictionnaire d'apprentissage, tous les lemmes étaient en majuscule).

- Et aussi un peu plus d’erreurs (normal pour un modèle entraîné sur du français des XVIe-XVIIIe siècles !).

Enfin, avec le modèle de langage du français parlé :

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma spoken-french.par`



Le jeu d’étiquettes du modèle de langage ressemble plus à celui qui est fourni avec TreeTagger pour l’écrit, les lemmes aussi… Mais il y a plein d’erreurs sur la ponctuation. Pourquoi ? Parce que le corpus d’apprentissage et son dictionnaire sont constitués de transcription de dialogues… dans lesquels il n’y a pas de ponctuation !

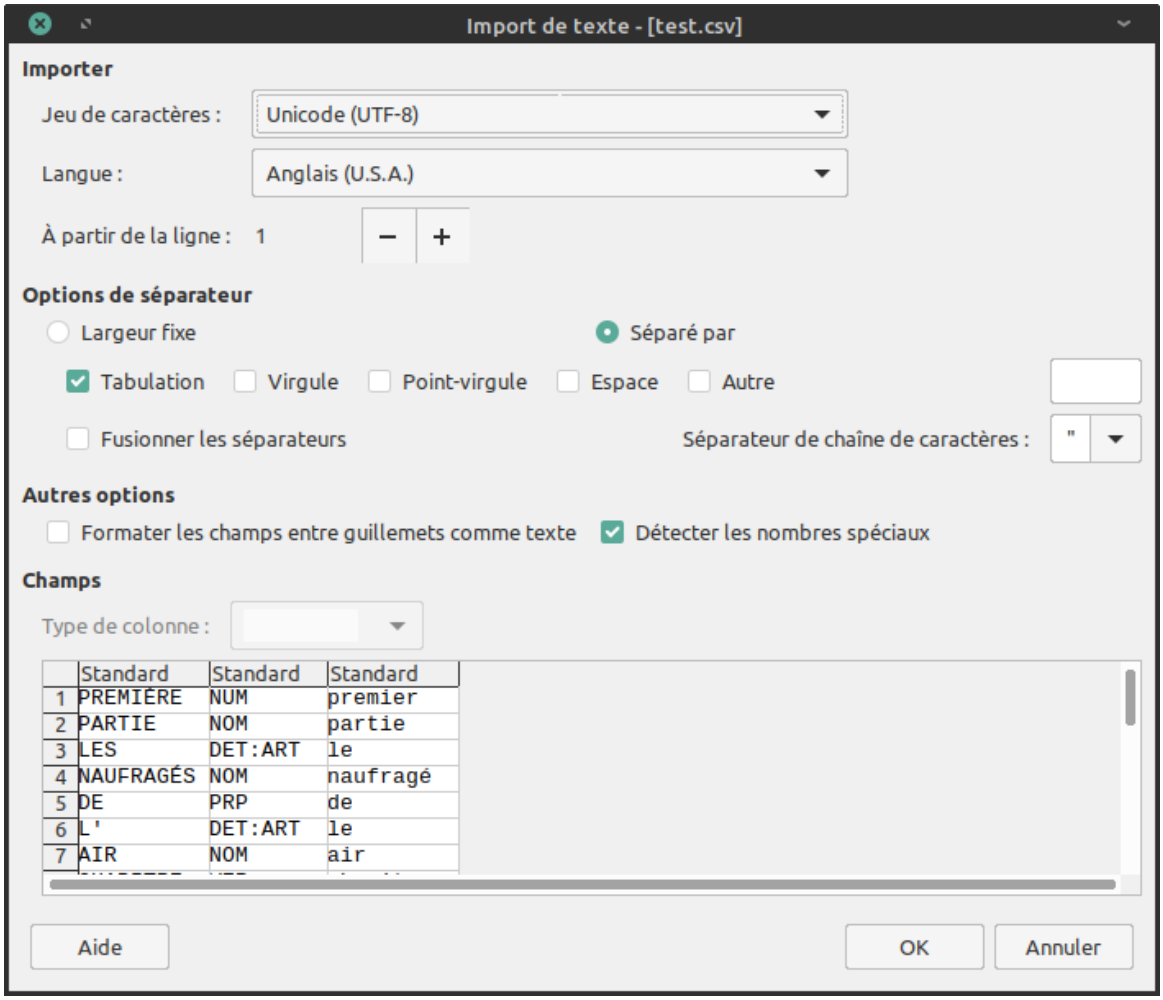
Sauvegarder dans un fichier

Pour terminer, nous allons voir comment sauvegarder nos résultats dans un fichiers. Pour cela, il suffit d'ajouter > *nomDuFichier* à la fin de la commande.

Par exemple :

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma french.par > Test.csv`

Cela donne un fichier CSV, qui peut se lire sous LibreOffice. Quand on l'ouvre, LibreOffice demande de lui préciser quel est le séparateur. En général, il devine assez bien ; ici, il s'agit de la Tabulation (cocher Tabulation, et décocher tous les autres séparateurs). Il faut aussi indiquer le jeu de caractères : Unicode (UTF-8).



Sous LibreOffice, on se retrouve alors avec un fichier que l'on peut facilement lire, et éditer !

	A	B	C
1	PREMIÈRE	NUM	premier
2	PARTIE	NOM	partie
3	LES	DET:ART	le
4	NAUFRAGÉS	NOM	naufragé
5	DE	PRP	de
6	L'	DET:ART	le
7	AIR	NOM	air
8	CHAPITRE	VER:pres	chapitrer
9	I	NUM	I
10	L'	DET:ART	le
11	ouragan	NOM	ouragan
12	de	PRP	de
13	1865	NUM	@card@
14	.	SENT	.
15	-	PUN	-
16	Cris	NOM	cri
17	dans	PRP	dans
18	les	DET:ART	le
19	airs	NOM	air

On verra la semaine prochaine comment, justement, éditer ce genre de fichier et le redonner « à manger » à TreeTagger.

Commandes à retenir

Assurez-vous que vous comprenez bien les commandes suivantes, et que vous sauriez les refaire. Manipulez-les un peu, en faisant varier les options ! Pour l'examen, vous aurez du temps, vous aurez droit aux documents, mais vous aurez sans doute quelques commandes à taper.

- `echo "Une phrase." | perl utf8-tokenize.perl -f -a french-abbreviations`
- `perl utf8-tokenize.perl -f -a french-abbreviations unfichier.txt`
- `echo "Une phrase." | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger french.par`
- `cat unfichier.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger french.par`

Pour TreeTagger, nous avons vu les options :

- -token
- -lemma
- -prob
- -threshold n
- -ignore-prefix

Solutions

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma french.par`

Pour aller plus loin

- Le cours de Franck Sajous :
  - http://fsajous.free.fr/SDL/SL0720X/treetagger/SL0720-sajous-observationsTT.pdf
- Mon cours à l'UNIL (une bonne partie de ce TD est basée dessus)
  - https://pro.aiakide.net/cours/L2016/Diapos.pdf